# Taxonomy Extraction Using Knowledge Graph Embeddings and Hierarchical Clustering

No Author Given

No Institute Given

**Abstract.** While high-quality taxonomies are essential to the Semantic Web, building them for large knowledge graphs is an expensive process. Likewise, creating taxonomies that accurately reflect the content of dynamic knowledge graphs is another challenge. In this paper, we propose a method to automatically extract a taxonomy from knowledge graph embeddings and a set of typed entities, and evaluate which embedding models are better suited for this task. Our approach produces a taxonomy by leveraging the type information contained in the graph and the tree-like structure of a hierarchical clustering performed over entity embeddings. We evaluate our method on DBpedia and conduct an analysis of the results.

**Keywords:** Knowledge Graph Embedding · Taxonomy Extraction · Hierarchical Clustering.

## 1 Introduction

Ontologies and taxonomies are essential to the Semantic Web while being expensive to build manually. Automating the process of building taxonomies is therefore an active research field, and many approaches have been proposed [25,2]. Most of these approaches work directly at the level of classes [12,8,16], for example by identifying subsumption patterns in some data source, *e.g.* text corpora. Therefore they can only handle classes that already exist in the data. Alternatively, working at the level of instances (that is, representing a class as the set of its instances) offers the possibility to identify groups of entities that have no corresponding class in the ontology but are nonetheless relevant. Since knowledge graphs are dynamic, relevant and irrelevant classes can change over time, so instance-based approaches can be a way to dynamically adjust ontologies to changes in their corresponding graph.

In this paper, we propose a method to automatically extract a taxonomy from a knowledge graph, using hierarchical clustering over embedding vectors. Embedding vectors are dense vector representations of a graph's entities: they are based on the triples each entity is involved in, so that similar entities in the graph have similar embeddings in the vector space. By applying hierarchical clustering to these embeddings, we obtain groups (or *clusters*) of geometrically close embeddings, each group being part of a broader group, and containing several subgroups. Since geometric proximity between vectors mean semantic proximity between entities, the result is a tree-like structure of semantically coherent collections of entities. Finally, we match these collections of entities with known classes, which turns the tree of clusters into a taxonomy.

**Definitions and notations.** Throughout this paper, we use bold lowercase letters for vectors and bold capital letters for matrices. Let $\mathcal{E}$ and $\mathcal{R}$ be a set of entities and relations respectively, and $\mathcal{KG} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ a knowledge graph. An entity $e$ from $\mathcal{KG}$ has type $t$ if and only if $(e, r_{\text{IS\_A}}, t) \in \mathcal{KG}$, where $r_{\text{IS\_A}}$ represents the relation `rdf:type`. We denote $\mathcal{T} \subseteq \mathcal{E}$ the set of types[1] in the knowledge graph: for example, $\mathcal{T} = \{\texttt{Person}, \texttt{Place}, \texttt{Thing}, \ldots\}$. An axiom $t_i \sqsubset t_j$ is a *subsumption axiom*, with $t_i$ being the *subtype* and $t_j$ the *supertype*.

A model mapping entities and relations to a vector space ($\mathbb{R}^d$ or $\mathbb{C}^d$) is called a *knowledge graph embedding model*. For a given embedding model, we denote $\mathbf{u}$ the vector representation (also called the *embedding*) of an entity or a relation $u \in \mathcal{E} \cup \mathcal{R}$. Knowledge graph embedding models are described in greater detail in section 3.1.

**Problem Statement.** We assume we have at our disposal a dataset of labeled entities $\mathcal{D} \subseteq \mathbb{R}^d \times \mathcal{T}$. $\mathcal{D}$ is a set of entity-type pairs $(\mathbf{e_i}, t_i)$, where $e_i \in \mathcal{E}$ is an entity with type $t_i \in \mathcal{T}$, and $\mathbf{e}_i$ is its embedding. The goal is to extract a taxonomy for types in $\mathcal{T}$, that is to say a set of subsumption axioms $\hat{T} = \{t_i \sqsubset t_j\}$. For $\hat{T}$ to be valid, we need to enforce two rules: each type must have at most one supertype, and $\hat{T}$ must not contain cycles (*i.e.* no $t_0 \sqsubset t_1 \sqsubset \ldots \sqsubset t_m \sqsubset t_0$). In this paper, we present a novel way to achieve this goal by using hierarchical clustering and type-cluster mapping, as well as a comparison of different embedding models on this taxonomy extraction task.

The reminder of the paper is organized as follow: in Section 2 we describe related work regarding taxonomy extraction and clustering over knowledge graphs. In Section 3 we explain our approach and break it down to four main steps. In Section 4, we evaluate our approach over a subset of DBpedia and provide a comparison between different embedding models and extraction parameters.

## 2 Related Work

Extracting logical rules from data is an important research field. In its most general formulation, ontology generation aims at infering logical rules from knowledge bases, or from other sources such as text corpora. This includes refining or completing an existing ontology [26,14], adapting a known ontology to a new knowledge graph [6] or building an ontology from scratch [18]. In this paper, we consider only a specific subproblem, namely taxonomy generation: a taxonomy is an ontology containing only subsumption axioms.

[1] Types are often named *classes* in the litterature, but we refrained from doing so to avoid notation clash with clusters.

Early approaches for this task used symbolic artificial intelligence, specifically Inductive Logic Programming [13] or Formal Concept Analysis with human supervision [21]. Such approaches are often sensitive to noisy data and won't scale well [23]. As such, they are not suitable for large, real-word knowledge graphs. A more recent and most successful approach is to rely on natural language text (book corpora, Web crawl, and so on). The goal is generally to extract pairs of hyponym-hypernym concepts from text. [10,12] and others use textual patterns to identify such pairs of words. Many modern methods use word embeddings: [8] recursively clusters embeddings of concepts to build a taxonomy, [7] or [28] learns a geometric relation between the embeddings of hyponyms and hypernyms. [17] and [16] suggest that embedding models with non-euclidean geometries are better suited to model hierarchical relations betweens objects and propose new architectures to train them.

The approaches described above only work at the level of concepts, and do not handle sets of instances. More closely related to our work is the TIEmb method, described in [19]. It shares a similar setting: starting from a knowledge graph without taxonomic information, the goal is to re-create a taxonomy using labeled vector space embeddings. In this vector space, each type $t$ is represented by a sphere whose center is the average embedding of the entities from $t$, and whose radius is the average distance between these embeddings and the center. A type $A$ is predicted to be a subtype of type $B$ if the sphere representing $A$ is included in the sphere representing $B$. An extra pruning step is necessary to avoid cycles in the induced taxonomy. TIEmb uses type information from the start: all the extracted taxonomic structure relies on supervised data. In our approach, the structure is first extracted in an unsupervised fashion, using only embeddings and without type information. In Section 4, we compare TIEmb to our own approach.

## 3 Proposed Approach

Prior to the taxonomy extraction itself, the knowledge graph is embedded into a $d$-dimensional vector space. The input dataset can thus be seen as a $N \times d$ point cloud. This point cloud is first clustered using agglomerative clustering. Then, types are mapped to clusters and injected in the clustering tree. From this typed tree, the taxonomic tree is extracted. The main steps of our approach are represented in figure 1 and described in details in the following sections.

### 3.1 Embedding Models

For the method described here, we work on vector representations of entities, and not on the graph itself. To obtain such representations, many embedding models have been proposed [24]. In this section, we present a general framework for these models, and describe the four models used in this paper.

In a given embedding model, each entity $e$ is represented by a vector $\mathbf{e} \in \mathbb{R}^d$, and each relation $r$ is represented by a vector $\mathbf{r} \in \mathbb{R}^{d'}$; the model defines an

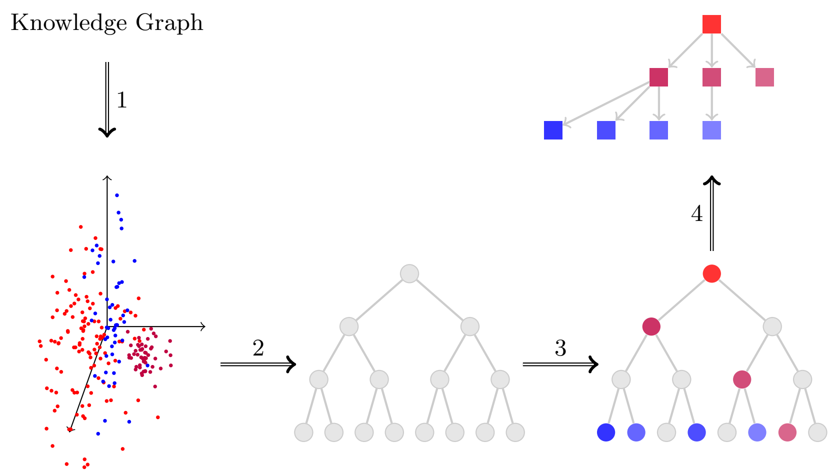

**Fig. 1.** Overview of the proposed approach: starting from a knowledge graph $\mathcal{KG}$ and a set of entity-type pairs, (1) entities are embedded into a $d$-dimensional vector space (2) then they're hierarchically clustered; (3) each type in the original dataset is then mapped to one of the cluster, (4) the taxonomy is extracted by removing non-selected clusters

evaluation function to assess if a triple $(h, r, t)$ is valid or not, based solely on the embeddings $\mathbf{h}, \mathbf{r}, \mathbf{t}$. This evaluation function can be a score function $\sigma : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \to \mathbb{R}$, which can be interpreted as a probability:

$$P((h, r, t) \text{ is valid}) = S(\sigma(\mathbf{h}, \mathbf{r}, \mathbf{t})) \tag{1}$$

With $S$ the logistic function. Alternatively, the evaluation function can be an energy function $E : \mathbb{R}^d \times \mathbb{R}^{d'} \times \mathbb{R}^d \to [0, +\infty)$: in that case, a low-energy triple is likely to be valid, whereas a high-energy triple is likely to be invalid.

For training a knowledge graph embedding model, a set of valid triples $\Delta^+$ (usually $\Delta^+ = \mathcal{KG}$) and a set of invalid triples $\Delta^-$ are collected. After a random initialization of the embeddings, the model is trained in order to maximize $\sigma$ over $\Delta^+$ and minimize it over $\Delta^-$. For energy-based models, the energy function $E(h, r, t)$ is minimized over $\Delta^+$ and maximized over $\Delta^-$.

In this paper, we explore four different embedding models: TransE [3], DistMult [27], ComplEx [22] and RDF2Vec [20] . These four models are widely used, yield good results on standard evaluation tasks (link prediction and triple classification) and yet are light enough to be trained on large-scale graphs such as DBpedia. For all models but RDF2Vec, we used the implementation provided by OpenKE [9] and the default hyperparameters. For RDF2Vec, we used the embeddings released by [5].

**TransE.** The geometric assumption behind TransE is that a relation can be seen as a translation operation between its subject and its object. Each object (entity or relation) is represented by a $d$-dimensional vector, and the energy of a triple is:

$$E(h, r, t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_2$$

**DistMult.** The DistMult model was introduced by [27] and uses the following bilinear score function:

$$\sigma(h, r, t) = \mathbf{h}^\top \mathbf{D_r} \mathbf{t}$$

With $\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$ and $\mathbf{D_r} \in \mathbb{R}^{d \times d}$ a diagonal matrix. It has the same number of parameters than TransE, but the interaction between the embeddings of $h, r$ and $t$ is multiplicative instead of additive.

**ComplEx.** The ComplEx model extends the idea of DistMult, but embeds instances in the complex space: $\mathbf{h}, \mathbf{t} \in \mathbb{C}^d, \mathbf{D_r} \in \mathbb{C}^{d \times d}$ a diagonal matrix, and:

$$\sigma(h, r, t) = \Re(\mathbf{h}^\top \mathbf{W_r} \bar{\mathbf{t}})$$

Depending on whether $\mathbf{W_r}$ is real, purely imaginary or neither one, the score function is symmetric ($\sigma(h, r, t) = \sigma(t, r, h)$), antisymmetric ($\sigma(h, r, t) = -\sigma(t, r, h)$) or neither symmetric nor antisymmetric. Since relations in a knowledge graph can themselves be symmetric (*e.g.* `owl:sameAs`), antisymmetric (*e.g.* `rdf:type`) or neither, ComplEx offers a more flexible representation of relations. By contrast, DistMult's score function is always symmetric and TransE's energy function is never symmetric, except in one special case ($\mathbf{r} = \mathbf{0}$).

**RDF2Vec.** RDF2Vec applies the idea of Word2Vec [15] to knowledge graph embeddings. Instead of sentences, RDF2Vec uses random walks sampled over the knowledge graph, *i.e.* random sequences of connected entities and relations. These walks are then fed to a shallow neural network whose goal is to predict an entity based on its neighbors.

### 3.2 Clustering

After the embedding step, the labeled dataset $\mathcal{D} = \{(\mathbf{e_i}, t_i)\}$ contains $N$ points in the $d$-dimensional Euclidean space. An agglomerative clustering is performed over these points: at first, there are $N$ leaf clusters, each containing a single entity. Clusters are then merged recursively, until there is only one root cluster containing all entities. At each step, the two merged clusters are chosen so as to minimize some merging criterion. Criterion can be the variance of the resulting cluster (Ward linkage) or the average distance between entities from each merged cluster (average linkage), for some distance function (cosine or euclidean). The output of this algorithm is a binary clustering tree whose root is the whole dataset and whose leaves are all entities. This step is not supervised, and does not use the labels (types) from $\mathcal{D}$.

We denote $\mathcal{C}$ the set of all clusters, $X$ the clustering tree, and `root` the root cluster. For two clusters $C$ and $C'$, we note $C \preceq C'$ if $C$ is a predecessor of $C'$, that is $C$ belongs to the unique path between the root cluster and $C'$ (conversely, $C'$ is a successor of $C$, denoted $C' \succeq C$). We say that $C$ is a *strict* predecessor of $C'$ (denoted $C \prec C'$) if $C \preceq C'$ and $C \neq C'$. $\preceq$ defines a partial order over the tree. For a given cluster $C$, the subtree with root $C$, denoted $X[C]$, is the tree containing $C$ and all its successors. left($C$) and right($C$) are respectively the left and right sub-clusters of $C$.

### 3.3 Mapping Types to Clusters

At this point, we have on one hand a clustering tree containing hierarchical information about groups of entities but no type information, and on the other hand a flat set of types with no hierarchy between them: this step merges these two sources of information by injecting types into the clustering tree. We propose two ways of achieving this. The first one finds a one-to-one mapping from types to clusters : each type $t$ is mapped to the cluster $C$ that best represents it. The second approach is an extension of the first one: instead of associating a type to one single cluster, it computes a mapping probability for each type-cluster pair (that is, a probability that cluster $C$ represents type $t$). In both cases, the resulting mapping is used to transform the clustering tree into a taxonomic tree.

**First Approach: Hard Mapping.** To measure if a cluster $C$ represents well a type $t$, we can measure its precision and its recall. Let $N_{C,t} = |e \in C, type(e) = t|$ be the number of entities of type $t$ in cluster $C$, $N_t = |e \in \mathcal{D}, type(e) = t|$ the total number of entities with type $t$, and $N_C = |C|$ the total number of entities in cluster $C$. Then we'll have:

$$prec(C,t) = \frac{N_{C,t}}{N_C}$$

$$rec(C,t) = \frac{N_{C,t}}{N_t}$$

$$F(C,t) = 2 \cdot \frac{prec(C,t) \times rec(C,t)}{prec(C,t) + rec(C,t)} = 2 \cdot \frac{N_{C,t}}{N_C + N_t}$$

A high $F$-score $F(C,t)$ means that cluster $C$ contains mostly entities of type $t$, and most of the entities of type $t$ are in $C$.

Ideally, we would map type $t$ to the cluster that maximizes its F-score, *i.e* choose $m^*(t) = \arg\max_C F(C,t)$, but then we could have two types mapped to the same cluster. If two types $t_A$ and $t_B$ are mapped to the same cluster $C$, then any type that is mapped to a successor of $C$ will have both $t_A$ and $t_B$ as ancestors, and the resulting taxonomy will not be valid. The mapping thus need to be unambiguous. Formally, this condition means that we need to find a injective mapping between types and clusters $m^* : \mathcal{T} \to \mathcal{C}$, so that distinct types are mapped to distinct clusters. For any injection $m : \mathcal{T} \to \mathcal{C}$, we define a score function $J(m)$ which is simply the global $F$-score induced by $m$:

$$J(m) = \sum_{t \in \mathcal{T}} F(m(t), t)$$

Finally, our optimal mapping is :

$$m^* = \underset{\substack{m:\mathcal{T}\to\mathcal{C} \\ m \text{ injective}}}{\arg\max} \sum_{t\in\mathcal{T}} F(m(t),t) \tag{2}$$

This is a linear sum assignment problem, for which an optimal solution exists (and is unique if all F-scores are distinct). To find this optimal solution, we used the Jonker-Volgenant algorithm, with a worst-case complexity of $O(|\mathcal{C}|^3)$ [11]. If time were an issue, complexity could be brought down using heuristics, such as Asymmetric Greedy Search [4].

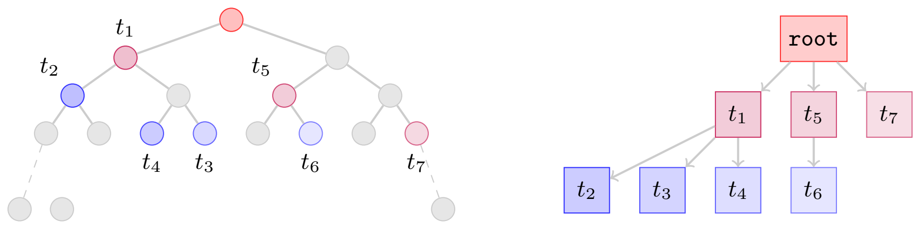

**Fig. 2.** Extracting a taxonomy from a clustering tree using Hard Mapping: each type is mapped to one cluster in the clustering tree, which results in a partial labeling of the tree (left); then we extract the subtree containing only labeled clusters (right).

*Taxonomy Extraction.* Once the mapping is done, extracting taxonomic axioms is straightforward because the clustering tree already induces a partial order over the clusters. Labeled clusters are clusters which are associated to one of the types, *i.e* all clusters $C$ such that $\exists t \in \mathcal{T}, m^*(t) = C$. The predicted taxonomy $\hat{T}$ is a directed graph whose vertices are the root cluster and all labeled clusters. $\hat{T}$ contains an edge $C_i \to C_j$ if the path from the root to $C_j$ contains $C_i$ and no other cluster is labeled in the path between $C_j$ to $C_i$ (that is, $C_i$ is the closest labeled ancestor of $C_j$ in the clustering tree). Computing $\hat{T}$ is inexpensive, as it only requires a depth-first search over the clustering tree. Thanks to the injectivity of $m^*$, the resulting graph is guaranteed to be a tree.

**Second Approach: Soft Mapping.** The previous approach results in a hard decision function: an axiom $(t_i \sqsubset t_j)$ is predicted or not, with no middle ground. This is because the arg max function from equation 2 is discontinuous. Thus, the predicted taxonomy is sensitive to small changes in the input data.

Here, we explore a soft version of the previous approach, which retains the idea of a mapping between types and clusters, but smoothes it by replacing arg max by a softmax function. A type is no longer mapped to a single cluster, but rather has a probability vector indicating how well each cluster represents it. Then, once again, this soft mapping is used to transform the clusters' hierarchy into a type hierarchy.

The first step is to turn $F$-scores into probabilities, using a softmax: for a type $t$ and and cluster $C$, we have the following probability that type $t$ is represented by cluster $C$:

$$P(m(t) = C) = \frac{e^{\beta F(t,C)}}{\sum_{C' \in \mathcal{C}} e^{\beta F(t,C')}} \tag{3}$$

The $\beta$ parameter controls the concentration of the distribution around the maximum value. $\beta \to 0$ gives all axioms the same probability, whereas $\beta \to \infty$ gives all the probability mass to the maximum value. The choice of $\beta$ value is discussed in 4.3.

This soft mapping between types and clusters can in turn be used to compute the probability of a subsumption axiom $(t' \sqsubset t)$, which is simply the probability that the cluster $m(t)$ representing $t$ is a predecessor of the cluster $m(t')$ representing $t'$. Under the hypothesis of independence between $m(t)$ and $m(t')$, we have:

$$\begin{aligned} P(t' \sqsubset t) = P(m(t) \prec m(t')) &= \sum_{\substack{C_A, C_B \in \mathcal{C} \\ C_A \prec C_B}} P(m(t) = C_A, m(t') = C_B) \\ &= \sum_{\substack{C_A, C_B \in \mathcal{C} \\ C_A \prec C_B}} P(m(t) = C_A) \cdot P(m(t') = C_B) \end{aligned} \tag{4}$$

To compute these probabilities for all $t, t'$ without summing over all pairs of clusters, we devise a divide-and-conquer algorithm. For this, we need a recursive formulation of $P(t' \sqsubset t)$, so we introduce $P_C(t' \sqsubset t)$, the probability of axiom $(t' \sqsubset t)$ computed on the subtree $X[C]$, for any given cluster $C$. Similarly to equation 4, it represents the probability that the cluster $m(t)$ representing $t$ is a predecessor of the cluster $m(t')$ representing $t'$, with the additional condition that both $m(t)$ and $m(t')$ must belong to the subtree $X[C]$ (that is, be successors of $C$):

$$\begin{aligned} P_C(t' \sqsubset t) &= P(C \preceq m(t) \prec m(t')) \\ &= \sum_{\substack{C_A, C_B \in \mathcal{C} \\ C \preceq C_A \prec C_B}} P(m(t) = C_A) \cdot P(m(t') = C_B) \end{aligned} \tag{5}$$

A pair $C_A, C_B$ such that $C \preceq C_A \prec C_B$ is in exactly one the following situation: $C_A$ and $C_B$ are either both in $C$'s left subcluster, both in $C$'s right subcluster, or $C_A = C$ and $C_B$ is in one of $C$ subclusters. Let $L = \text{left}(C)$ and $R = \text{right}(C)$, we use the previous observation to split the sum in three, which gives:

$$\begin{aligned}
P_C(t' \sqsubset t) = &\sum_{C_B\,:\,C \prec C_B} P(m(t)=C) \cdot P(m(t')=C_B) \\
&+ \sum_{\substack{C_A,C_B \in \mathcal{C} \\ L \preceq C_A \prec C_B}} P(m(t)=C_A) \cdot P(m(t')=C_B) \\
&+ \sum_{\substack{C_A,C_B \in \mathcal{C} \\ R \preceq C_A \prec C_B}} P(m(t)=C_A) \cdot P(m(t')=C_B) \\
&= P(m(t)=C) \cdot \sum_{\substack{C_B\,: \\ C \prec C_B}} P(m(t')=C_B) + P_L(t' \sqsubset t) + P_R(t' \sqsubset t) \qquad (6)
\end{aligned}$$

Equation 6 can be written more concisely in matrix form: let $n_T = |\mathcal{T}|$ be the number of types, $\mathbf{Q_C}$ be a $n_T \times n_T$ matrix such that its $(i,j)$ coordinate contains $P_C(t_j \sqsubset t_i)$, and $\mathbf{p_C} = (P(m(t_i)=C))_{i=1,\dots,n_T}$ the $n_T$-dimensional probability vector for cluster $C$, we get the following formula for $\mathbf{Q_C}$:

$$\mathbf{Q_C} = \mathbf{p_C} \cdot \left( \sum_{C'\,:\,C \prec C'} \mathbf{p_{C'}} \right)^{\top} + \mathbf{Q_L} + \mathbf{Q_R} \qquad (7)$$

At this point, we're close to a recursive formulation for $\mathbf{Q_C}$, which would allow us to compute $\mathbf{Q}_{\texttt{root}}$ and thus the probabilities $P(t_j \sqsubset t_i)$ for all $i,j$, but we need to eliminate the sum $\sum_{C'\,:\,C \prec C'} \mathbf{p_{C'}}$ because it depends on all $C$'s successors and not only on $C, L$ and $R$. We denote this sum by $\mathbf{s_C}$: it is a $n_T$-dimensional vector and represents the probability for each type to be mapped to a strict successor of $C$. Since a strict successor of $C$ is either $L$, $R$ or a strict successor of $L$ or $R$, we have:

$$\mathbf{s_C} = \mathbf{p_L} + \mathbf{p_R} + (\mathbf{s_L} + \mathbf{s_R}) \qquad (8)$$

We finally define $\mathbf{u_C} = \mathbf{p_C} + \mathbf{s_C}$ the probability for each type to be mapped to a (non-strict) successor of $C$, so equation 8 can be rewritten as:

$$\mathbf{s_C} = \mathbf{u_L} + \mathbf{u_R} \qquad (9)$$

If $C$ is a leaf, the probability for any type $t$ to be mapped to a strict successor of $C$ is zero, since $C$ has no strict successor. For the same reason, the probability for any axiom $t' \sqsubset t$ to be valid within $X[C]$ is also zero. $\mathbf{s_C}$ and $\mathbf{Q_C}$ are thus zeros for any leaf cluster. We denote $\mathbf{0}_{n_T}$ and $\mathbf{0}_{n_T,n_T}$ the zero vector and zero matrix of dimension $n_T$ and $n_T \times n_T$ respectively. The final recursive equations for $\mathbf{s_C}, \mathbf{Q_C}, \mathbf{u_C}$ are:

$$\mathbf{Q_C} = \begin{cases} \mathbf{0}_{n_T, n_T} & \text{if } C \text{ is a leaf} \\ \mathbf{p_C} \cdot \mathbf{s_C}^\top + \mathbf{Q}_{\text{left}(C)} + \mathbf{Q}_{\text{right}(C)} & \text{otherwise} \end{cases} \tag{10}$$

$$\mathbf{s_C} = \begin{cases} \mathbf{0}_{n_T} & \text{if } C \text{ is a leaf} \\ \mathbf{u}_{\text{left}(C)} + \mathbf{u}_{\text{right}(C)} & \text{otherwise} \end{cases} \tag{11}$$

$$\mathbf{u_C} = \mathbf{p_C} + \mathbf{s_C} \tag{12}$$

With $\mathbf{p_C}$ the probability vector of cluster $C$:

$$\mathbf{p_C} = (P(m(t_i) = C))_{i=1,\ldots,n_T} \tag{13}$$

Our implementation uses linear programming. We denote $D_C = (\mathbf{s_C}, \mathbf{Q_C}, \mathbf{u_C})$. Clusters are iterated over in a reverse DFS order. For each cluster $C$, we compute and cache $D_C$ using $D_{\text{left}(C)}$ and $D_{\text{right}(C)}$. The reverse DFS order guarantees that all successors of $C$ have already been visited, so $D_{\text{left}(C)}$ and $D_{\text{right}(C)}$ are already cached. After computing $D_C$, we can delete cached values $D_{\text{left}(C)}$ and $D_{\text{right}(C)}$ because no further computation depends on them. The last visited cluster is `root`: when the algorithm ends, the cache contains only $D_{\texttt{root}} = (\mathbf{s}_{\texttt{root}}, \mathbf{Q}_{\texttt{root}}, \mathbf{u}_{\texttt{root}})$, and the final probability matrix $\mathbf{Q}_{\texttt{root}}$ can be returned. Its $(i, j)$ coordinate contains the probability of axiom $(t_j \sqsubset t_i)$. For a balanced binary tree, this algorithm guarantees a maximum cache size of $O(|\mathcal{T}|^2 \log_2(|\mathcal{C}|))$, with $|\mathcal{T}| \ll |\mathcal{C}|$.

*Taxonomy Extraction.* As in the deterministic approach, we have a list of subsumption axioms, but the taxonomy reconstruction differs, because we lost the injectivity of the mapping. We thus need a pruning step, to make sure the predicted taxonomy is indeed a tree.

We start from a directed acyclic graph $G = (\mathcal{T}, \emptyset)$ whose vertices are the types $t_i$, and without any edge. Then, we add axioms $(t_i \sqsubset t_j)$ to the graph by decreasing probability order: if adding the edge $t_j \to t_i$ creates a cycle, then we discard $t_i \sqsubset t_j$. Else we add the edge to $G$. We stop when the remaining axioms are below a given probability threshold $t$, or when no more edges can be added without creating a cycle. At each step, $G$ remains acyclic, so the final graph is acyclic as well.

By construction, $G$ is the transitive closure of the taxonomy we want to predict: if axioms $t'' \sqsubset t'$ and $t' \sqsubset t$ are both in $G$ (meaning they have high probabilities), then $t'' \sqsubset t$ should also have high probability and thus will likely be in $G$ too. This is because $P(t' \sqsubset t) = P(m(t') \succ m(t))$ and $\succ$ is transitive. Hence we need to compute the transitive reduction $T$ of $G$ to transform it into a taxonomy. Since $G$ is acyclic, this transitive reduction is unique [1].

Contrary to the previous method, we have no guarantee that $T$ is a tree. We thus compute its maximum spanning tree, using $P(t' \sqsubset t)$ as the weight of edge $t \to t'$ in order to maximize the overall probability of the predicted taxonomy. In this case, this is equivalent to removing, for each vertex $t_i$ that has an indegree larger than 1 (*i.e* it has several direct predecessors), all its incoming edges but the one of highest probability. The resulting tree is the predicted taxonomy $\hat{T}$.

## 4 Results and Experiments

### 4.1 Data

We used the 2016-10 release of DBpedia[2] to evaluate our approach, from which we removed all `rdfs:subClassOf` statements. The full DBpedia taxonomy[3] contains 589 classes spread over seven levels of hierarchy: there is one root class at level 0 (`owl:Thing`), 24 at level 1 (such as `dbo:Agent` or `dbo:Place`), and so on. Classes are strongly imbalanced: `dbo:Agent` has 1.2M+ instances, whereas `dbo:BowlingLeague` and `dbo:MouseGene` have less than five. The width (number of children) and the depth of each subtree also vary greatly. For example, `dbo:Agent`, `dbo:Device` and `dbo:Media` are all level-1 classes, but `Agent` has 275 subclasses spread over five levels of hierarchy, `Device` only has 13 subclasses and 2 levels, while `Media` has no subclass.

To evaluate our approach, we extracted a simpler taxonomy consisting only of the most frequent classes in the knowledge base and use it a our gold standard. To reduce the depth of the taxonomy, only 4 levels of hierarchy were considered. To reduce its width, we kept the 7 most frequent subclasses of each class. Finally, classes with less than 500 instances were removed. The resulting taxonomy has 84 classes (not counting the root class `owl:Thing`) and covers 75% of DBpedia `IS A` triples. In the following, this simpler, 84-classes taxonomy is referred to as DBpedia-Freq. For each type in DBpedia-Freq, we sampled entities of this type, resulting in a dataset $\mathcal{D}$[4] containing $N = 51,418$ entity-type pairs.

### 4.2 Evaluation

For each embedding model, we compare the Hard Mapping and Soft Mapping methods presented above with the state-of-the-art taxonomy extraction method TIEmb [19]. All three methods were tested with euclidean and cosine distance. For the clustering step of our methods, since Ward criterion is not defined for cosine distance, we fell back to average linkage instead.

To compare a predicted taxonomy $\hat{T}$ with the true DBpedia-Freq taxonomy $T^*$, we measure the differences between the predicted axioms and the true axioms using precision (number of predicted axioms that are true), recall (number of true axioms that are predicted), and F-score (geometric mean of precision and recall). We also compute the transitive closure of $\hat{T}$ and $T^*$: the transitive closure of a tree $T$ is a graph $T^+$ defined by $(t \sqsubset t') \in T^+ \iff \exists t_1, \ldots, t_k$ s.t. $(t \sqsubset t_1), (t_1 \sqsubset t_2), \ldots, (t_k \sqsubset t') \in T$. Then, once again, we measure the precision, recall and $F$-score between $\hat{T}^+$ and $(T^*)^+$. These two evaluations, with and without transitive closure, are called respectively *direct evaluation* and *transitive evaluation*.

[2] data can be downloaded at wiki.dbpedia.org/downloads-2016-10

[3] the latest version is available at mappings.dbpedia.org/server/ontology/classes and differs slightly from the 2016-10 one

[4] the dataset and the gold standard taxonomy can be found in the supplemental material folder

Both direct and transitive evaluations are needed to understand if a predicted taxonomy is close to the reference taxonomy. If the extraction model has correctly predicted most axioms but made mistakes on high-level ones (*e.g.* wrongly predicted `Person` $\sqsubset$ `Event`), it will have a high direct $F$-score, but a low transitive $F$-score, because high level errors will spread to lower levels when computing the transitive closure (in our example, all subtypes of `Person` will wrongly become subtypes of event `Event`, and precision will deteriorate). Conversely, if the model poorly handles the depth of the underlying taxonomy (*e.g.* predicted `SoccerPlayer` $\sqsubset$ `Person` and `Athlete` $\sqsubset$ `Person` instead of `SoccerPlayer` $\sqsubset$ `Athlete` $\sqsubset$ `Person`), it can achieve a high transitive $F$-score but will have a low direct $F$-score.

**Table 1.** Evaluation of our approaches and TIEmb over DBpedia-Freq, for different embedding models and different distance functions. $p, r, F$ stand for precision, recall and F-score respectively. cos and euc refers to cosine and euclidean distance. *Average* is the average of direct and transitive metrics.

| | | | Direct | | | Transitive | | | Average | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Embeddings | Distance | $p$ | $r$ | $F$ | $p$ | $r$ | $F$ | $p$ | $r$ | $F$ |
| TIEmb | ComplEx | cos | 0.38 | 0.38 | 0.38 | 0.28 | 0.57 | 0.37 | 0.33 | 0.48 | 0.38 |
| | | euc | 0.39 | 0.42 | 0.4 | 0.34 | 0.8 | 0.48 | 0.37 | 0.61 | 0.44 |
| | DistMult | cos | 0.26 | 0.27 | 0.26 | 0.19 | 0.44 | 0.27 | 0.23 | 0.36 | 0.27 |
| | | euc | 0.37 | 0.41 | 0.39 | 0.33 | 0.79 | 0.47 | 0.35 | 0.60 | 0.43 |
| | RDF2Vec | cos | 0.77 | **0.84** | 0.81 | 0.43 | 0.87 | 0.57 | 0.60 | 0.86 | 0.69 |
| | | euc | 0.70 | 0.77 | 0.73 | 0.30 | 0.73 | 0.42 | 0.50 | 0.75 | 0.58 |
| | TransE | cos | 0.77 | 0.72 | 0.74 | 0.83 | 0.89 | 0.86 | 0.80 | 0.81 | 0.80 |
| | | euc | 0.76 | 0.75 | 0.76 | 0.7 | 0.9 | 0.79 | 0.73 | 0.83 | 0.78 |
| Hard Mapping (HM) | ComplEx | cos | 0.53 | 0.5 | 0.52 | 0.78 | 0.7 | 0.74 | 0.66 | 0.60 | 0.63 |
| | | euc | 0.51 | 0.44 | 0.47 | 0.87 | 0.52 | 0.65 | 0.69 | 0.48 | 0.56 |
| | DistMult | cos | 0.55 | 0.47 | 0.5 | 0.73 | 0.57 | 0.64 | 0.64 | 0.52 | 0.57 |
| | | euc | 0.43 | 0.36 | 0.39 | 0.75 | 0.54 | 0.63 | 0.59 | 0.45 | 0.51 |
| | RDF2Vec | cos | 0.60 | 0.44 | 0.50 | 0.88 | 0.50 | 0.63 | 0.74 | 0.47 | 0.57 |
| | | euc | 0.60 | 0.53 | 0.56 | 0.89 | 0.64 | 0.74 | 0.74 | 0.58 | 0.65 |
| | TransE | cos | 0.82 | 0.8 | 0.81 | 0.97 | 0.89 | **0.93** | **0.90** | 0.85 | 0.87 |
| | | euc | 0.73 | 0.67 | 0.7 | **0.99** | 0.68 | 0.8 | 0.86 | 0.68 | 0.75 |
| Soft Mapping (SM) | ComplEx | cos | 0.52 | 0.48 | 0.5 | 0.89 | 0.7 | 0.79 | 0.71 | 0.59 | 0.65 |
| | | euc | 0.51 | 0.44 | 0.47 | 0.84 | 0.6 | 0.7 | 0.68 | 0.52 | 0.59 |
| | DistMult | cos | 0.5 | 0.44 | 0.47 | 0.87 | 0.65 | 0.74 | 0.69 | 0.55 | 0.61 |
| | | euc | 0.48 | 0.42 | 0.45 | 0.86 | 0.65 | 0.74 | 0.72 | 0.37 | 0.49 |
| | RDF2Vec | cos | 0.61 | 0.36 | 0.45 | 0.84 | 0.39 | 0.53 | 0.77 | 0.51 | 0.61 |
| | | euc | 0.61 | 0.47 | 0.53 | 0.92 | 0.55 | 0.69 | 0.74 | 0.58 | 0.65 |
| | TransE | cos | **0.83** | 0.81 | **0.82** | 0.93 | **0.93** | **0.93** | 0.88 | **0.87** | **0.88** |
| | | euc | 0.78 | 0.77 | 0.77 | 0.98 | 0.87 | 0.92 | 0.88 | 0.82 | 0.85 |

**Results.** The results are presented in Table 1. Soft Mapping (SM) has consistently higher $F$-scores than TIEmb and Hard Mapping (HM) for all embedding models, with the exception of the combination TIEmb/RDF2Vec. The best model overall is SM with TransE embeddings and cosine distance[5], with the highest F-score for both direct and transitive evaluation, but results for HM-TransE-cosine and SM-TransE-euclidean are close. Mapping models with TransE all have precisions within 93-99% on the transitive evaluation, meaning that few false axioms are predicted. TIEmb with RDF2Vec also achieves high scores in the direct evaluation, but has low transitive precision. This indicates that errors are made in the first levels of the predicted taxonomy. On average, TIEmb has better recall than precision.

Regarding embedding models, TransE yields significantly and consistently better results than the others, even though RDF2Vec also achieves high scores for some metrics. This can seem surprising, given that TransE is generally considered less expressive than DistMult, ComplEx and RDF2Vec. Yet, the simple geometric assumptions behind TransE forces the model to embed entities of the same type close to each other in the vector space. For two entities $h, h'$ of type $t$, we have:

$$\begin{aligned}\|\mathbf{h} - \mathbf{h}'\|_2 &= \|(\mathbf{h} + \mathbf{r}_{\text{IS\_A}} - \mathbf{t}) - (\mathbf{h}' + \mathbf{r}_{\text{IS\_A}} - \mathbf{t})\|_2 \\ &\leq \|\mathbf{h} + \mathbf{r}_{\text{IS\_A}} - \mathbf{t}\|_2 + \|\mathbf{h}' + \mathbf{r}_{\text{IS\_A}} - \mathbf{t}\|_2 \\ &\leq E(h, r_{\text{IS\_A}}, t) + E(h', r_{\text{IS\_A}}, t)\end{aligned}$$

Since $(h, r_{\text{IS\_A}}, t)$ and $(h', r_{\text{IS\_A}}, t)$ are both valid triples, $E(h, r_{\text{IS\_A}}, t) + E(h', r_{\text{IS\_A}}, t)$ is minimized during training, so the distance between $h$ and $h'$ is also minimized. The geometric translation of taxonomic relations is thus more straightforward than in other, more complex models. This would suggest that sacrificing some expressivity in exchange for more simplicity can be beneficial when designing an embedding model, especially if taxonomy matters.

## 4.3 Hyperparameters

The Hard Mapping method has no other parameter than the distance function and the linkage method, which have been discussed in the previous section. Soft Mapping has two more parameters: the softmax base $\beta$ and the probability threshold $t$. In this section, we discuss the choice of these two parameters for a Soft Mapping method with TransE embeddings and cosine distance. Figure 3 contains a summary of our results.

We tried different pruning thresholds in the interval $[0, 1]$ for $\beta = 100$ and ran our method on DBpedia-Freq. For transitive evaluation, as expected, increasing $t$ improves the precision but reduces the recall. In the range $[0.1, 0.7]$, these two effects make up for each other, so the $F$-score plateaus near its optimal value. Depending on the task, adjusting the pruning threshold within this interval can thus favour precision over recall (extracting less axioms with a higher confidence)

[5] the taxonomy predicted with this model can be found in the supplemental material folder

or the other way around. This ability gives more flexibility to the Soft Mapping method, compared to the Hard Mapping one. Outside of the $[0.1, 0.7]$ interval, drops of $F$-score are observed. For $t > 0.7$, the improvement in precision is not enough to balance the drop in recall, thus $F$ drops too: in this range, most remaining axioms are true, so increasing $t$ only results in discarding valid axioms. $t = 0$ also decreases the $F$-score, because axioms can then be predicted despite low scores, which decreases precision for only a marginal gain in recall.

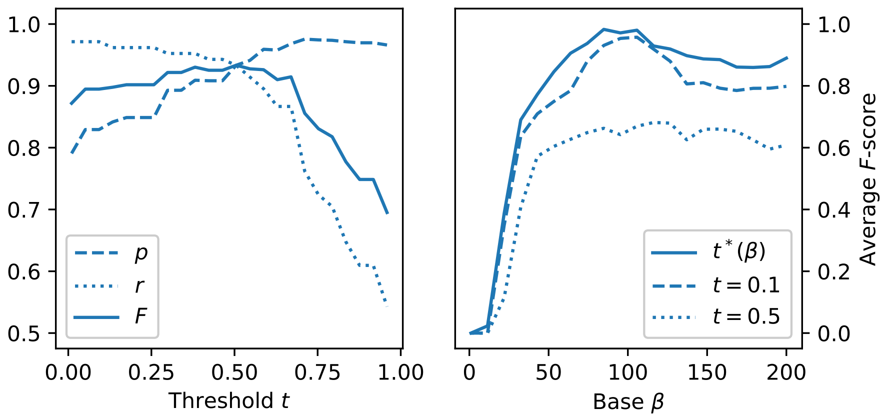

**Fig. 3.** Effect of hyperparameters $t$ and $\beta$ for the soft mapping method. *Left:* transitive evaluation metrics for different values of $t \in (0, 1)$ and $\beta = 100$. *Right:* $F$-scores obtained for different values of parameter $\beta$, with the optimal threshold $t^*(\beta)$ and two fixed threshold 0.1 and 0.5. Scores are averaged over five different subtaxonomies of DBpedia, and expressed as a fraction of the optimal score.

In order to choose the value of $\beta$, we created five smaller datasets on different subsets of DBpedia as in section 4.1, with a number of types ranging from 20 to 180. We ran the taxonomy extraction pipeline on each dataset, for different values of $\beta$ and for $t = 0.1$ or $t = 0.5$, and evaluated the predicted taxonomy using the average of direct and transitive evaluation results. Results were then averaged over the five datasets. We also computed, for each $\beta$, the optimal threshold $t^*(\beta)$, which gives us an upper bound of the model's performance for a given $\beta$.

For low values of $\beta$, the model is close to random and thus has low scores. Scores improve sharply in the $[5 - 75]$ interval. The highest scores are reached in the interval $\beta \in [75, 125]$. As for the pruning threshold $t$, we find that 0.1 is consistently better than 0.5, and especially for $\beta$ values in the $[75 - 125]$ range. On our five validation datasets, choosing of $\beta = 100, t = 0.1$ gives a $F$-score within $93\% - 100\%$ of the optimal $F$-score (97% on average).

## 5 Conclusion

In this paper, we proposed a method for extracting a taxonomy from knowledge graph embeddings. The method uses agglomerative clustering to create a hierarchy between groups of entities, and then turn this hierarchy into a taxonomy by mapping types to corresponding groups of entities, using either soft or hard mapping. We find this approach to work well, especially with TransE embeddings. We also observe that embedding models are not equally suited to this task. Moreover, the taxonomy extraction ability of a model is not reflected in standard benchmarks such as link prediction and triple classification. A natural extension to this work would be to design a benchmark for assessing the ability of an embedding model to embed taxonomic information, and to conduct a thorough evaluation of existing models on this task.

This work does not exhaust the possibilities offered by clustering over graph embeddings. We are working on extending the method proposed here by using Linked Data to generate rich descriptions of clusters and identify potential new types, which would lead to expressive taxonomy learning.

## References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM Journal on Computing **1**(2), 131–137 (1972)
2. Asim, M.N., Wasim, M., Khan, M.U.G., Mahmood, W., Abbasi, H.M.: A survey of ontology learning techniques and applications. Database (2018). https://doi.org/10.1093/database/bay101
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems. pp. 2787–2795 (2013)
4. Brown, P., Yang, Y., Zhou, Y., Pullan, W.: A heuristic for the time constrained asymmetric linear sum assignment problem. Journal of Combinatorial Optimization **33**(2), 551–566 (2017). https://doi.org/10.1007/s10878-015-9979-2
5. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: RDF2Vec DBpedia uniform embeddings (2017). https://doi.org/10.5281/zenodo.1318146
6. Faralli, S., Panchenko, A., Biemann, C., Ponzetto, S.P.: The ContrastMedium algorithm: Taxonomy induction from noisy knowledge graphs with just a few links. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. vol. 1, pp. 590–600 (2017)
7. Fu, R., Guo, J., Qin, B., Che, W., Wang, H., Liu, T.: Learning semantic hierarchies via word embeddings. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. vol. 1, pp. 1199–1209 (2014)
8. Gupta, N., Podder, S., Annervaz, K., Sengupta, S.: Domain ontology induction using word embeddings. In: 15th IEEE International Conference on Machine Learning and Applications. pp. 115–119. IEEE (2016). https://doi.org/10.1109/icmla.2016.0027
9. Han, X., Cao, S., Lv, X., Lin, Y., Liu, Z., Sun, M., Li, J.: OpenKE: An open toolkit for knowledge embedding. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 139–144. Association for Computational Linguistics (2018). https://doi.org/10.18653/v1/D18-2024

10. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th conference on Computational Linguistics. vol. 2, pp. 539–545. ACL (1992)
11. Jonker, R., Volgenant, A.: A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing **38**(4), 325–340 (1987)
12. Kozareva, Z., Hovy, E.: A semi-supervised method to learn and construct taxonomies using the web. In: Proceedings of the 2010 conference on Empirical Methods in Natural Language Processing. pp. 1110–1118. ACL (2010)
13. Lehmann, J.: DL-learner: learning concepts in description logics. Journal of Machine Learning Research **10**, 2639–2642 (2009)
14. Li, N., Bouraoui, Z., Schockaert, S.: Ontology completion using graph convolutional networks. In: International Semantic Web Conference. pp. 435–452 (2019)
15. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119 (2013)
16. Nickel, M., Kiela, D.: Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In: Proc. ICML (2018)
17. Nickel, M., Kiela, D.: Poincaré embeddings for learning hierarchical representations. In: Advances in Neural Information Processing Systems 30, pp. 6338–6347. Curran Associates, Inc. (2017)
18. Petrucci, G., Rospocher, M., Ghidini, C.: Expressive ontology learning as neural machine translation. Journal of Web Semantics **52**, 66–82 (2018). https://doi.org/10.1016/j.websem.2018.10.002
19. Ristoski, P., Faralli, S., Ponzetto, S.P., Paulheim, H.: Large-scale taxonomy induction using entity and word embeddings. In: Proceedings of the International Conference on Web Intelligence. pp. 81–87. ACM (2017). https://doi.org/10.1145/3106426.3106465
20. Ristoski, P., Paulheim, H.: Rdf2vec: RDF graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
21. Sertkaya, B.: Ontocomp: A protege plugin for completing OWL ontologies. In: European Semantic Web Conference. pp. 898–902. Springer (2009)
22. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: Proc. International Conference on Machine Learning (2016)
23. Völker, J., Niepert, M.: Statistical schema induction. In: Extended Semantic Web Conference. pp. 124–138. Springer (2011)
24. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering **29**(12), 2724–2743 (2017)
25. Wong, W., Liu, W., Bennamoun, M.: Ontology learning from text: A look back and into the future. ACM Computing Surveys **44**(4), 20 (2012)
26. Wu, F., Weld, D.S.: Automatically refining the wikipedia infobox ontology. In: Proc. 17th International Conference on World Wide Web. pp. 635–644. ACM (2008)
27. Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: Proc. 3rd International Conference on Learning Representations (2015)
28. Zafar, B., Cochez, M., Qamar, U.: Using distributional semantics for automatic taxonomy induction. In: Proc. 2016 International Conference on Frontiers of Information Technology. pp. 348–353. IEEE (2016). https://doi.org/10.1109/fit.2016.070