Taxonomy Extraction Using Knowledge Graph Embeddings and Hierarchical Clustering

Anonymous Author(s)

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 evaluate it on DBpedia. Our approach produces a taxonomy by leveraging the type information contained in the graph and the tree-like structure of an unsupervised hierarchical clustering performed over entity embeddings. We then extend our method with an axiom induction mechanism which allows us to identify new classes from the data and describe them with logical axioms, thus leading to expressive taxonomy extraction.

CCS CONCEPTS

• **Information systems** → **Clustering**; • **Computing methodologies** → **Ontology engineering**.

KEYWORDS

Knowledge graph embeddings, expressive taxonomies, ontology learning, hierarchical clustering.

ACM Reference Format:

Anonymous Author(s). 2021. Taxonomy Extraction Using Knowledge Graph Embeddings and Hierarchical Clustering. In *ACM/SIGAPP Symposium On Applied Computing, March 22–26, 2021, Gwangju, South Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

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 [2, 25]. Most of these approaches work directly at the level of classes [8, 12, 17], 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 new groups of entities that emerge from the content of the knowledge graph but have no corresponding class in the ontology. Since knowledge graphs are dynamic, relevant and irrelevant classes can change over time, so instance-based approaches can be a way to adjust ontologies to changes in their corresponding graph without human supervision.

In this paper, we propose a method to automatically extract a taxonomy from a knowledge graph, using unsupervised hierarchical clustering over embedding vectors. Embedding vectors of entities 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.

The goal of this paper is to show how such clustering tree can be used for taxonomy extraction. First, we propose two methods for turning a clustering tree into a taxonomy by mapping known types to relevant clusters. Then, we show that, contrary to methods based on supervised clustering, we're able to identify new classes from the data and label them with logical axioms, which leads to expressive taxonomy learning.

1.0.1 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 (or *classes*) 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 3.1.1.

1.0.2 Paper Outline. The reminder of the paper is organized as follows: in Section 2 we describe related work regarding taxonomy extraction and clustering over knowledge graphs. In Section 3 we explain our approach for extracting a taxonomy over known types and break it down to four main steps. We then extend it to the problem of *expressive* taxonomy learning. In Section 4, we evaluate our approach over a subset of DBpedia and compare it to the state of the art.

2 RELATED WORK

Extracting logical rules from data is an important research field. In its most general formulation, ontology generation aims at infering

ACM SAC '21, March 22–26, 2021, Woodstock, NY

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

logical rules from knowledge bases, or from other sources such as text corpora. This includes refining or completing an existing ontology [14, 26], 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.

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 and build a taxonomy from them. These approaches have to distinguish classes from instances among those words. Many modern methods use word embeddings: [8] recursively clusters embeddings of concepts to build a taxonomy, [7] or [28] learn a geometric relation between the embeddings of hyponyms and hypernyms. [16] and [17] 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: 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, which allows us to discover new types in the data. In Section 4, we compare TIEmb to our own approach.

3 PROPOSED APPROACH

3.1 Taxonomy on Named Classes

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 represented by its embedding $\mathbf{e}_i$, with $t_i \in \mathcal{T}$ its type. 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 immediate 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 our approach, the embedding vectors from $\mathcal{D}$ are first clustered agglomeratively without using type information. 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.

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

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: $\mathbf{h}, \mathbf{t} \in \mathbb{C}^d, \mathbf{W_r} \in \mathbb{C}^{d \times d}$ 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

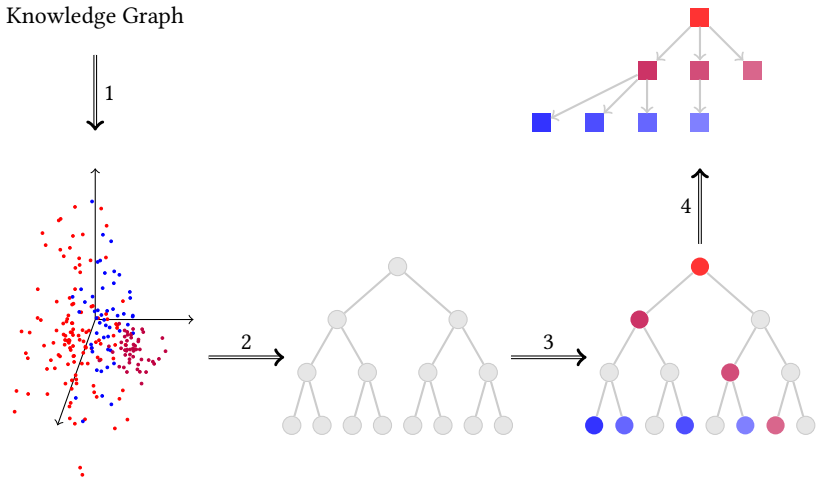

Figure 1: Overview of the proposed approach: starting from a knowledge graph $\mathcal{KG}$ and a set of entity-type pairs, (1) entities are embedded into a d-dimensional vector space (2) then they're hierarchically clustered; (3) each type in the original dataset is then mapped to one of the cluster,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

(*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.

3.1.2 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 C the set of all clusters, X the clustering tree, and `root` the root cluster. For two cluster C and C' such that $C' \subseteq C$, we say that C' is a *subcluster* of C and C a *parent cluster* of C'. $\subseteq$ defines a partial order over the tree.

3.1.3 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 subcluster of C will have both t_A and t_B as supertypes, and the resulting taxonomy will not be valid. The mapping thus needs 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:

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

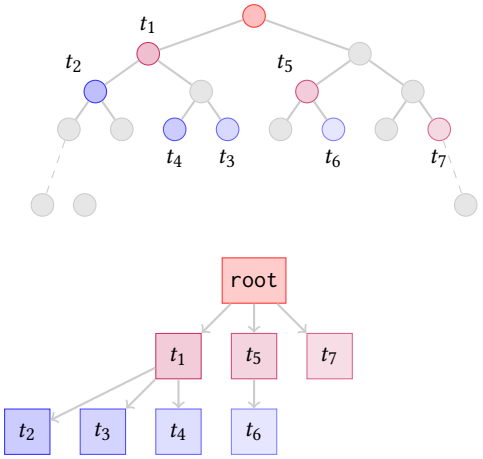

Figure 2: Extracting a taxonomy from a clustering tree using Hard Mapping: each type is mapped to one cluster in the clustering tree, which results in a partial labeling of the tree (top); then we extract the subtree containing only labeled clusters (bottom).

Finally, our optimal mapping is :

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

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 $O(|C|^3)$ [11]. If time were an issue, complexity could be brought down using heuristics, such as Asymmetric Greedy Search [4].

Once the mapping is completed, 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 $\exists t \in \mathcal{T}, m^*(t) = C$. Since there are more clusters than types, some clusters are left unlabeled. 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:

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

The β parameter controls the concentration of the distribution around the maximum value. $\beta \to 0$ gives all subsumption axioms the same probability, whereas $\beta \to \infty$ gives all the probability mass to the maximum value.

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 subcluster of the cluster $m(t)$ representing t. Under the hypothesis of independence between $m(t)$ and $m(t')$, we have:

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

We use linear programming to compute these probabilities for all t, t' without summing over all pairs of clusters. As in the Hard Mapping approach, this results in 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 δ, 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 $P(t' \sqsubset t) = P(m(t') \subseteq m(t))$ and $\subseteq$ 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 parents), all its incoming edges but the one of highest probability. The resulting tree is the predicted taxonomy $\hat{T}$.

3.2 Expressive Taxonomy Extraction

In the previous section, we presented a method to transform a clustering tree into a taxonomy over known types. However, the main interest of using unsupervised clustering is to be able to identify new classes from the data. Here, we propose a method to achieve this goal, by applying an axiom mining algorithm to each cluster in the clustering tree. Contrary to the previous case, the expressive method uses the entire graph, and not only the embeddings.

Learning an expressive taxonomy is done in two steps: first, entity embeddings are clustered hierarchically, thus creating a structure over groups of semantically close entities. Then, clusters are labeled with axioms whose score is above a given threshold (see section 3.2.1). To make the result more robust, these steps are repeated: new entities are sampled using the previously extracted axioms, and the clustering and labeling steps are repeated with these new entities, until no relevant cluster is found.

We start by randomly sampling n entities from the knowledge graph, and we embed them into the Euclidean space $\mathbb{R}^d$. As in the non-expressive case, this $n \times d$ point cloud is clustered hierarchically, resulting in a clustering tree X.

3.2.1 Axiom Mining for Cluster Labeling. Once the clustering tree is computed, it is searched and an axiom induction algorithm is ran over each cluster. When an axiom that accurately describes the elements of a cluster is found, the cluster is labeled with this axiom. Otherwise, it is left unlabeled. Extracted axioms indicate regularities in the triples involving the entities of each cluster, such as frequent classes or frequent relations. Once this labeling is done, the structure of the clustering tree naturally induces subsumption axioms between labels, *i.e.* an expressive taxonomy. The following paragraphs describe in more detail this axiom induction algorithm.

Problem Overview and Metrics. For a given cluster C, the goal is to find an axiom A such that A holds for as many elements of C as possible while being specific to it. To ensure this specificity, we define a set of *negative* examples, in addition to the *positive* examples that are the elements of C. Relevant axioms should hold for positive examples but not for negative examples. As negative examples, we chose to use the entities from the sibling cluster C' of C, that is the only cluster which has the same parent as C. Since C and C' are siblings, they have been merged at some point during the clustering step, which indicates a geometric proximity between their entities' embeddings, and thus a semantic similarity between the entities themselves. Hence, this choice of C' ensures that negative examples are increasingly specific as we search deeper into the tree. From now on, we denote E^+ and E^- the set of positive and negative examples, respectively.

We then define the *coverage* of axiom A as the proportion of elements in E^+ that verify A, and the *specificity* of A as the proportion of elements in E^- that don't verify A. We finally define a synthetic partition score for A as the arithmetic mean of coverage and specificity, weighted by the respective sizes of E^+ and E^-. This score varies within $[0, 1]$, with 1 being the optimal score. The goal is to induce axioms with high partition scores from the triples involving the entities in E^+ and E^-.

In this paper, axioms can be simple axioms (called *atomic axioms*), or a combination of these simple axioms. We consider two types of atomic axioms: named classes (concepts in description logic terminology), and existential restrictions. Three types of existential restrictions are considered: $\exists R.C$ with C a named class, $\exists R.\{v\}$ with v an entity, and $\exists R.t$ with t representing literals of type t (*e.g.* `xsd:date`). These atomic axioms can be combined with disjunctions and conjunctions, in order to create more complexe axioms.

Axiom Induction. To find axioms with high partition scores with respect to E^+ and E^-, we first extract atomic axioms from $E = E^+ \cup E^-$, and then improve them iteratively. For each example x in E, we extract all the triples $(x,\ R,\ y)$ in the graph. If R is the `rdf:type` relation, then y is a class, and the triple $(x,\ R,\ y)$ is transformed into the atomic axiom y (concept axiom). If y is a literal, its datatype t is extracted, and the triple is transformed into the restriction $\exists R.t$. Otherwise, the classes $C_1, \ldots, C_m$ of y are extracted, and the triple $(x,\ R,\ t)$ is transformed into atomic axioms $\exists R.\{y\}$ and $\exists R.C_1,\ \ldots,\ \exists R.C_m$. We end up with a list of atomic axioms for the entire set of examples. As a last pruning step, we remove atoms that are verified by less than 10% of the entities.

Next, we generate candidate axioms using this set of atomic axioms. At first, candidate axioms are the atomic axioms themselves. Each candidate axiom is evaluated using the metrics of specificity and coverage. We experimentally define a score threshold δ, typically $\delta = 0.9$. If the coverage is below the threshold, the axiom does not cover enough entities. We thus start improving an axiom a iteratively, by adding disjunctions (OR clauses): for each atomic axiom b that describes this cluster, we create a new candidate axiom $a \vee b$, which is in turn evaluated and extended if necessary. Conversely, if the specificity is below the threshold, the axiom does not separate well enough the two clusters, so we add conjunctions (AND clauses) as in the previous case. If both coverage and specificity are below the threshold, then the axiom is not a good candidate for explaining the split, and the search continues with other candidates. If both coverage and specificity are above the threshold, then the axiom is returned. At each step, only the N_{ax} candidate axioms with the highest scores are extended, to limit the size of the search space, where N_{ax} is a parameter. The search stops when the score improve by less than a given value *min_gain* from one step to the other, or when the number of steps exceeds a given value *max_steps*, with *min_gain* and *max_steps* two parameters.

We perform a depth-first search on the clustering tree, and run this axiom induction algorithm for each cluster. If an axiom with a score greater than δ is found, it is used as label. Else, the cluster is left unlabeled. The search stops after a given depth D, to avoid extracting overly specific axioms. At this stage, we end up with a clustering tree with some of the clusters associated to logical axioms.

3.2.2 Taxonomy Learning. The final step is to extract the expressive taxonomy itself. For this, we remove unlabeled clusters from the tree and connect labeled clusters in order to preserve the hierarchy between them, as in the Hard Mapping method. The result is a taxonomic tree $\hat{T}$.

All axioms from $\hat{T}$ are added to an axiom queue Q. The clustering and labeling steps are then repeated. While the axiom queue Q is not empty, an axiom A is dequeued from it. n entities are sampled

among those which verify A, and their embeddings are clustered, resulting in a clustering tree X_A. This clustering tree is labeled with logical axioms, then transformed into a taxonomy T_A. If T_A is not empty, all axioms in T_A are added to the queue Q, and T_A is added to the full taxonomy $\hat{T}$ (that is, all subsumption axioms contained in T_A are added to $\hat{T}$). When the axiom queue Q is empty, the algorithm stops and the expressive taxonomy $\hat{T}$ is returned. This resampling mechanism allows us to sample increasingly specific entities, and thus to extract increasingly specific axioms. The algorithm can be further refined by decreasing the score threshold δ over time.

4 EVALUATION METHODOLOGY

4.1 Evaluation of Taxonomies on Named Classes

4.1.1 Data. We used the 2016-10 release of DBpedia[1] to evaluate our approach, from which we removed all `rdfs:subClassOf` statements. The full DBpedia taxonomy[2] contains 455 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 of only the most frequent classes in the knowledge base and use it as 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. These parameters where chosen so as to cover 75% of DBpedia `IS-A` triples. The resulting taxonomy has 84 classes, not counting the root class `owl:Thing`. 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 $\mathcal{D}$[3] containing $N = 51,418$ entity-type pairs.

4.1.2 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. For the Soft Mapping method, we performed a grid search over five randomly sampled sub-taxonomies of DBpedia and found the optimal values for β and δ to be 100 and 0.1, respectively.

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 $(t \sqsubset t') \in T^+ \iff \exists t_1, \dots, t_k$ s.t. $(t \sqsubset t_1), (t_1 \sqsubset t_2), \dots, (t_k \sqsubset t') \in T$. 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*.

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`, 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.

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[4], 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.

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

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

[3] the dataset and the gold standard taxonomy can be found at zenodo.org/record/4031692

[4] the taxonomy predicted with this model can be found at zenodo.org/record/4031692

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.

Method	Embeddings	Distance	Direct			Transitive			Average		
			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

4.2 Evaluation of the Expressive Taxonomy

Assessing the quality of an expressive taxonomy is difficult because there is no gold standard available. In this section, we describe a quantitative evaluation of our approach on the "non-expressive" taxonomy extraction task, for which we have the DBpedia ontology as a gold standard, and we also provide a qualitative analysis of the expressive taxonomy that is obtained. In both cases, we use the TransE embedding model with cosine distance, since it achieves the best results in the previous section.

By restricting the extracted axioms to concepts (thus not considering existential relations), our algorithm can extract a taxonomy of named classes $\hat{T}$. As a gold standard, we use the DBpedia ontology T^* and remove classes that have no instances, resulting in 455 classes. To compare $\hat{T}$ and T^*, we use the evaluation metrics described above: precision, recall and F-score, computed on the taxonomies (direct evaluation) as well as on their transitive closures (transitive evaluation). Results are shown in Table 2. As we can see, the obtained F-score is slightly under 70 percent on the direct evaluation, but exceeds 90 percent on the transitive evaluation. This indicates that the general structure of the taxonomy is recovered correctly, but differences remain on the exact location of each class within the taxonomy. All these differences are not necessarily erroneous as they might reflect a new usage of the entities that was not foreseen in the gold standard taxonomy, which is the objective of our approach.

Table 2: Evaluation of the class subsumption axioms extracted, using DBpedia ontology as the gold standard. *Transitive* and *direct* indicates the evaluation with and without transitive closure respectively.

	Precision	Recall	F-score
Direct	68.94	68.79	68.87
Transitive	98.28	85.99	91.72

We also provide a limited qualitative analysis of the expressive taxonomy[5]. Compared to the DBpedia taxonomy, which contains 455 classes, our taxonomy adds 106 expressive classes. We can notice that unpreviously defined classes emerge from the data.

[5]Our expressive taxonomy can be found at zenodo.org/record/4031692, along with the hyperparameters used.

These emerging classes can subsume existing classes, such as the class Work ∧ ∃runtime.{xsd:double} which contains all entities from Film∨TVShow. We also obtain more specific subclasses such as Language∧∃spokenIn.Place that becomes a subclass of Language and is a sibling of ProgrammingLanguage.

We can also note that, at the first level, we are able to retrieve all the previously known DBpedia classes (Place, Event, Agent,...). Some classes that emerge highlight the need to complete the data instances. For example, the class Automobile is associated with the properties height, length and weight, but at least one of these properties is missing for 50% of its instances. Frequent classes appear as more accurately described than rare classes, as expected.

Overall, our generated taxonomy reflects the content of the knowledge graph, so our approach can be used to extract up-to-date schemas automatically. Some limitations include the difficulty to automatically determine the threshold at which to stop the decomposition of a cluster into further clusters and whether a class that emerges is valuable or only reflective of data incompleteness.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose 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.

We extend this work to the expressive taxonomy extraction task, by leveraging the structure of the clustering tree to define a small yet relevant set of positive and negative examples, and induce logical axioms from these examples. Our approach creates a rich, high-level view of the content of a graph without human supervision.

Future work includes refining the axiom induction process to increase the expressivity of the extracted axioms, and experimenting on other knowledge graphs with more embedding models.

REFERENCES

[1] Alfred V. Aho, Michael R Garey, and Jeffrey D. Ullman. 1972. The transitive reduction of a directed graph. *SIAM J. Comput.* 1, 2 (1972), 131–137.

[2] Muhammad Nabeel Asim, Muhammad Wasim, Muhammad Usman Ghani Khan, Waqar Mahmood, and Hafiza Mahnoor Abbasi. 2018. A survey of ontology learning techniques and applications. *Database* (2018).

[3] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems*. 2787–2795.

[4] Peter Brown, Yuedong Yang, Yaoqi Zhou, and Wayne Pullan. 2017. A heuristic for the time constrained asymmetric linear sum assignment problem. *Journal of Combinatorial Optimization* 33, 2 (2017), 551–566.

[5] Michael Cochez, Petar Ristoski, Simone Paulo Ponzetto, and Heiko Paulheim. 2017. RDF2Vec DBpedia uniform embeddings.

[6] Stefano Faralli, Alexander Panchenko, Chris Biemann, and Simone Paolo Ponzetto. 2017. The ContrastMedium algorithm: Taxonomy induction from noisy knowledge graphs with just a few links. In *Proc. 15th Conference of the European Chapter of the Association for Computational Linguistics*, Vol. 1. 590–600.

[7] Ruiji Fu, Jiang Guo, Bing Qin, Wanxiang Che, Haifeng Wang, and Ting Liu. 2014. Learning semantic hierarchies via word embeddings. In *Proc. 52nd Annual Meeting of the Association for Computational Linguistics*, Vol. 1. 1199–1209.

[8] Niharika Gupta, Sanjay Podder, KM Annervaz, and Shubhashis Sengupta. 2016. Domain ontology induction using word embeddings. In *15th IEEE International Conference on Machine Learning and Applications*. IEEE, 115–119.

[9] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. OpenKE: An Open Toolkit for Knowledge Embedding. In *Proc. 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 139–144.

[10] Marti A Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proc. 14th conference on Computational Linguistics*, Vol. 2. ACL, 539–545.

[11] Roy Jonker and Anton Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, 4 (1987), 325–340.

[12] Zornitsa Kozareva and Eduard Hovy. 2010. A semi-supervised method to learn and construct taxonomies using the web. In *Proc. 2010 conference on Empirical Methods in Natural Language Processing*. ACL, 1110–1118.

[13] Jens Lehmann. 2009. DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research* 10 (2009), 2639–2642.

[14] Na Li, Zied Bouraoui, and Steven Schockaert. 2019. Ontology completion using graph convolutional networks. In *International Semantic Web Conference*. 435–452.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[16] Maximillian Nickel and Douwe Kiela. 2017. Poincaré Embeddings for Learning Hierarchical Representations. In *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 6338–6347.

[17] Maximilian Nickel and Douwe Kiela. 2018. Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry. In *Proc. ICML*.

[18] Giulio Petrucci, Marco Rospocher, and Chiara Ghidini. 2018. Expressive ontology learning as neural machine translation. *Journal of Web Semantics* 52 (2018), 66–82.

[19] Petar Ristoski, Stefano Faralli, Simone Paolo Ponzetto, and Heiko Paulheim. 2017. Large-scale taxonomy induction using entity and word embeddings. In *Proc. International Conference on Web Intelligence*. ACM, 81–87.

[20] Petar Ristoski and Heiko Paulheim. 2016. Rdf2vec: RDF graph embeddings for data mining. In *International Semantic Web Conference*. Springer, 498–514.

[21] Barış Sertkaya. 2009. Ontocomp: A protege plugin for completing OWL ontologies. In *European Semantic Web Conference*. Springer, 898–902.

[22] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *Proc. International Conference on Machine Learning*.

[23] Johanna Völker and Mathias Niepert. 2011. Statistical schema induction. In *Extended Semantic Web Conference*. Springer, 124–138.

[24] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[25] Wilson Wong, Wei Liu, and Mohammed Bennamoun. 2012. Ontology learning from text: A look back and into the future. *Comput. Surveys* 44, 4 (2012), 20.

[26] Fei Wu and Daniel S Weld. 2008. Automatically refining the wikipedia infobox ontology. In *Proc. 17th International Conference on World Wide Web*. ACM, 635–644.

[27] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proc. 3rd International Conference on Learning Representations*.

[28] Bushra Zafar, Michael Cochez, and Usman Qamar. 2016. Using distributional semantics for automatic taxonomy induction. In *Proc. 2016 International Conference on Frontiers of Information Technology*. IEEE, 348–353.