Relational Data Embeddings for Feature Enrichment with Background Information

Alexis Cvetkov-Iliev, Alexandre Allauzen, Gaël Varoquaux

Noname manuscript No.
(will be inserted by the editor)

Relational Data Embeddings for Feature Enrichment with Background Information

Alexis Cvetkov-Iliev · Alexandre Allauzen · Gaël Varoquaux

September 13, 2022

Abstract For many machine-learning tasks, augmenting the data table at hand with features built from external sources is key to improving performance. For instance, estimating housing prices benefits from background information on the location, such as the population density or the average income. However, this information must often be assembled across many tables, requiring time and expertise from the data scientist. Instead, we propose to replace human-crafted features by vectorial representations of entities (*e.g.* cities) that capture the corresponding information. We represent the relational data on the entities as a graph and adapt graph-embedding methods to create feature vectors for each entity. We show that two technical ingredients are crucial: modeling well the different relationships between entities, and capturing numerical attributes. We adapt knowledge graph embedding methods that were primarily designed for graph completion. Yet, they model only discrete entities, while creating good feature vectors from relational data also requires capturing numerical attributes. For this, we introduce KEN: Knowledge Embedding with Numbers. We thoroughly evaluate approaches to enrich features with background information on 7 prediction tasks. We show that a good embedding model coupled with KEN can perform better than manually handcrafted features, while requiring much less human effort. It is also competitive with combinatorial feature engineering methods, but much more scalable. Our approach can be applied to huge databases, creating general-purpose feature vectors reusable in various downstream tasks.

Keywords feature engineering, feature enrichment, knowledge graph embedding

A. Cvetkov-Iliev
Soda, INRIA Saclay, 1 Rue Honoré d'Estienne d'Orves, 91120 Palaiseau – France E-mail: alexis.cvetkov-iliev@inria.fr

A. Allauzen
ESPCI Paris, 10 Rue Vauquelin, Paris, 75005, France

G. Varoquaux
Soda, INRIA Saclay, 1 Rue Honoré d'Estienne d'Orves, 91120 Palaiseau – France

1 Introduction

For machine learning on data tables, a data scientist may encounter columns with many different discrete entries or entities, for instance cities in a housing price prediction setting (Fig. 1a). These city names can be encoded as a categorical variable, but generalizing to housing in a new city is then impossible. A good solution for such columns is often to use external sources to bring in information: the GPS coordinates of the cities, the population, the average income (Fig. 1b)... From a data-science perspective, this requires feature engineering on relational data: merging and aggregating information across data sources to create an enriched table with extra features (Fig. 1c). In practice however, such feature engineering is difficult and time consuming for the human analyst, because it requires a good understanding of both the different data sources and the application domain. For instance the number of wealthy people living in a city may be important, but estimating it may require crossing information across many tables to build a single somewhat abstract indicator. In fact, it is often recognized that data preparation is one of the biggest bottlenecks of data-science [2, 23].

A specificity of learning across a complex relational structure is that different entries come with very different information. For instance, when collecting information on local wealth in Wikipedia –querying DBPedia [27] or YAGO [28]–, a data scientist will find for *San Francisco* the GDP as well as many known individuals and companies. But for the neighboring locality *Muir Beach*, none of this is available. The data scientist may then need to dig information at the county level,

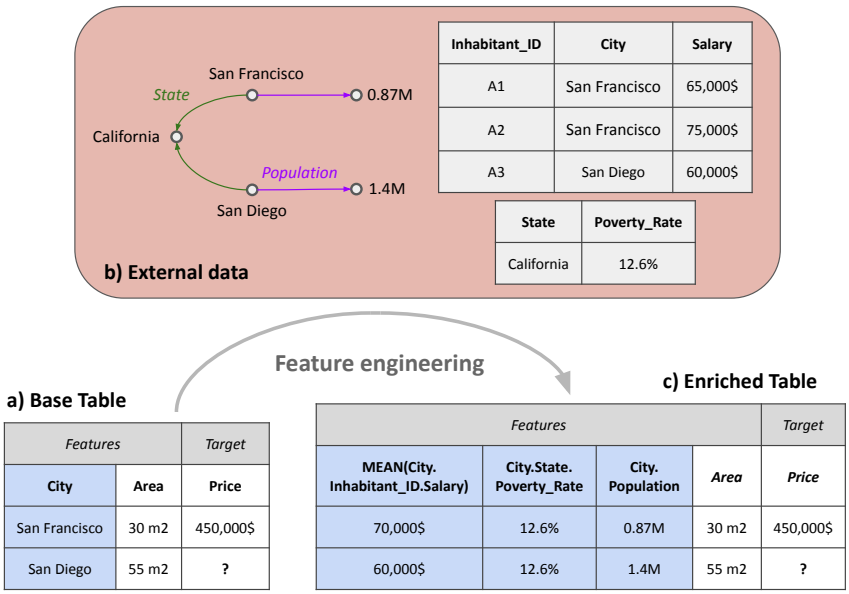


Fig. 1 The classical pipeline of feature enrichment. A base table **(a)** contains a target to predict and several features, including a categorical feature with discrete entities (here cities). To boost prediction performance, external data **(b)** about the entities of interest is incorporated into the base table –usually via tedious feature engineering– to obtain the enriched table **(c)**. The external data **(b)** can come under various formats, *e.g.* tables or multi-relational graphs.

which has a different set of attributes. The root of the challenge is that the original relational information is fundamentally irregular and cannot be represented to a learning algorithm as a fixed set of "features".

Our goal here is to make it very easy for the data scientist to enrich a feature with information from external data sources. Inspired by word embeddings [30] which brought a breakthrough to text processing by their ease of use, we strive to associate entities to general-purpose feature vectors that can be used in multiple downstream tasks. This requires a feature extraction method that captures well entity attributes, and is scalable enough to be used on large databases. For instance, a general-purpose knowledge-base such as YAGO3 [28] is a particularly useful source of data, with information on 75,000 cities; but it is huge: millions of entities and hundreds of attributes. Existing automatic feature engineering methods, such as Deep Feature Synthesis (DFS) [19], are combinatorial: they greedily join and aggregate entity attributes across tables to create feature vectors. Their combinatorial nature leads to tractability challenges: running DFS on YAGO3 produces very high dimensional vectors ($d \sim$ 10,000 – 140,000) which entail large storage costs and computational hurdles in downstream machine-learning tasks.

Instead, we propose to use embedding models that learn a static vector representation for each entity. Indeed, they provide compact representations that can encode knowledge about various entities into a fixed, low-dimensional space (*e.g.* $d = 200$). We learn these vectors from the external data, and add them to the base table as new features to enhance prediction performance. A pioneering work in this direction is RDF2vec [38] and its variants, which have been used to learn entity embeddings from multi-relational graphs for various downstream tasks [16, 41, 40, 43]. These works directly build on word-embedding tools developed for natural language –namely word2vec [30]. As such, they leverage *contextual* information: as *San Francisco* and *California* are connected in the graph they are related. However, they do not account for the nature of these relations, which requires modeling the *relational* information: Wikipedia specifies that *San Francisco* is in *California*, but *Sacremento* is the capital of *California*. We will see that capturing well this information is important to generate feature vectors for downstream analytic applications. Another, more general, drawback of embedding methods is that they are designed for discrete entities, and are less suited to capture numerical attributes. Yet these attributes are often useful for the end task: densely populated cities tend to exhibit high housing prices for instance.

We propose here an approach that addresses these two limitations and provide high-performance embeddings. To capture relational information, we rely on knowledge graph embedding models [47], widely used for graph completion but not studied for feature extraction purposes. In such models, embeddings are directly optimized to capture relationships between entities. We then introduce KEN (Knowledge Embedding with Numbers), a module that extends knowledge graph embedding models to numerical attributes. Finally, we conduct a thorough empirical evaluation of our approach, using entity embeddings to boost machine-learning performance in multiple tasks, and show that:

- Feature vectors obtained via knowledge graph embedding models perform much better than RDF2vec embeddings.
- Embeddings learned with KEN do capture numerical information, which greatly improves prediction performance in downstream tasks.

– A good embedding model coupled with KEN outperforms manually handcrafted features, while requiring much less human effort. It is also competitive with Deep Feature Synthesis, but is more scalable in terms of computation time, memory and size of the created features.
– Although designed for multi-relational graphs, simple heuristics allow our approach to be applied to tabular data, with good performance.

The rest of the paper follows as such: section 2 goes into depth explaining related work, section 3 details our contributed approach, and section 4 gives a thorough empirical study of approaches to create features from relational data.

2 Related work: extracting features from relational data

We focus here on two common data structures for data-science: tabular data, as in relational databases, and multi-relational graphs (a.k.a. knowledge graphs), the backbone of Linked Open Data [7]. We broadly refer to both as *relational data*. In this section we give an overview of various lines of work related to creating vectors from relational data, drawing from a variety of scientific communities.

2.1 The classic view: feature engineering

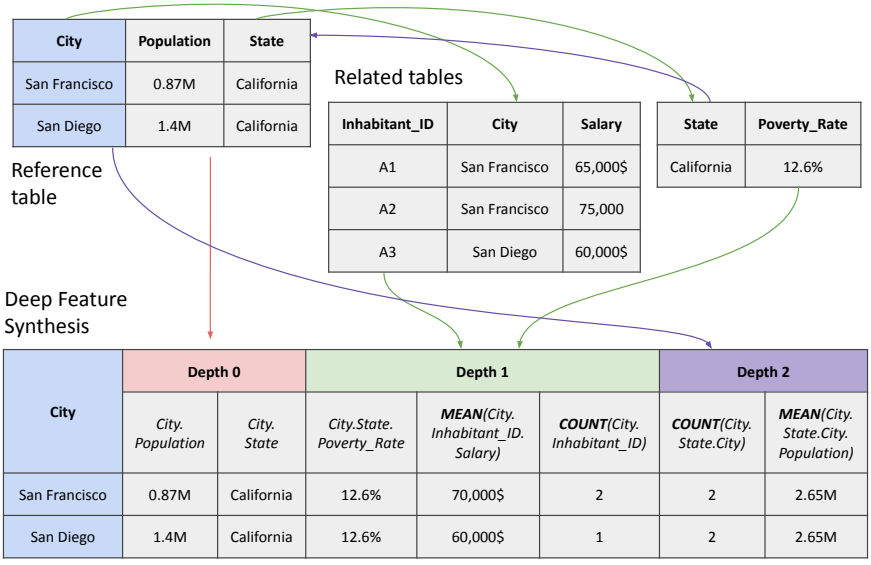
Manual feature engineering Feature engineering across multiple tables traditionally relies on a human analyst crafting SQL queries or dataframe operations, such as joins or aggregations, to build a single feature matrix. The problem is the same with Linked Open Data [32,39]: statistical studies require features extracted from the data, here coming as knowledge graphs rather than multiple tables. *Propositionalization* approaches used to mine knowledge graphs [20] tackle this by creating for each entity (node) of the graph a set of features, statistical fingerprints and aggregates of its neighbourhood [33,37]. Here again, manual crafting is needed to capture specific information such as wealth.

Whether it is done on tables or knowledge graphs, feature engineering is a time-consuming task: studies show that data scientists spend 60% or more of their time transforming the data for analysis [14]. Indeed, designing the right features often requires careful effort from the analyst: which information is relevant for the task at hand? How to query it? This is particularly difficult on large data sources. For instance, a knowledge graph representation of Wikipedia leads to hundreds of entity classes described by thousands of attributes in DBPedia [27]. Exploring which joins are best for a given analysis is difficult even for an expert: How to assemble indirect signals that capture information on the question at hand, for instance estimating the distribution of wealth in a locality.

Automated feature engineering A few approaches have been proposed to automate the construction of queries for feature engineering on relational databases. A fundamental challenge is that assembling such multi-table data transformations calls for discrete choices –*e.g.* to join, or not to join?– with combinatorial possibilities that explode on large databases. For instance, Deep Feature Synthesis (DFS) [19] is a greedy approach that denormalizes a database by chaining joins from one

reference table to all related tables and aggregates one-to-many relations using combinations of a small base of functions (see Fig. 2). Typical aggregation functions include `COUNT`, `MODE` (most common) for categorical features, and `MEAN`, `MIN`, `MAX`, `STD` for numerical features. A crucial parameter of DFS is the *depth*, which limits how many times joins can be chained to create new features. Higher depths capture a wider range of information and usually improve performance, but quickly result in very large feature vectors and computation times, as the number of possible join paths grows exponentially. This often calls for post-processing techniques to remove unpredictive or redundant features.

Subsequent works have improved over DFS by adding aggregation functions for other types of data (text, sequences) [25], for instance via recurrent neural networks [24]. Although powerful feature extractors, all these methods remain combinatorial in nature, and do not scale to large databases. Even with a limited depth, a large number of entities of different types leads to increasingly wide feature matrices with many missing values, as the different entities come with different sets of attributes. Finally, automated feature engineering methods present other drawbacks: the created features often contain categorical or missing values that must be encoded, and their interpretability (we can trace back the joins and aggregations needed to compute each feature) is challenged as their dimension quickly grows.



City	Population	State
San Francisco	0.87M	California
San Diego	1.4M	California

Inhabitant_ID	City	Salary
A1	San Francisco	65,000$
A2	San Francisco	75,000
A3	San Diego	60,000$

State	Poverty_Rate
California	12.6%

City	Depth 0		Depth 1			Depth 2	
	City. Population	*City. State*	*City.State. Poverty_Rate*	***MEAN**(City. Inhabitant_ID. Salary)*	***COUNT**(City. Inhabitant_ID)*	***COUNT**(City. State.City)*	***MEAN**(City. State.City. Population)*
San Francisco	0.87M	California	12.6%	70,000$	2	2	2.65M
San Diego	1.4M	California	12.6%	60,000$	1	2	2.65M

Fig. 2 An example of Deep Feature Synthesis. Starting from a reference table with entities of interest (here cities), new features are created by chaining joins to related tables, up to a certain depth = 2. To aggregate values from one-to-many relations (*e.g.* city inhabitants), we use the `MEAN` and `COUNT` operators, respectively for numerical and categorical features. Colored arrows indicate join paths across tables for each depth.

2.2 Entity embeddings in relational data

While entity embeddings come from a body of literature far from that of feature engineering, they also create feature vectors from relational data [26].

Prelude: word embeddings Many embedding methods for relational data take inspiration from word embeddings. By injecting discrete entities (words) in vector spaces, word embeddings have boosted statistical analyses of text. They rely on the distributional semantics idea, which can be summarized by Firth's sentence: "a word is characterized by the company it keeps". The central model is Skip-Gram with Negative Sampling (SGNS), used in word2vec [30]. Each word w is associated to an embedding $\boldsymbol{w} \in \mathbb{R}^p$[1]. SGNS learns these embeddings by optimizing similarities of pairs of words, using a *scoring function*:

Scoring function
$$f(w, w') = \boldsymbol{w} \cdot \boldsymbol{w'} \tag{1}$$

Given a text corpus, embeddings are optimized so that a word w is more similar to a word w' observed in the same context –*e.g.* the same sentence–, than another word $w^\dagger$ not in the context; minimizing a cross-entropy loss[2]:

SGNS
$$L = - \sum_{\substack{w,\ w' \in \text{context}(w),\\ w^\dagger \notin \text{context}(w)}} \log(\sigma(f(w, w'))) + \log(1 - \sigma(f(w, w^\dagger))) \tag{2}$$

After training, word embeddings capture *contextual* similarities: words with the similar contexts (neighbors) end up close in the embedding space.

2.2.1 Embedding entities in a table

Word embedding methods, such as SGNS, can be extended to other data structures by defining a corresponding notion of context [18]. In tables, a common choice is to view rows as sentences: two entities are in one another context if they appear in the same row. This was for instance applied to enable semantic queries over tables [9] and for automatic table completion and retrieval [50]. More recent work integrates intra-row and intra-column information to learn richer representations. Cappuzzo *et al.* [11] link entries of a table to the row and column nodes they belong to. Random walks through the resulting graph generate "sentences" of tokens, then fed to a SGNS model.

2.2.2 Embeddings entities in knowledge graphs

Knowledge graphs use a more general representation of relational data than tables. They replace the notion of columns by that of *relations*, which enables a uniform representation over many tables, and helps assembling information from multiple sources of data. Each piece of information is encoded as a triple (h, r, t), indicating a

[1] To be precise, two embeddings are learned for each word. Which one is used in the scoring function depends if we view it as the context word ($w \in \text{context}(w')$) or not ($w' \in \text{context}(w)$).

[2] This is actually a simplified version of the loss optimized by word2vec; *eg* it does not account for multiple negative examples.

certain *relation* r between the *head* and *tail* entities (h, t). Large knowledge graphs, such as YAGO3 [28] or DBPedia [27] contain millions or even billions of triples – *e.g.* (San Francisco, HasState, California) – and cover millions of entities.

Knowledge graph embedding models learn a vector for each entity (node) and relation (edge) of the graph. They have been mostly developed for two purposes, leading to two distinct lines of research [36]:

1) **Predicting new triples** of the knowledge graph for completion purposes, which has been the main application of knowledge graph embeddings.
2) **Providing feature vectors for downstream tasks** outside the knowledge graph, which received much less attention in the literature, but is our focus here.

Embeddings for downstream tasks RDF2vec [38] is a central work applying knowledge graph embeddings in external downstream tasks. It has been used to incorporate background information in various tasks: geospatial data analysis [16], recommender systems [41,40], or biomedical prediction tasks [43]. Given a knowledge graph, RDF2vec generates sequences of tokens by performing random walks on the graph, alternating between entities and relations (see Fig. 3). These sequences are then fed to a SGNS model to obtain embeddings for entities and relations. An important parameter is the *depth*, which limits the number of hops in the random walk, and thus the range of information to capture. A depth of 1 captures relationships between entities and their nearest neighbors in the graph, and so on... Similarly to Deep Feature Synthesis, a challenge is that the number of possible walks increases exponentially with depth. To avoid this, walks are often computed for certain entities of interest only, with a limited number of walks for each entity.

Since RDF2vec, most research efforts focused on the creation of walks, for instance giving more weight to relations/entities based on their frequency, PageRank or degree, removing rare entities, or allowing teleportations between entities that share similar properties [13,46].

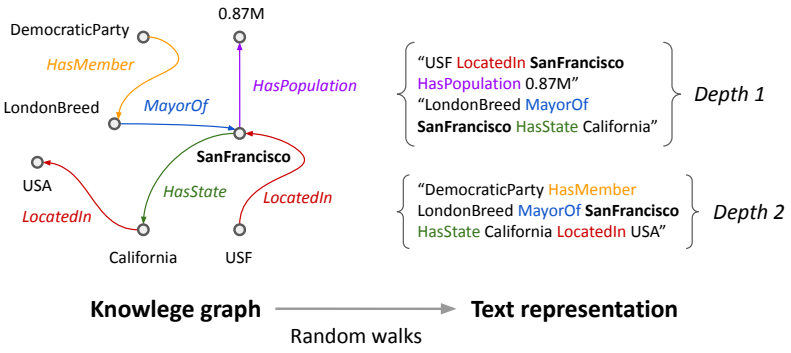


Fig. 3 Graph to text representation in RDF2vec. Random walks are performed on the knowledge graph to generate sentences of tokens. Often, walks are only computed for a subset of entities, here San Francisco. The depth parameter limits the number of hops in the random walk, either forward or backward.

Embeddings for graph completion Knowledge graph embeddings have been widely used for graph completion, either through *link prediction* (predicting the missing entity in an incomplete triple $(h, r, ?)$) or *triple classification* (predicting if a triple is True of False). Similarly to SGNS, these models define a scoring function $f(h, r, t)$ that represent the plausibility of a given triple (h, r, t). Embeddings are then optimized so that observed triples obtain high scores, while negative ones (typically sampled by corrupting the head or tail entity in observed triples) obtain low scores.

Scoring functions typically model the different relations between entities as geometrical operations in the embedding space. For instance, the seminal TransE model [10] represents a relation r as a translation vector $\boldsymbol{r} \in \mathbb{R}^p$ between entity embeddings $\boldsymbol{h}$ and $\boldsymbol{t}$:

$$\text{TransE} \qquad f(h, r, t) = -\|\boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t}\| \tag{3}$$

with $\|.\|$ a ℓ_1 or ℓ_2 norm. Given a knowledge graph $\mathcal{G}$, embeddings are trained to minimize a margin loss:

$$L = \sum_{\substack{(h,r,t)\in\mathcal{G},\\ (h',t') \text{ s.t.} (h',r,t')\notin\mathcal{G}\\ \text{with } h'=h \text{ or } t=t'}} [f(h', r, t') - f(h, r, t) + \gamma]_+ \tag{4}$$

Many models that improve upon TransE [47] focus on better modeling of one-to-many relationships and certain relational patterns (*e.g.* symmetry/antisymmetry, inversion, composition) [49, 44, 6]. For link prediction in knowledge bases, one of the best performing methods [3] is MuRE, Multi-Relational Poincare graph embeddings [6]. The key component of the method is the model of the link between head and tail entity (homologous to (3) for TransE):

$$\text{MuRE} \qquad f(h, r, t) = -d(\boldsymbol{\rho}_r \odot \boldsymbol{h}, \boldsymbol{t} + \boldsymbol{r}_r)^2 + b_h + b_t \tag{5}$$

where $\odot$ is the element-wise multiplication, two vectors $\boldsymbol{\rho}_r, \boldsymbol{r}_r \in \mathbb{R}^p$ represent the relation r, and the head and tail entities are represented by vectors $\boldsymbol{h}, \boldsymbol{t} \in \mathbb{R}^p$ and biases $b_h, b_t \in \mathbb{R}$. d is the Euclidean distance[3]. The model is optimized by sampling positive and negative triples (as in (4), but using a logistic loss (2) instead).

Structure of contextual vs relational embeddings Approaches based on SGNS such as RDF2vec only capture *contextual* information, while much progress in knowledge graph embedding has focused on modeling different types of relations separately. As a consequence they induce very different neighborhood structures on entities embeddings.

Contextual embeddings, as RDF2vec, are trained on "sentences" of tokens, where each entity is surrounded by the relations and entities it co-occurs with in triples (Fig. 3). Two entities end up close in the embedding space if they have similar contexts: *1)* They may share a relation, but not necessarily with the same entity, *e.g.* (San Francisco, **LocatedIn**, California) and (Paris, **LocatedIn**, France). This tend to group entities of the same type, since entities of different nature, like people and cities, share few relations. *2)* They may share a connection to a common entity, but not necessarily via the same relation, *e.g.* (MathWorks,

[3] MuRE can also use the Poincaré non-Euclidean geometry. However in practice [6] the Euclidean version is an excellent performer, as good as the non-Euclidean one for $p \geq 150$.

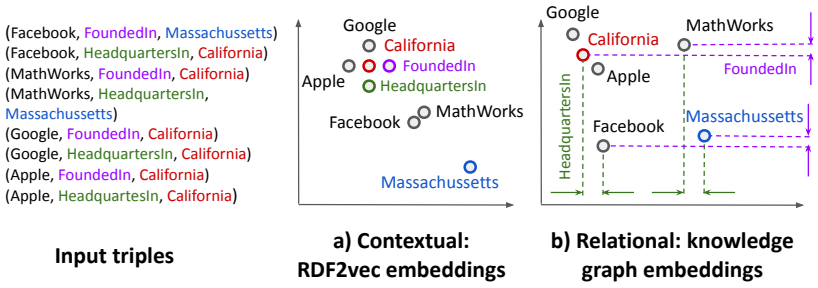


Fig. 4 What drives entity neighborhoods in embedding space? a) Contextual embeddings (as RDF2vec) ignore the nature of the relation: given information on states in which companies have been founded and have their headquarters, it cannot differentiate *Facebook* (born in Massachussetts, moved to California), from *MathWorks* (born in California, moved to Massachussetts). **b)** Knowledge graph embeddings models can give rise to different geometric constraints for these two relations, separating out the companies. For instance here a relation is encoded with a projection.

FoundedIn, **California**) and (Nevada, HasBorderWith, **California**). Fig. 4a gives a paradigmatic example: such contextual information is blind to the difference between Facebook, founded in Massachussetts but headquartered in California, and MathWorks, founded in California but headquartered in Massachussetts.

Knowledge graph embeddings using the relation type in the scoring function between two entities create a very different structure in the embedding space. As relations of different nature lead to different transformations of the embedding space, they each "pull" entities in different directions. In addition, modern models can learn transformations that are not one-to-one –non bijective–, better suited to many-to-one relations, as when many cities are located in the same state. As a result the different relations can be encoded separately in the entities embeddings, for instance along different coordinates (Fig. 4b).

Integrating numerical attributes in embeddings Numerical attributes, such as city populations, are poorly handled by most embedding methods. They are often simply dismissed, or at best binned and treated as discrete entities [11], which remains suboptimal as it does not capture the topology of numbers.

Recent knowledge graph embedding models address this issue [17]. TransEA [48] adds a loss to reconstruct numerical values from embeddings with a linear model. LiteralE [21] is a state-of-the-art approach where each entity i is represented by two vectors: $\boldsymbol{e}_i \in \mathbb{R}^p$ representing the entity itself, and $\boldsymbol{l}_i \in \mathbb{R}^q$, $\boldsymbol{l}_i$ containing each of its numerical attribute (0 if no value, and where q is the number of numerical relations in the KG). When used in the scoring function, embeddings $\boldsymbol{h}$ and $\boldsymbol{t}$ are constructed with a function g that combines the two vectors into a single one: $\boldsymbol{h} = g(\boldsymbol{e}_h, \boldsymbol{l}_h)$, and $\boldsymbol{t} = g(\boldsymbol{e}_t, \boldsymbol{l}_t)$, both in $\mathbb{R}^p$. LiteralE implements g as a learnable mechanism similar to gated recurrent units.

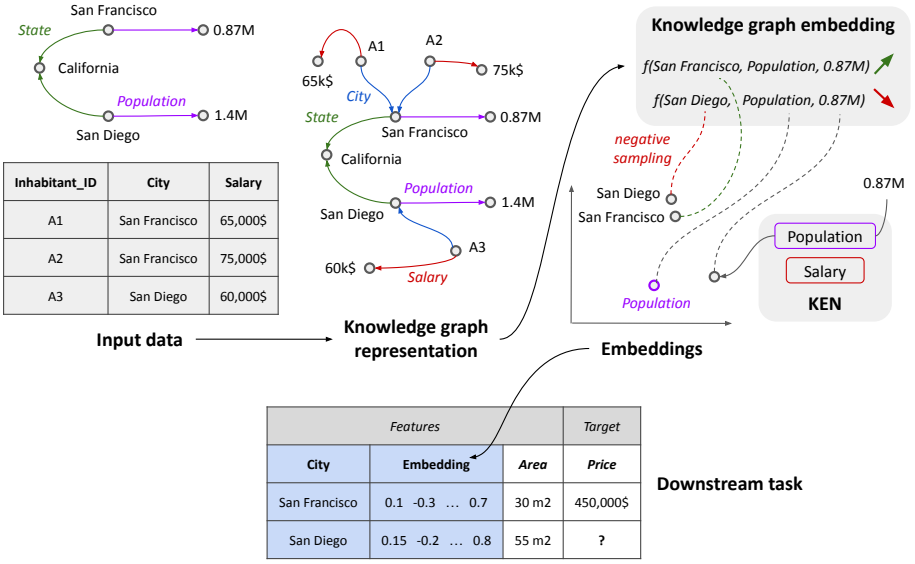


Fig. 5 Our pipeline for automatic feature extraction from relational data. 1) The input data, which may contain tables, is transformed into a knowledge graph. 2) We use a knowledge graph embedding model to learn a vector for each entity, and leverage numerical values by embedding them in the same space as other entities with KEN. 3) After training, entity embeddings can be easily added as new features in downstream tasks.

3 Contribution: multi-relational embeddings that capture numbers

We introduce here our approach to automatically extract information from relational data, creating feature vectors that can be used in downstream tasks. It relies on 3 key ingredients, that we describe in the following subsections:

1) Using knowledge graph embedding models designed for graph completion, as opposed to RDF2vec, to capture well *relational* information.
2) KEN (Knowledge Embedding with Numbers), a module that extends knowledge graph embedding models to numerical attributes.
3) Representing tables as knowledge graphs, to leverage them in our approach.

Fig. 5 summarizes our pipeline for automatic feature extraction from relational data.

3.1 Relational rather than contextual embeddings to encode information

With our goal of creating embeddings as features for downstream tasks, we motivate here the importance of using *relational* embeddings, originally designed for knowledge graph completion, rather than *contextual* RDF2vec-like models, traditionally used to extract features for downstream tasks.

From a big picture perspective, given an entity h of interest (*e.g.* a city), we would like an embedding $\boldsymbol{h}$ that encodes as well as possible the information related to h in the data. At the very least, it implies representing well the various

relationships h has to other entities (*e.g.* its state), to make them available to the machine-learning model used in the downstream task. Representing not only the related entity t but also the nature of the relation r is often important: knowing whether a person A is the mother, the sister, or the daughter of a person B informs on the age difference.

In contextual embeddings such as RDF2vec, the presence of a link between a entity h to another entity t is modeled somewhat independently from the nature r of the link, *i.e.* the type of the relation. Indeed, the scoring function used in SNGS –eq (1)– is only applied to pairs (h, t), (h, r) and (r, t). Structure between h, r, and t is created indirectly as they appear in the same context.

In contrast, relational embeddings developed for knowledge graph embeddings use a scoring function involving h, r, and t jointly. As this scoring function is minimized for triples in the graph, it induces algebraic relations between the corresponding embeddings: for TransE $\boldsymbol{t} \approx \boldsymbol{h} + \boldsymbol{r}$, or for MuRE $\boldsymbol{t} \approx \boldsymbol{\rho}_r \odot \boldsymbol{h} - \boldsymbol{r}_r$. These algebraic relations imply that $\boldsymbol{t}$ captures the link to $\boldsymbol{h}$ in a way that is specific to r and hence a downstream analysis model can recover this specific information, *e.g.* selecting on the mother, and not all relatives.

Fig. 4 illustrates the specificity of the link: for RDF2vec the relations are encoded as vectors which lie in the middle of the embeddings of the entities while a knowledge graph embedding encodes the relations as a transformation of these vectors (here a projection), and allows the different relations to be expressed on different coordinates of the vectors.

3.2 Capturing numerical attributes with KEN

Numerical attributes are omnipresent in relational data, and often contain precious information for downstream tasks, *e.g.* a city's wealth influences housing prices. While they are readily-available as numbers, the irregular nature of the information prevents from merely adding them as coordinates to the feature vectors. A first challenge is that different entities have different numerical attributes. A more serious one arises when aggregating numerical information across many-to-one relations: there are many ways of doing so. For instance, to characterize wealth in a county from the GDP of its cities, the mean, the Gini index, the percentiles, *etc.* are all useful aggregates. As a result, Deep Feature Synthesis generates more than 2,000 features derived from numerical attributes for cities in YAGO3.

We strive for lower-dimensional representations, and thus aim to capture numerical information in entity embeddings. However, embedding methods are formulated in terms of discrete elements (sec. 2.2): words, entities. A naive way to adapt them to numerical attributes would be to consider numbers as tokens and learn an independent embedding for each value. Yet doing so discards the topology underlying those numbers: close numerical values should have similar representations. Binning values before embedding reduces this effect, but remains suboptimal. To tackle this, we introduce here KEN (Knowledge Embedding with Numbers), a module that adapts embedding models to numerical attributes.

The KEN module Entity-embedding approaches can be seen as relying on a linear encoder to associate an entity h with its vector representation $\boldsymbol{h} \in \mathbb{R}^p$. In this

light, we propose to inject numerical values in the same vector space also with an encoder, learning a function $\boldsymbol{e} : \mathbb{R} \to \mathbb{R}^p$ that maps numerical values to embeddings.

We use as function a single-layer neural network with a ReLU activation to embed numerical values. To embed different types of attribute separately (e.g. city populations and GPS coordinates), we learn a function $\boldsymbol{e}_r$ for each attribute r:

$$\boldsymbol{e}_r(x) = \text{ReLU}(x\,\boldsymbol{w}_r + \boldsymbol{b}_r) \tag{6}$$

with $x \in \mathbb{R}$ the numerical value to embed, and $\boldsymbol{w}_r, \boldsymbol{b}_r \in \mathbb{R}^p$ the weights and biases of the linear layer. Embeddings $\boldsymbol{e}_r(x)$ of numerical values can then be used in place of tail embeddings $\boldsymbol{t}$ in the scoring function $f(h, r, t)$.

Comparison with other methods capturing numerical attributes An asset of KEN is that it comes with no hyper-parameters to tune. This is unlike TransEA [48], where the importance of numerical attributes must be controlled, with the danger that the optimal value might differ for each attribute. Another important difference with TransEA is that KEN can capture non-linear interactions between entities and numerical attributes, thanks to the ReLU activation. For instance, cities in California are associated to latitudes between 32° N and 41° N which cannot be expressed by a mere threshold on a linear representation.

Importantly, KEN uses numerical values x during the training as new triples (h, r, x) to be predicted, which forces entity embeddings to capture these numerical attributes. This is different from LiteralE [21], where numerical values are incorporated to entity embeddings to better predict non-numerical triples (h, r, t). LiteralE therefore only captures the information in numerical values useful to triangulate other entities, and not the values in themselves. In particular non discriminant numerical attributes can be discarded by the gate mechanism. As an extreme example, an entity linked to numerical attributes but not to other entities will not be embedded in LiteralE, as there is no training data.

In contrast, KEN draws no major distinction between discrete entities and numerical values: they are embedded in the same space. Each type of numerical attribute is associated to a specific relation and thus embedded on a specific line segment via eq. (6). An analytic model for a downstream task can extract this information, proceeding in a similar way as with discrete information (as described in 3.1). The numerical attributes that an entity has and its relations to other entities may contribute to create similar neighborhood structures: for a city to be *locatedIn California* is equivalent to its GPS coordinate taking specific value ranges.

Making the architecture robust to attribute distribution One challenge of heterogeneous data is that different numerical attributes have very different distributions.

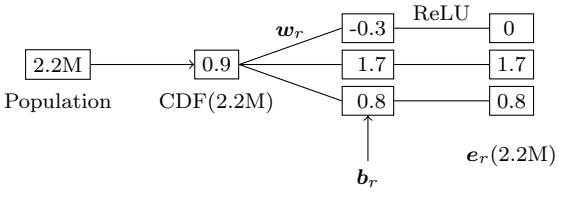


Fig. 6 Embedding numerical values with KEN.

We normalize numerical values $x \in \mathbb{R}$ to the interval $[0, 1]$ before embedding them. With neural networks, a common way to do so is "min-max" normalization: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$. However it is problematic when dealing with heavy-tailed distributions, such as city populations. Indeed, after normalization, most values x' will be very close to zero and have similar representations $\boldsymbol{e}_r(x') \simeq \text{ReLU}(\boldsymbol{b}_r)$. This makes it difficult for instance to distinguish a village with 1 000 inhabitants from a medium-sized town of 10 000 people.

Ideally, we would like the values x' to be evenly distributed in $[0, 1]$, to separate as well as possible their embeddings. We achieve this with quantile normalization, which maps numerical values to their quantile in the attribute distribution, using an empirical estimate of the cumulative distribution function: $x' = \text{CDF}(x)$.

Fig 6 summarizes the complete picture of numerical value embedding with KEN.

3.3 Representing tables as knowledge graphs

To create embeddings with rich semantics, the source data must contain as much detail as possible about the entities under study. This often requires to leverage data from different sources, for instance combining broad but shallow information (*e.g.* city populations) from large knowledge graphs with more granular data

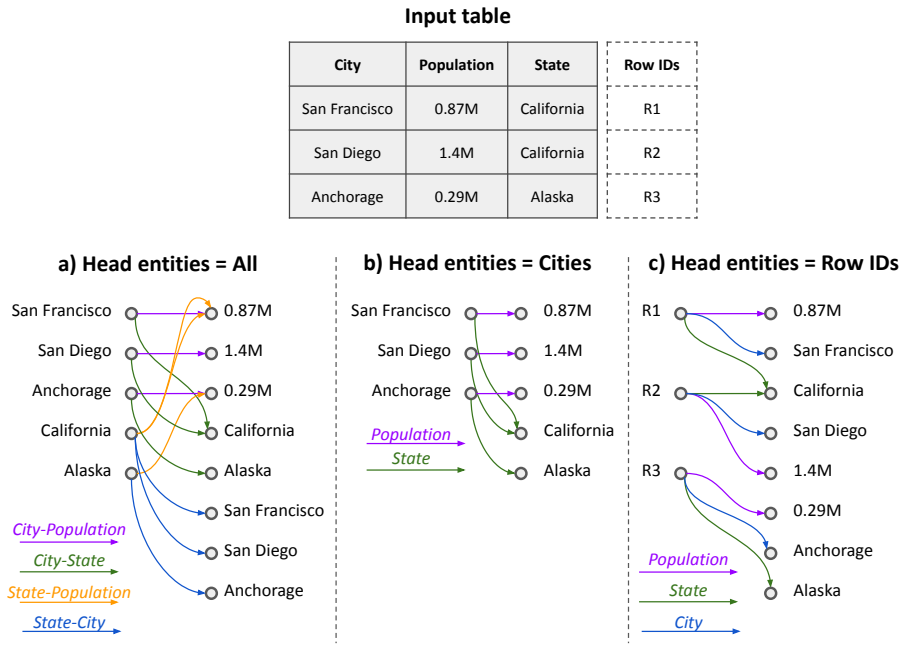


Fig. 7 Representing tables with triples. For each row of the table, we generate triples by linking its entries through different relations. The methods we present here differ on their choice of head entities when building triples: **a)** using all discrete entries as heads **b)** using only the entities of interest (generally from the same column) and **c)** introducing a "row id" entity for each row and using it as head entity.

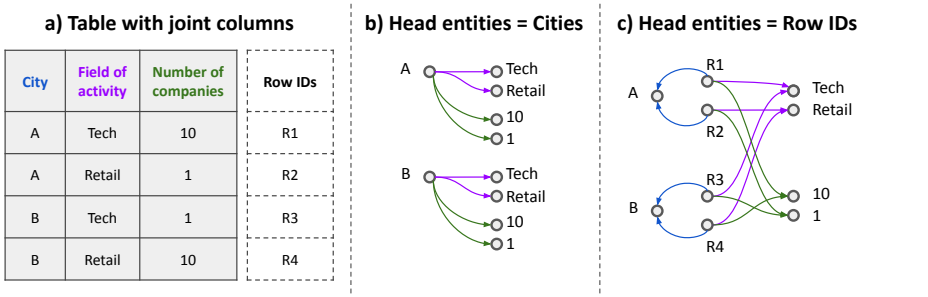


Fig. 8 Capturing joint information across columns. a) A table describing cities with two joint attributes that must be considered together to be meaningful. **b)** Using cities as head entities encodes the two attributes separately, hence we cannot differentiate them from their triples. **c)** Introducing row entities allows to capture all attributes jointly and distinguish the two cities.

(*e.g.* recent house prices at the neighbourhood-level) from domain-specific tables. Although our approach inputs knowledge graphs (*i.e.* triples (h, r, t)), this representation is general enough to easily encode information from other data structures. We focus here on tabular data, and explore a few strategies to represent tables as knowledge graphs.

The core idea to generate triples from tables is to link entities from the same rows with different relations. For instance, an exhaustive strategy consists in building all possible triples from the table, linking all discrete entries to other entities or numerical values from the same rows (Fig. 7a). One asset of this method is that it produces good embeddings for all entities, as they are directly connected to their attributes in the graph. But it generates a large number of triples: $\mathcal{O}(n_{cols}^2\, n_{rows})$, which increases the training time of embeddings. If we know beforehand the entities of interest, *i.e.* those used in the end task (*e.g.* cities), we can instead build triples from these entities only (Fig. 7b). This greatly reduces the number of triples to $(n_{cols} - 1)\, n_{rows}$ (these entities generally come from a single column) and returns embeddings tailored for the entities under study. However, this approach neglects other entities: they are not directly connected to the entries of the row and are thus likely to underperform in other applications. Finally, we consider a third heuristic that assigns a row id to each row of the table, treats this row id as an entity, and then links it to the various entries of the row (Fig. 7c). This method combines benefits of the previous methods: it does not require any prior knowledge of the downstream application and generates a light graph with $n_{cols}\, n_{rows}$ triples. Yet learning an additional embedding for each row also raises scalability issues if there are much more rows than distinct entities to embed.

A desirable property of table-to-graph methods is their ability to represent joint information across columns. For instance Fig. 8a considers two cities A, B with their number of companies in different fields of activity. Taken alone, the two columns are not very informative: what matters here is the number of companies in a certain field of activity, which requires to consider both columns jointly. Methods that build triples from table entries such as cities encode the attributes "field of activity" and "number of companies" independently, and thus cannot distinguish A and B from their triples (Fig. 8b). In contrast, introducing row entities allows to capture row data jointly and differentiate the two cities (Fig. 8c).

Finally, if missing data are present in the table, we encode them with specific entities (one for each column).

4 Empirical study

We compare our approach with automatic feature extraction techniques, such as Deep Feature Synthesis (DFS) or RDF2vec, and focus on two criteria:

- the *quality* of the extracted features: how well do they improve performance in downstream tasks?
- the *scalability* of the approach: time and space complexity, size of the feature vectors

4.1 Downstream tasks

We evaluate our approach on 7 prediction tasks on various types of entities. In each task, we extract features for the entities of interest (*i.e.* target entities) from a *source* dataset, and add them to a *target* dataset containing the variable to predict. To showcase the versatility of our method, we consider tables and knowledge graphs as source data. More details about the downstream tasks and datasets are given in the appendix 8.1.

Tabular data We first consider two classification tasks: **KDD14** (classification of educational crowdfunding projects) and **KDD15** (student dropout prediction in MOOCs). For these tasks the source data consists of multiple tables describing the target entities. To leverage this data in our approach, we represent it as a knowledge graph by using target entities as head entities and linking them to other entries from the same rows, similarly to Fig. 7b.

Knowledge graphs To support our claim that general-purpose embeddings can be learned from large databases and used in various end tasks, we consider a more challenging setup: enriching several downstream tasks with background information from Wikipedia. To that end, we leverage YAGO3, a knowledge graph representation of common knowledge, built from Wikipedia and other sources [28].

Our version of YAGO3 contains 2.8 million entities, described by 7.2 million triples. We learn embeddings for various entities that are common in data science problems (counties, cities, people, companies, movies...) and use them in 5 regression tasks on socio-economic topics[4]:

- **Elections**: predict the number of votes per party in 3000 US counties.
- **Housing prices**: predict the average housing price in 23000 US cities.
- **Accidents**: predict the number of accidents in 8500 US cities.
- **Movie revenues**: predict the box-office revenues of 4900 movies.
- **Employees**: predict the number of employees in 3000 companies.

Note that there exists a more recent version of YAGO [34], with a much greater coverage of information: 64 million entities, with about 2 billion triples. However, we could not include it in our empirical study as the DFS baseline was intractable on such a large database.

[4] Target entities for which we extract features from YAGO3 are underlined.

4.2 Approaches considered for evaluation

We describe below the feature extraction approaches that we include in our empirical study.

Our approach We implement KEN on top of 3 embedding algorithms: TransE [10], the seminal work that introduced relations as translations of embeddings, DistMult [49], with scoring function $f(h, r, t) = \boldsymbol{h} \cdot (\boldsymbol{r} \odot \boldsymbol{t})$, and MuRE [6] because it emerged as a top-performing method in link prediction [3]. We learn 200-dimensional embeddings and keep all hyper-parameters constant, except for the number of epochs $\in [2, 4, 8, 16, 24, 32, 40]$ that we tune (see the appendix 8.2 for the exact parameters used). We base our implementations on PyKEEN [4], a Python library for learning knowledge graph embeddings. In addition, PyKEEN implements a version of DistMult that leverages numerical values with LiteralE [21], which allows for a comparison with KEN.

Deep Feature Synthesis We compare our embedding approach to Deep Feature Synthesis (DFS, see Fig. 2). We use an implementation of DFS from the Python package *featuretools* and extract features at depths (0, 1, 2, 3) with the default aggregation functions: `MEAN, MIN, MAX, STD, SKEW, SUM` for numerical features, `MODE, NUM_UNIQUE` for categorical features and `COUNT` for both. Categorical features are one-hot encoded to their 10 most common categories. To apply DFS on YAGO3, we convert it to tabular format by creating a table with two columns (head, tail) for each forward/inverse relation.

Manual feature engineering Besides DFS, we include manual feature engineering to our empirical study. The objective is to estimate how well an analyst would perform given a time budget of 1-2 hours per dataset. Results obviously depend on the analyst and could be improved with more effort, but they provide a simple baseline for a time-constrained analysis. See appendix 8.2 for a description of the handcrafted features we used.

RDF2vec Finally, we also compare our approach to RDF2vec, traditionally used to extract features for downstream tasks. For each entity under study, we generate all possible walks of depth 2, going through forward and backward relations (as in Fig. 3). However, as the number of walks can be very high for certain entities (*e.g.* tens of millions), we cap this number to 10000, and checked empirically that this value is large enough to impact only a small fraction of entities. We then feed these sequences to a SGNS model with embedding dimension = 200, window size = 4 (which allows to capture 1-hop and 2-hop neighborhoods), and pick the epoch $\in$ [1, 5, 10, 20] that performs best. We used the pyRDF2Vec package [45] to run the experiments.

4.3 Quality of the extracted features

Methodology We first study how well feature vectors created from a source database can improve performance in data-science tasks. For this, we consider the prediction

Table 1 Quality of the extracted features: Cross-validation scores on target datasets using either embeddings, deep feature synthesis, or manually handcrafted vectors as features. The scoring metrics are: average precision (KDD14), AUC (KDD15) and R2 for the remaining datasets. Bold and underlined scores correspond to the first and second best-performing approaches. Grayed cells indicate when MuRE + KEN outperforms deep feature synthesis. Results with standard deviations are given in the appendix (Table 11).

	Feature enrichment from domain-specific tables		*Feature enrichment from a general-purpose knowledge graph, YAGO3*				
Approach	**KDD14**	**KDD15**	**Elections**	**Housing prices**	**Accidents**	**Movie revenues**	**Employees**
Advanced analytic models: gradient boosted trees							
Feature vectors tailored for target entities							
Manual feature handcrafting	0.267	0.869	0.955	0.273	0.360	0.141	0.367
DFS, depth 0	0.158	0.584	0.836	0.165	0.162	0.016	0.126
DFS, depth 1	0.461	0.880	0.960	0.369	0.423	0.153	0.382
DFS, depth 2	0.463	0.880	0.964	0.605	0.570	0.163	0.384
DFS, depth 3	**0.499**	0.881	**0.969**	0.683	**0.590**	0.189	0.381
DFS, depth 3 + ontology			0.958	**0.686**	0.589	0.259	**0.390**
RDF2vec	0.173	0.849	0.873	0.355	0.236	0.074	0.380
General-purpose feature vectors							
TransE	0.242	0.854	0.899	0.321	0.256	0.092	0.003
TransE + KEN	0.334	0.875	0.939	0.447	0.381	0.095	0.214
DistMult	0.264	0.859	0.916	0.525	0.454	0.145	0.117
DistMult + LiteralE	0.286	0.870	0.841	0.484	0.443	0.110	0.227
DistMult + KEN	0.386	0.879	0.921	0.542	0.486	0.162	0.242
MuRE	0.287	0.863	0.945	0.571	0.461	0.165	0.109
MuRE + KEN	0.443	**0.883**	0.966	0.604	0.524	0.175	0.313
MuRE + KEN + ontology			0.957	0.602	0.541	**0.266**	0.345
Simple analytic models: K-Nearest Neighbors							
DFS, depth 0	0.078	0.504	0.742	0.004	0.130	-0.026	0.004
DFS, depth 1	0.110	0.821	0.715	0.297	0.320	0.121	0.144
DFS, depth 2	0.107	0.821	0.763	0.395	0.349	0.119	0.086
DFS, depth 3	0.142	0.816	0.618	0.503	0.361	0.043	0.025
MuRE + KEN	**0.205**	**0.830**	**0.936**	**0.536**	**0.488**	**0.136**	**0.273**

problems introduced in section 4.1 and the feature extraction approaches presented in section 4.2: TransE, DistMult and MuRE with and without KEN; Deep Feature Synthesis; manual feature engineering; and RDF2vec.

We measure performance with cross-validation scores, and only use entity representations to predict the target values [5]. For regression and classification, we use two analytic models from the *scikit-learn* library: k-nearest neighbors and gradient boosted trees, whose hyper-parameters are tuned. We report in Table 1 5-fold cross-validation scores, averaged over multiple seeds for shuffling the data and training the embedding models. See appendix 8.3 for a more detailed description of the experimental setup.

[5] Except in the Elections dataset, where we also include the political party when predicting the number of votes.

Results When using entity-embeddings as feature vectors, DistMult and MuRE overall outperform RDF2vec by a wide margin (except on the Employees dataset, where RDF2vec gets surprisingly good results), with MuRE appearing as the best approach. We explain this gap by their ability to capture well relational information. In particular, MuRE is more expressive than TransE and DistMult (their scoring functions can be seen as special cases of MuRE) and thus better model complex relations. In contrast, TransE does not model well many-to-one relationships: if we have (h, r, t) and (h', r, t), then h and h' are forced to have very close embeddings $\boldsymbol{h} = \boldsymbol{h'} = \boldsymbol{t} - \boldsymbol{r}$. Similarly, the scoring function of DistMult is symmetrical, i.e. $f(h, r, t) = f(t, r, h)$, which is not suited for non symmetrical relations like *locatedIn*. We can also see from Table 1 that leveraging numerical attributes with KEN *always* improves performance in TransE, DistMult and MuRE, and that it is superior to LiteralE in DistMult.

We now compare the performance of MuRE + KEN (the best embedding approach) to manual and automatic feature engineering methods. When using powerful prediction models (gradient boosted trees), MuRE + KEN does not consistently outperforms DFS, but is often competitive for depths ≤ 2, and almost always outperforms manual feature engineering. However, when using simpler prediction models (K-Nearest Neighbors), MuRE + KEN significantly outperforms DFS for all depths. Indeed, embeddings tend to be well structured (as induced by the scoring function) and have homogeneous coefficients with similar distributions, which facilitates the downstream learning. In contrast, DFS creates a huge number of heterogeneous features, which even after scaling are hard to leverage by simple models.

We also study whether injecting taxonomic information into embedding models improves performance. Following [15], we augment YAGO3 with triples describing its ontology, such as entity types and their relations (*subClassOf* and *disjointWith*). We apply MuRE + KEN on this augmented version of YAGO3 and observe that it generally improves prediction performance and reduces the gap with DFS.

Capturing entity types Finally, we investigate whether knowledge graph embeddings capture entity types, for instance differentiating cities from movies or counties. Such information can be useful in certain tasks that we did not consider in our previous experiments, *e.g.* clustering. To evaluate this, we take many entities of various types (cities, counties, movies, companies) from our previous tasks on YAGO3, and measure how well entity types can be predicted from their MuRE + KEN embeddings. We use a simple K-Nearest Neighbor model, whose number of neighbors is tuned and obtain a ROC AUC score of 0.996, showing that knowledge graph embeddings indeed capture entity types. We detail the experimental setup in appendix 8.3.

4.4 Scalability concerns

Large databases, such as YAGO3, bear promises to provide general-purpose feature enrichment. For this, the scalability of features extraction methods is crucial. To that end, we compare in Table 2 the scalability of various approaches: Deep Feature Synthesis (for $0 \leq \text{depth} \leq 3$), RDF2vec and MuRE (with and without KEN).

Methodology We quantify computational scalability with several metrics capturing:

1) the **scalability of feature extraction**: duration and RAM usage when computing the feature vectors.
2) the **scalability of feature usage**: dimension of the feature vectors, disk memory needed to store them, and duration of cross-validated evaluation in prediction tasks (using gradient boosted trees).

A benefit of knowledge graph embedding models is that they learn representations for all entities at once (*e.g.* cities, counties, movies in YAGO3). This is unlike DFS and RDF2vec which typically extracts feature vectors for target entities only. Given our objective to provide representations for many different entities, we thus benchmark DFS and RDF2vec when extracting features for all entities.

In some cases (KDD14 with depth 3 and YAGO3 with depth 2/3), DFS breaks the RAM capacity of our machine (400 GB) and does not terminate, even when splitting entities into 1000 chunks to lower the RAM usage. For these cases, we extrapolate the total duration based on the duration for a subset of entities, and the disk memory required to store features based on the memory it takes for a smaller number of features.

Similarly, we were not able to learn RDF2vec embeddings for all YAGO3 entities due to memory overflow. We tried limiting the number of walks to 100 per entity, and only generating them from the 1% most frequent ones, but we still could not compute them in less than a day, even with parallelization over 40 CPUs. We thus interrupted the process, and measured the duration and RAM usage just before stopping.

Results We report in Table 2 the scalability metrics described above. As expected, DFS quickly becomes intractable on large databases: it requires huge amounts of time and RAM to run, and returns very high-dimensional feature vectors that need a lot of memory to be stored and a lot of time to be leveraged by machine-learning models. Interestingly, we saw in Table 1 that DFS must be computed at a depth of 2 or more to outperform MuRE + KEN (using powerful gradient boosted tree models). Yet based on this scalability study, this is already too deep to run DFS for all entities in YAGO3, due to memory issues. In the end, DFS produces high-performance features, but its usage is limited to small databases, or when the downstream task is known beforehand so as to extract features for a subset of entities only. Unlike knowledge graph embedding models, it cannot be used to create general-purpose feature vectors from large databases with millions of entities.

We observe similar trends with RDF2vec: feature extraction for all entities overall requires much more time and memory than MuRE. Actually, even creating feature vectors for target entities only with RDF2vec can take more time (*e.g.* 9300s for 23000 cities in Housing prices) than applying MuRE to all YAGO3 entities, and must be repeated for every new downstream task.

4.5 KEN helps embeddings capture numerical attributes

As visible on Figure 9, KEN provides embeddings that represent in a much simpler way the numerical information associated with entities. When embedding counties

Table 2 Scalability of feature extraction methods: Computational scalability of embedding models versus deep feature synthesis. Grayed-out cells indicate models which are less tractable than MuRE + KEN. Red text indicates when DFS breaks the RAM capacity of our machine (400 GB).

Scalability metrics	Dataset	Deep Feature Synthesis *Depth 1*	*Depth 2*	*Depth 3*	MuRE	MuRE + KEN	RDF 2vec
Extracting feature vectors for all entities							
Duration (s)	KDD14	1014	11123	**≃110K**	2146	6708	52000
	KDD15	170	489	5107	3023	3566	1710
	YAGO3	690	**≃33K**	**≃8.5M**	1108	1762	**≥ 100K**
RAM usage (GB)	KDD14	10.5	48	**≥400**	13.2	18.6	240
	KDD15	4.9	8.2	57.7	14.8	18.8	95
	YAGO3	40.1	**≥400**	**≥400**	15.9	16.1	**≥ 30**
Using feature vectors in downstream tasks							
Dimension of feature vectors	KDD14	372	2202	19379	200	200	200
	KDD15	163	277	1870	200	200	200
	YAGO3	271	10281	141K	200	200	200
Disk memory needed to store features (GB)	YAGO3	2.8	107	1471	2.1	2.1	2.1
Duration of cross-validated evaluation (s)	KDD14	48	91	1684	103	103	103
	KDD15	4	4	9.9	19	19	19
	Elections	100	276	8989	176	176	176
	Housing prices	89	330	11589	145	145	145
	Accidents	92	317	11496	146	146	146
	Movie revenues	56	356	14988	132	132	132
	Employees	72	449	15762	88	88	88

from YAGO3, the structure of KEN embeddings reflects well the population density, with a direction grouping together metropolitan areas such as Chicago (Cook county), Los Angeles (Orange County), Houston (Harris county), and Phoenix (Maricopa county), well separated from rural counties. On the other hand, this information is more diluted in standard MuRE embeddings.

Methodology To evaluate quantitatively the ability of embeddings to capture numerical information, we compare the performance of simple supervised models to predict the numerical attributes of entities (*e.g.* county populations) from their embeddings. In practice we use K-Nearest Neighbors models (whose hyper-parameters

Table 3 Reconstructing numerical attributes - Cross-validation scores (R2) of simple nearest-neighbour models predicting the numerical attributes associated to an entity from its embedding.

Target		DistMult	DistMult + LiteralE	DistMult + KEN	MuRE	MuRE + KEN
Donation amount (KDD14)	Mean	0.20±0.05	0.58±0.14	0.62±0.12	0.22±0.06	**0.66**±0.12
	1st quartile	0.34±0.05	0.46±0.05	0.67±0.10	0.34±0.06	**0.72**±0.12
	3rd quartile	0.33±0.05	0.48±0.05	0.57±0.10	0.33±0.05	**0.59**±0.09
Connection time (KDD15)	Mean	0.09±0.01	0.33±0.01	0.92±0.01	0.10±0.02	**0.97**±0.01
	1st quartile	0.15±0.01	0.27±0.01	0.78±0.01	0.15±0.01	**0.82**±0.01
	3rd quartile	0.39±0.02	0.45±0.01	0.74±0.01	0.39±0.02	**0.84**±0.01
County attributes (YAGO3)	Population	**0.73**±0.17	0.71±0.22	**0.73**±0.15	0.32±0.08	0.51±0.16
	Latitude	0.92±0.01	0.72±0.03	**0.93**±0.01	0.72±0.03	0.91±0.01
	Longitude	0.83±0.07	0.72±0.05	**0.90**±0.07	0.64±0.06	0.81±0.06

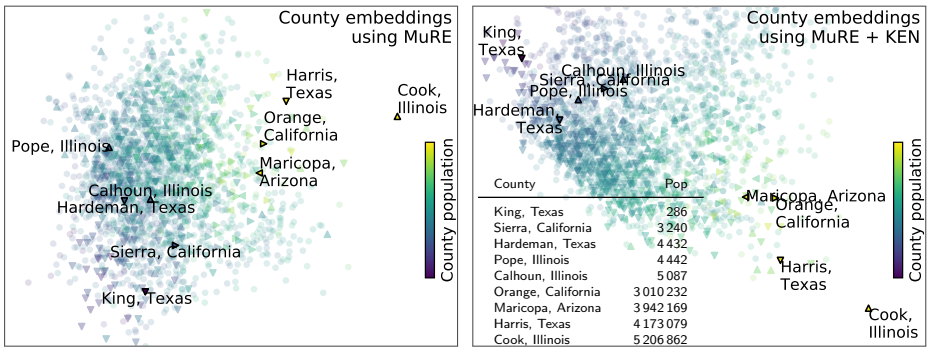


Fig. 9 Embeddings of counties using only categorical attributes (MuRE) or all attributes (KEN-E) from YAGO3: PCA projection of the 200-dimension embeddings in 2D. The color represents the county population and the symbols the state of the county. We randomly draw high and low population counties in the same state. Cook, Orange, Harris, and Maricopa counties correspond to major cities: Chicago, Los Angeles, Houston, and Phoenix. The global structure of MuRE + KEN embeddings better reflects the population of the counties, in particular separating the rural counties from those related to major cities. A simple linear projection of the MuRE + KEN embeddings suffices to roughly capture the rural-urban gradients, while it is less clear on MuRE embeddings.

Table 4 Ablation study - Drop in cross-validation scores of variants of MuRE + KEN and binning, relatively to the original MuRE + KEN. Scoring metrics are: average precision (KDD14), AUC (KDD15) and R2 for other datasets.

Dataset	**Binning**	**Variants of MuRE + KEN**	
		No quantile normalization	*No ReLU activation*
KDD14	-0.044	-0.068	-0.045
KDD15	-0.002	0	-0.001
Elections	-0.008	-0.020	-0.004
Housing prices	-0.091	-0.023	-0.021
Accidents	-0.063	-0.037	-0.010
Movie revenues	-0.015	-0.112	-0.030
Employees	-0.011	-0.007	0.002
Average across datasets	**-0.038**	**-0.047**	**-0.016**

are tuned) and aim to predict statistics about donations to projects in KDD14, students connections to MOOCs in KDD15 and county attributes in YAGO3. We measure performance with cross-validation scores. See appendix 8.4 for the exact evaluation setup.

Results The scores reported in Table 3 confirms that adding KEN significantly improves the ability to capture numerical information related to the entities: in all settings adding KEN leads to better reconstruction of numerical attributes, and also outperforms LiteralE by a wide margin. In addition, results show that these embeddings capture to some extent the whole distribution of numerical attributes: their mean, but also their quantiles.

Table 5 Embedding can capture deep features: Cross-validation scores (R2) of gradient boosted tree models using as features either embeddings trained on the full YAGO3 dataset, or on a subset of YAGO3 containing only the triples related to the target entities.

Dataset	YAGO3	TransE	TransE + KEN	MuRE	MuRE + KEN
Elections	subset	0.846	0.854	0.837	0.926
	full	0.899	0.939	0.945	0.966
Housing prices	subset	0.079	0.203	0.231	0.338
	full	0.321	0.447	0.571	0.604
Accidents	subset	0.117	0.170	0.243	0.345
	full	0.256	0.381	0.461	0.524
Movie revenues	subset	-0.003	-0.004	0.052	0.064
	full	0.092	0.095	0.165	0.175
Employees	subset	-0.015	0.071	0.087	0.297
	full	0.003	0.214	0.109	0.313

4.6 Ablation study

We study in this section the influence of two ingredients of KEN on the quality of entity-embeddings: 1) the quantile normalization of numerical values at the input, and 2) the presence of a ReLU activation function at the output (Fig. 6).

Methodology We measure the drop in performance relative to the original MuRE + KEN when: 1) replacing the quantile normalization by a min-max normalization $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$ and 2) removing the ReLU activation. We also compare KEN to a standard binning practice, where numerical values are divided into bins and an embedding is learned for each bin. In practice we use 20 bins and split values evenly across bins to be robust to fat-tailed distributions: the first bin corresponds to values in the top 5%, the second bin to values in the range 5%-10%, and so on... We use gradient boosted tree models for prediction, and the same setup as in Table 1.

Results Table 4 shows that all ingredients of KEN are important, especially the quantile normalization, and confirms that KEN leads to markedly better features than binning.

4.7 Capturing deep features with embeddings

Methodology We want to determine if embeddings can capture information deep in the knowledge graph, indirectly chaining relations as in Deep Feature Synthesis. For this purpose, we compare in Table 5 cross-validation scores of gradient boosted tree models with embeddings trained either on the full YAGO3 database, or on a subset of YAGO3 containing only the triples related to the target entities. For example, a subset with city-related triples would contain direct information about cities (*e.g.* the state in which they belong), but no information about the states themselves. Such "deep" information can however be helpful for analytical tasks, and should be captured by embeddings models. The evaluation setup is the same as in Table 1.

Results Table 5 shows that adding triples indirectly related to the target entities improves the quality of their embeddings; hence embedding models do capture deep information.

4.8 Influence of table representations

Methodology When the source data consists of tables, it must be represented as a knowledge graph to be leveraged by our approach. We introduced in section 3.3 three table-to-graph strategies, which differ on which entities are used as heads when generating triples (Fig. 7). We either use: 1) all entities, 2) only target entities (which require some prior knowledge of the downstream application) or 3) row ids. We evaluate the performance of these strategies with cross-validation scores on KDD14 and KDD15, using gradient boosted tree models for prediction (as in Table 1). To show the importance of choosing well the column with the target entities in the second approach, we also evaluate a simple baseline taking entities from another column.

Results Based on Table 6, the top performing table-to-graph strategy consists in generating triples from target entities. Indeed, the resulting graph directly connects them to their attributes, which facilitates the learning of embeddings. This intuition is confirmed when taking instead entities from another column, as we observe a sharp drop in performance. Interestingly, using all entities or row ids as head entities return embeddings that perform reasonably well without being tailored for the specific task at hand. These methods can provide general-purpose embeddings that perform well for various entities and applications. However, they either increase the number of triples (and thus the training time of embeddings) or the number of entities.

Table 6 Influence of table representations: Cross-validation scores of different strategies to represent tables as a knowledge graph. Scoring metrics are average precision (KDD14) and AUC (KDD15). We also report the number of entities and triples in the graph from each method.

Head entities in generated triples	**KDD14**		**KDD15**		**# triples** (KDD14, KDD15)	**# entities** (KDD14, KDD15)
	MuRE	MuRE + KEN	MuRE	MuRE + KEN		
Embeddings tailored for specific entities						
Target entities	<u>0.287</u>	**0.443**	<u>0.863</u>	**0.883**	44M 33M	0.94M 0.27M
Entities from another column	0.227	0.233	0.861	0.863	44M 33M	0.94M 0.27M
General-purpose embeddings						
All entities	**0.289**	0.406	**0.864**	**0.883**	155M 66M	0.94M 0.27M
Row IDs	0.282	<u>0.409</u>	0.856	0.878	51M 41M	8.4M 8.5M

5 Discussion

5.1 Embeddings capturing numerical information can provide feature enrichment

By relying on entity embeddings, our feature-synthesis pipeline departs strongly from the standard approach of feature engineering in databases. Our extensive experiments confirm that features created via knowledge graph embedding do capture the information needed for a statistical task. Embedding models coupled with KEN improve over manual feature engineering on almost all tasks.

We observe clear trends in the experimental results: Table 1 reveals the importance of capturing well *1)* the numerical attributes and *2)* relational, rather than contextual information. Indeed, across all analytic tasks and embedding methods explored, adding KEN leads to features that better capture numerical attributes and improve the downstream analytic task (Tables 3 and 1). It also improves over binning and LiteralE by a large margin. The ingredients that we introduced in KEN, such as the quantile normalization to account for the distribution of numerical attributes significantly improves performance (Table 4). Improving models of relations makes a strong difference in how useful the resulting features are for downstream tasks: there are notable improvements from RDF2vec –no explicit model of the relation– to MuRE (Table 1).

5.2 Deep Feature Synthesis cannot go so deep

Automated feature-engineering methods like Deep Feature Synthesis greatly reduce the human cost of manually handcrafting features across tables, while achieving excellent results on all datasets. With deep-enough features, DFS performs consistently better than manual feature engineering and often slightly better that MuRE + KEN (Table 1).

But this ability to generate good features comes at the price of scalability. Since DFS combines aggregation functions and features at each depth, the time and space complexity, as well as the number of created features grow exponentially (Table 2). Even on relatively small databases like KDD14 or YAGO3, building features for all entities with DFS at a depth of 2 or 3 becomes intractable, with the memory requirements greatly exceeding our machine capacity (400 GB). Besides memory limitations, the number of features quickly reaches tens or hundreds of thousands, making statistical models harder and slower to train (*e.g.* 180x longer on Employees), and reducing feature interpretability.

Yet, the databases that we have explored are smaller than the latest repositories of general knowledge: YAGO3 is 50 times smaller than YAGO4 [34]. Progress in linked open data is continuously increasing the amount of information available in a consistent representation: DBPedia [27] currently contains 900 millions triplets, and growth by a factor of 1.5 to 2 every two years [1]. For instance, we could not run DFS, even with a depth of 1, on YAGO4. Even if it could run, it would provide a huge number of features, hard to leverage.

Embeddings, on the opposite, readily provide low-dimension representations ($p = 200$) which are able to capture "deep" information, indirectly chaining relations (Table 5). Finally, knowledge graph embedding methods are very scalable: embeddings are optimized with stochastic gradient descent ($\mathcal{O}(\#\text{triplets})$), and can

be trained on huge amounts of data. Further optimizations can make embedding techniques $2-5\times$ faster than the implementations that we used [51].

Knowledge graph embedding models are also naturally suited to capture complex relational patterns between discrete elements. This is unlike DFS, which struggles to encode categorical features: ensembles of discrete entities (e.g. the cities located in a county) are aggregated by their most common element and then one-hot encoded, discarding a lot of information in the process.

5.3 Current limitations call for further work

Interpretability The biggest drawback of automatic feature generation is that it leads to models harder to interpret. Indeed, features are often manually crafted to capture a quantity of interest, such as wealth of a locality. Data scientists can then reason about the role of the corresponding quantity, for instance the impact of local wealth on housing prices. A challenge to these interpretations is that the crafted feature must represent well the quantity, but for this the burden is on the analyst and not the tool. With automatically generated features, the quantities of interest must be identified from the features. This is typically hard: even in DFS where features are associated with descriptive labels, we may have to distinguish between many partly redundant features. This is even harder in embedding models, which are black-box and do not associate human-understandable labels to individual features.

Matching and out-of-vocabulary The target data may come with different naming conventions as the source, for instance county names in the Elections dataset are written differently than in YAGO3. In such case, a form of matching must be performed (e.g. *Cook County* $\rightarrow$ *Cook, Illinois*). This is often done manually using domain-knowledge. Further work should explore automated techniques, for instance using fuzzy or similarity joins [29,42], or adapting NLP techniques used to create embeddings robust to out-of-vocabulary entities [8,35,12].

6 Conclusion

We have shown how turn-key extraction of embeddings from relational data can distill valuable information from a database, synthesizing feature vectors for data enrichment in downstream analytic tasks. For these feature vectors to be most useful in the analytic tasks, experiments show that embedding methods must model well the different relations between entities, and capture their numerical attributes. For this, we proposed to use knowledge graph embedding models designed for link prediction, and extended them to numerical attribute with KEN. Our extensive experiments show that these embeddings improve markedly upon manual feature engineering and embedding methods traditionally used for feature extraction such as RDF2vec. They are also competitive with automatic feature engineering methods based on systematic denormalizations like Deep Feature Synthesis, but do not face the same scalability challenges.

A pipeline to minimize human effort Our pipeline is designed to facilitate data preparation. Not only does it circumvent the human labor of designing manual features, but also is minimizes data integration and wrangling challenges. Operating on a triple representation –sometimes automatically built from tables– removes many tedious aspects of data input. For instance it works well on tables in "long" or "wide" formats. It also allows to capture and mix information from various data structures: tables, knowledge graphs... Yet, richer representations may be useful in the long run to better capture complex relationships within the data, such as temporal dependencies [5].

Towards general-purpose feature enrichment The scalability of our approach enabled to easily extract embeddings from YAGO3, capturing the corresponding information drawn from Wikipedia. These could readily be used as feature enrichment to improve statistical analysis on 5 different socio-economic datasets we investigated. Our work thus opens a path to capturing the large and complex stores of general information into feature vectors easy to integrate into any analysis. As such it contributes a major step towards facilitating data science with less manual data preparation.

7 Declarations

Funding The research leading to these results received funding from the french *Agence Nationale de la Recherche*, under Grant Agreement ANR-17-CE23-0018.

Conflicts of interest The authors have no relevant financial or non-financial interests to disclose.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Availability of data and material The data used to produce these results can be downloaded at `https://drive.google.com/file/d/1v4twxr0e_I9GSY9Xd7GEqnGLh3-4cGxn/view?usp=sharing`.

Code availability The code used to produce these results can be found at `https://github.com/alexis-cvetkov/KEN`

Authors' contributions Alexis Cvetkov-Iliev and Gaël Varoquaux conceived of the presented idea. Alexis Cvetkov-Iliev implemented the approach and carried out the experiments. Gaël Varoquaux and Alexandre Allauzen were involved in supervising the project and helped designing the experiments. All authors discussed the results and contributed to the final manuscript.

References

1. DBPedia web page. https://www.dbpedia.org/resources/latest-core. Accessed: 2021-11-18.
2. Kaggle industry survey, 2018.
3. Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. *arXiv preprint arXiv:2006.13365*, 2020.
4. Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. Pykeen 1.0: A python library for training and evaluating knowledge graph embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.
5. Siddhant Arora and Srikanta Bedathur. On embeddings in relational databases, 2020.
6. Ivana Balazevic, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings. *Neural Information Processing Systems*, 32:4463, 2019.
7. Florian Bauer and Martin Kaltenböck. Linked open data: The essentials. *Edition mono/monochrom, Vienna*, 710, 2011.
8. Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.
9. Rajesh Bordawekar and Oded Shmueli. Using word embedding to enable semantic queries in relational databases. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, DEEM, 2017.
10. Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems*, page 2787, 2013.
11. Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *SIGMOD*, page 1335, 2020.
12. Lihu Chen, Gaël Varoquaux, and Fabian Suchanek. Imputing out-of-vocabulary embeddings with love makes language models robust with little cost. In *ACL 2022-60th Annual Meeting of the Association for Computational Linguistics*, 2022.
13. Michael Cochez, Petar Ristoski, Simone Paolo Ponzetto, and Heiko Paulheim. Global rdf vector space embeddings. In *International Semantic Web Conference*, pages 190–207. Springer, 2017.
14. CrowdFlower. Data science report, 2016.
15. Claudia d'Amato, Nicola Flavio Quatraro, and Nicola Fanizzi. Injecting background knowledge into embedding models for predictive tasks on knowledge graphs. In *Eighteenth Extended Semantic Web Conference - Research Track*, 2021.
16. Shusaku Egami, Satoshi Nishimura, and Ken Fukuda. A framework for constructing and augmenting knowledge graphs using virtual space: Towards analysis of daily activities. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1226–1230, 2021.
17. Genet Asefa Gesese, Russa Biswas, Mehwish Alam, and Harald Sack. A survey on knowledge graph embeddings with literals: Which model links better literal-ly? *Semantic Web*, page 617, 2021.
18. Martin Grohe. Word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, 2020.
19. James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2015.
20. Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization Approaches to Relational Data Mining*, page 262286. Springer-Verlag, Berlin, Heidelberg, 2001.
21. Agustinus Kristiadi, Mohammad Asif Khan, Denis Lukovnikov, Jens Lehmann, and Asja Fischer. Incorporating literals into knowledge graph embeddings, 2019.
22. MIT Election Data Science Lab. County Presidential Election Returns 2000-2020, 2018.
23. Hoang Thanh Lam, Beat Buesser, Hong Min, Tran Ngoc Minh, Martin Wistuba, Udayan Khurana, Gregory Bramble, Theodoros Salonidis, Dakuo Wang, and Horst Samulowitz. Automated data science for relational data. In *International Conference on Data Engineering (ICDE)*, page 2689. IEEE, 2021.

24. Hoang Thanh Lam, Tran Ngoc Minh, Mathieu Sinn, Beat Buesser, and Martin Wistuba. Neural feature learning from relational database, 2019.
25. Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases, 2017.
26. Nada Lavrač, Blaž Škrlj, and Marko Robnik-Šikonja. Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, 109(7):1465–1507, 2020.
27. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6:167, 2015.
28. Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*, Asilomar, United States, January 2013.
29. Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment*, 9:636, 2016.
30. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, page 3111. 2013.
31. Sobhan Moosavi, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. A countrywide traffic accident dataset, 2019.
32. Heiko Paulheim. Exploiting linked open data as background knowledge in data mining. In *Proceedings of the 2013 International Conference on Data Mining on Linked Data*, DMoLD'13, page 110, 2013.
33. Heiko Paulheim and Johannes Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, 2012.
34. Thomas Pellissier Tanon, Gerhard Weikum, and Fabian Suchanek. Yago 4: A reason-able knowledge base. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *The Semantic Web*, pages 583–596. Springer International Publishing, 2020.
35. Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns, 2017.
36. Jan Portisch, Nicolas Heist, and Heiko Paulheim. Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction two sides of the same coin? *Semantic Web*, pages 1–24, 01 2022.
37. Petar Ristoski and Heiko Paulheim. A comparison of propositionalization strategies for creating features from linked open data. volume 1232, 09 2014.
38. Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *SEMWEB*, 2016.
39. Petar Ristoski and Heiko Paulheim. Semantic web in data mining and knowledge discovery: A comprehensive survey. *J. Web Semant.*, 36:1–22, 2016.
40. Petar Ristoski, Jessica Rosati, T. D. Noia, Renato De Leone, and Heiko Paulheim. Rdf2vec: Rdf graph embeddings and their applications. *Semantic Web*, 10:721, 2019.
41. Muhammad Rizwan Saeed and Viktor K. Prasanna. Extracting entity-specific substructures for rdf graph embedding. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 378–385, 2018.
42. Yasin N Silva, Walid G Aref, and Mohamed H Ali. The similarity join database operator. In *International Conference on Data Engineering (ICDE)*, page 892. IEEE, 2010.
43. Rita Sousa, Sara Silva, and Catia Pesquita. Evolving knowledge graph similarity for supervised learning in complex biomedical domains. *BMC Bioinformatics*, 21, 01 2020.
44. Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space, 2019.
45. Gilles Vandewiele, Bram Steenwinckel, Terencio Agozzino, and Femke Ongenae. pyrdf2vec: A python implementation and extension of rdf2vec. 2022.
46. Gilles Vandewiele, Bram Steenwinckel, Pieter Bonte, Michael Weyns, Heiko Paulheim, Petar Ristoski, Filip De Turck, and Femke Ongenae. Walk extraction strategies for node embeddings with rdf2vec in knowledge graphs, 2020.
47. Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29:2724, 2017.

48. Yanrong Wu and Zhichun Wang. Knowledge graph embedding with numeric attributes of entities. In *Workshop on Representation Learning for NLP*, page 132, 2018.
49. Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *ICLR*, 2015.
50. Li Zhang, Shuo Zhang, and Krisztian Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 1029, 2019.
51. Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George 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*, pages 739–748, 2020.
52. Zillow. Home value index, 2021.

8 Appendix

8.1 Downstream tasks

Tabular data

- The **KDD14** competition aims to predict "exciting" educational projects on a crowdfunding platform (binary target). The source data consists of three tables describing the projects, the donations they received, and the resources they need. The exact columns used in our experiments are described in Table 7. Since embedding models with KEN are designed for discrete entities or numerical values, we perform minimal preprocessing on a few columns with different data types. For instance, we encode *donations_message* (free text) by their length. Temporal data, such as *donation_timestamp* are converted to a number of days after the project posting date. We also convert *date_posted* to a number of days after an arbitrary reference date. For a fair comparison, we use the same preprocessed features when running DFS.
- The **KDD15** challenge aims to predict student dropout prediction in MOOCs (binary target), using as source data 4 tables that contain information about the courses and how often students interacted with them (see Table 8). To account for the temporal information in KDD15, we replace logs times (date) by numbers in $[0, 1]$, describing when they occur relatively to the courses start/end dates. We also replace the courses starting dates by a number of days after a reference date, and drop the ending dates as all courses have the same duration (29 days), making this feature uninformative.

Datasets augmented with YAGO3 embeddings

- **Elections -** We consider voting statistics in the 2020 presidential election, and aim to predict the number of votes per party in 3000 US counties. As the

Table 7 Description of the KDD14 dataset. The *outcomes* table contains the target entities *project_id* for which we want to create feature vectors, and the binary value to predict *is_exciting*. We always use *project_id* as the head column when building the graph.

outcomes
project_id (str)
is_exciting (target)

donations
project_id (str)
donor_city (str)
donor_state (str)
is_teacher_acct (bool)
donation_timestamp (date)
donation_to_project (float)
donation_optional_support (float)
donation_message (text)

resources
project_id (str)
project_resource_type (str)
item_unit_price (float)
item_quantity (int)

projects
project_id (str)
teacher_id (str)
school_id (str)
school_city (str)
school_state (str)
primary_focus_subject (str)
primary_focus_area (str)
secondary_focus_subject (str)
secondary_focus_area (str)
resource_type (str)
poverty_level (str)
grade_level (str)
eligible_double_your_impact_match (bool)
eligible_almost_home_match (bool)
total_price_excluding_optional_support (float)
total_price_including_optional_support (float)
students_reached (float)
date_posted (date)

Table 8 Description of the KDD15 dataset. The *outcomes* table contains the target entities *enrollment_id* for which we want to create feature vectors, and the binary value to predict *dropout*. We use the first column of each table as head column when building the graph.

outcomes
enrollment_id (str)
dropout (target)

enrollments
enrollment_id (str)
student_id (str)
course_id (str)

dates
course_id (str)
from (date) — Course starting date
to (date) — Course ending date

objects (course modules)
module_id (str) — A module of a course
course_id (str)
category (str)

logs (student interactions with courses)
enrollment_id (str)
event (str) — Type of interaction
source (str) — Event source
object (str) — Module being interacted with
time (date) — Time of the event

original data [22] come with no general information about counties, we enrich them with county embeddings learned on YAGO3

- **Housing prices -** We want to predict the typical housing price in 23000 US cities using their YAGO3 embeddings. We take target estimates from the Zillow group [52].
- **Accidents -** We aim to predict the number of accidents in 8500 US cities between 2016 and 2020 using their YAGO3 embeddings. We use data described in [31].
- **Movie revenues -** We aim to predict the box-office revenues of 4900 movies using their YAGO3 embeddings. We used data from: https://www.kaggle.com/rounakbanik/the-movies-dataset.
- **Employees -** We aim to predict the number of employees in 3000 companies using their YAGO3 embeddings. We used data from: https://www.kaggle.com/peopledatalabssf/free-7-million-company-dataset

Since all these targets span over several orders of magnitude. We predict log(target) instead of the target in our experiments.

Statistics on source datasets We give in Table 9 the number of entities, relations and triples in the knowledge graph representations of the source data used to learn entity-embeddings.

Table 9 Statistics of the knowledge graphs representations for the data used to train embeddings in our experiments. Numbers in parenthesis describe the part of numerical relations and triplets in the total.

Source data	Entities	Relations	Triples
KDD14	945k	27 (10)	44M (22.3M)
KDD15	227k	9 (2)	33M (8.2M)
YAGO3	2.8M	58 (21)	7.2M (1.6M)

8.2 Approaches considered for evaluation

Our approach When training embedding models (MuRE, DistMult and TransE), we do not tune hyper-parameters and use the following values in all experiments:

- embedding dimension = 200
- distance in scoring function: ℓ_2 norm for MuRE, ℓ_1 norm for TransE and DistMult
- batch size = 10^5
- optimizer: Adam with learning rate = 10^{-3}.
- loss function: margin loss with $\gamma = 4$ in TransE, and a softplus loss for MuRE and DistMult
- negative sampling: for each positive triple (h, r, t), we generate 10 negative samples by replacing the head h by a random entity h' that co-occurs with the relation r. Doing so provides harder negative triples and improves the results.

We then train each model for 40 epochs, and pick the epoch $\in [2, 4, 8, 16, 24, 32, 40]$ that leads to the best cross-validation scores in downstream tasks.

A technical subtlety with MuRE is that we must define biases $b_t(x)$ for numerical values x. We do so by learning a constant bias b_r for each numerical attribute r: $\forall x, b_t(x) = b_r$.

Manual feature engineering We describe below the typical feature engineering steps that we performed. See Table 10 for the exact list of handcrafted features.

- identifying relevant features
- building features using joins and simple aggregation functions (mean, counts)
- one-hot encoding of low-cardinality categorical features
- removing irrelevant, redundant, or hard to encode features (e.g. with high cardinality)

Table 10 Manually handcrafted features for each dataset.

Dataset	Handcrafted features	
	Numerical	Categorical (one-hot encoded)
KDD14	– donation_to_project (mean, counts) – length_donation_message (mean) – students_reached – total_price_excluding_optional_support – total_price_including_optional_support – eligible_double_your_impact_match – eligible_almost_home_match	– primary_focus_subject – primary_focus_area – resource_type – poverty_level – grade_level
KDD15	– # of interactions (events) with courses – mean event time (relative to the course starting/ending dates) – course starting date – # of modules per course	– course_id
Elections	county population, latitude, longitude, area, population density	
Housing prices	city population, latitude, longitude, area, population density	
Accidents	city population, latitude, longitude, area, population density	
Movie revenues	– duration of the movie – number of actors, creators, editors, directors, music writers	– country of production
Employees	– mean value of all numerical attributes that exist for at least 5% of the companies – counts of all non-numerical attributes that exist for at least 5% of the companies	

8.3 Quality of the extracted features

When using gradient boosted tree models (which offer native support for missing values), we use the default parameters from sklearn, except on the smaller datasets using YAGO3 embeddings. For these datasets, we tune the following model parameters with a cross-validated grid search: *max_depth* $\in$ [2, 4, 6, None] and *min_samples_leaf* $\in$ [4, 6, 10, 20].
When using KNNs, we tune the number of neighbors $\in$ [1, 3, 5, 10, 30], except on KDD14/15. We also impute missing values (common in DFS) with the median of each feature, and then normalize feature values between 0 and 1 with min-max scaling.

We report in Table 1 5-fold cross-validation scores, averaged across 5 random shuffles of the data (3 for KDD14/15) and over 3 different seeds for training the RDF2vec and knowledge graph embeddings (1 for KDD14/15). We also provide in Table the standard deviations across train-test splits associated to these scores.

To evaluate the ability of knowledge graph embedding models to capture entity types, we sample 1000 entities from the following datasets: Elections (counties), Housing prices (cities), Movie revenues (movies), Employees (companies), for a total of 4000 entities. When then measure with cross-validation how well MuRE + KEN embeddings predict entity types, using a simple KNN model whose number of neighbors $\in$ [1, 3, 5, 10, 30] is tuned. The cross-validation parameters are the same as above.

Table 11 Quality of the extracted features: Cross-validation scores and standard deviations on target datasets using either embeddings, deep feature synthesis, or manually handcrafted vectors as features.

	Feature enrichment from domain-specific tables		*Feature enrichment from a general-purpose knowledge graph, YAGO3*				
Approach	**KDD14**	**KDD15**	**Elections**	**Housing prices**	**Accidents**	**Movie revenues**	**Employees**
Advanced analytic models: gradient boosted trees							
Feature vectors tailored for target entities							
Manual feature handcrafting	0.267 ± 0.004	0.869 ± 0.002	0.955 ± 0.003	0.273 ± 0.011	0.360 ± 0.021	0.141 ± 0.017	0.367 ± 0.035
DFS, depth 0	0.158 ± 0.002	0.584 ± 0.005	0.836 ± 0.007	0.165 ± 0.010	0.162 ± 0.016	0.016 ± 0.010	0.126 ± 0.036
DFS, depth 1	0.461 ± 0.006	0.880 ± 0.003	0.960 ± 0.003	0.369 ± 0.014	0.423 ± 0.020	0.153 ± 0.019	0.382 ± 0.035
DFS, depth 2	0.463 ± 0.006	0.880 ± 0.003	0.964 ± 0.003	0.605 ± 0.029	0.570 ± 0.016	0.163 ± 0.019	0.384 ± 0.035
DFS, depth 3	**0.499** ± 0.007	0.881 ± 0.003	**0.969** ± 0.002	0.683 ± 0.019	**0.590** ± 0.014	0.189 ± 0.023	0.381 ± 0.033
DFS, depth 3 + ontology			0.958 ± 0.005	**0.686** ± 0.019	0.589 ± 0.015	0.259 ± 0.023	**0.390** ± 0.031
RDF2vec	0.173 ± 0.003	0.849 ± 0.003	0.873 ± 0.009	0.355 ± 0.029	0.236 ± 0.019	0.074 ± 0.014	0.380 ± 0.047
General-purpose feature vectors							
TransE	0.242 ± 0.004	0.854 ± 0.003	0.899 ± 0.005	0.321 ± 0.046	0.256 ± 0.019	0.092 ± 0.016	0.003 ± 0.016
TransE + KEN	0.334 ± 0.004	0.875 ± 0.003	0.939 ± 0.006	0.447 ± 0.030	0.381 ± 0.020	0.095 ± 0.018	0.214 ± 0.031
DistMult	0.264 ± 0.004	0.859 ± 0.003	0.916 ± 0.043	0.525 ± 0.022	0.454 ± 0.023	0.145 ± 0.019	0.117 ± 0.032
DistMult + LiteralE	0.286 ± 0.005	0.870 ± 0.003	0.841 ± 0.038	0.484 ± 0.013	0.443 ± 0.022	0.110 ± 0.021	0.227 ± 0.029
DistMult + KEN	0.386 ± 0.007	0.879 ± 0.003	0.921 ± 0.053	0.542 ± 0.020	0.486 ± 0.022	0.162 ± 0.021	0.242 ± 0.034
MuRE	0.287 ± 0.005	0.863 ± 0.003	0.945 ± 0.015	0.571 ± 0.030	0.461 ± 0.021	0.165 ± 0.022	0.109 ± 0.038
MuRE + KEN	0.443 ± 0.006	**0.883** ± 0.003	0.966 ± 0.005	0.604 ± 0.020	0.524 ± 0.020	0.175 ± 0.025	0.313 ± 0.039
MuRE + KEN + ontology			0.957 ± 0.014	0.602 ± 0.028	0.541 ± 0.022	**0.266** ± 0.030	0.345 ± 0.040
Simple analytic models: K-Nearest Neighbors							
DFS, depth 0	0.078 ± 0.001	0.504 ± 0.030	0.742 ± 0.024	0.004 ± 0.027	0.130 ± 0.021	-0.026 ± 0.061	0.004 ± 0.140
DFS, depth 1	0.110 ± 0.002	0.821 ± 0.004	0.715 ± 0.019	0.297 ± 0.015	0.320 ± 0.023	0.121 ± 0.021	0.144 ± 0.020
DFS, depth 2	0.107 ± 0.001	0.821 ± 0.004	0.763 ± 0.015	0.395 ± 0.010	0.349 ± 0.020	0.119 ± 0.022	0.086 ± 0.028
DFS, depth 3	0.142 ± 0.002	0.816 ± 0.003	0.618 ± 0.050	0.503 ± 0.011	0.361 ± 0.021	0.043 ± 0.037	0.025 ± 0.022
MuRE + KEN	**0.205** ± 0.003	**0.830** ± 0.003	**0.936** ± 0.017	**0.536** ± 0.013	**0.488** ± 0.021	**0.136** ± 0.024	**0.273** ± 0.034

8.4 KEN helps embeddings capture numerical attributes

We obtain the results from Table 3 by predicting certain numerical attributes of entities from their embeddings, using simple K-Nearest Neighbors models. For the embeddings, we kept those from Table 1. We also tuned the hyper-parameters of nearest neighbors models to maximize prediction performance, using a cross-validated grid search over the following parameters:

- number of neighbors $\in$ [1, 2, 3, 4, 8, 16]
- distance: ℓ_1 or ℓ_2 norm
- weighting of the neighbors: uniform or proportional to the distance with the target entity

The final scores are then obtained with 5-fold cross-validation, averaged over 5 repeats.