Open Access

Open Journal of Semantic Web (OJSW)
Volume 4, Issue 1, 2017

http://www.ronpub.com/ojsw
ISSN 2199-336X

Scalable Generation of Type Embeddings Using the ABox

Mayank Kejriwal, Pedro Szekely

Information Sciences Institute, USC Viterbi School of Engineering, 4676 Admiralty Way, Marina Del Rey, CA, United States of America 90292, {kejriwal, pszekely}@isi.edu

Abstract

Structured knowledge bases gain their expressive power from both the ABox and TBox. While the ABox is rich in data, the TBox contains the ontological assertions that are often necessary for logical inference. The crucial links between the ABox and the TBox are served by is-a statements (formally a part of the ABox) that connect instances to types, also referred to as classes or concepts. Latent space embedding algorithms, such as RDF2Vec and TransE, have been used to great effect to model instances in the ABox. Such algorithms work well on large-scale knowledge bases like DBpedia and Geonames, as they are robust to noise and are low-dimensional and real-valued. In this paper, we investigate a supervised algorithm for deriving type embeddings in the same latent space as a given set of entity embeddings. We show that our algorithm generalizes to hundreds of types, and via incremental execution, achieves near-linear scaling on graphs with millions of instances and facts. We also present a theoretical foundation for our proposed model, and the means of validating the model. The empirical utility of the embeddings is illustrated on five partitions of the English DBpedia ABox. We use visualization and clustering to show that our embeddings are in good agreement with the manually curated TBox. We also use the embeddings to perform a soft clustering on 4 million DBpedia instances in terms of the 415 types explicitly participating in is-a relationships in the DBpedia ABox. Lastly, we present a set of results obtained by using the embeddings to recommend types for untyped instances. Our method is shown to outperform another feature-agnostic baseline while achieving 15x speedup without any growth in memory usage.

Type of Paper and Keywords

Regular research paper: *semantic embeddings, knowledge graphs, DBpedia, machine learning, probabilistic typing, entity typing, graph embeddings, word2vec, RDF2vec*

1 Introduction

Lately, the *distributional semantics* paradigm has been used with great effect in natural language processing (NLP) for embedding words in vector spaces [26]. The distributional hypothesis (also known as *Firth's axiom*) states that the meaning of a word is determined by its *context* [25]. Algorithms like word2vec use neural networks on large corpora of text to embed words in *semantic* vector spaces such that contextually similar words are close to each other in the vector space [16]. Simple arithmetic operations on such embeddings have yielded semantically consistent results (e.g. $King - Man + Woman = Queen$).

Recent work has extended such neural embedding techniques, traditionally introduced only for natural language word sequences, to alternate kinds of data, including entities in large knowledge graphs like DBpedia [22, 23]. The basic approach is to convert an

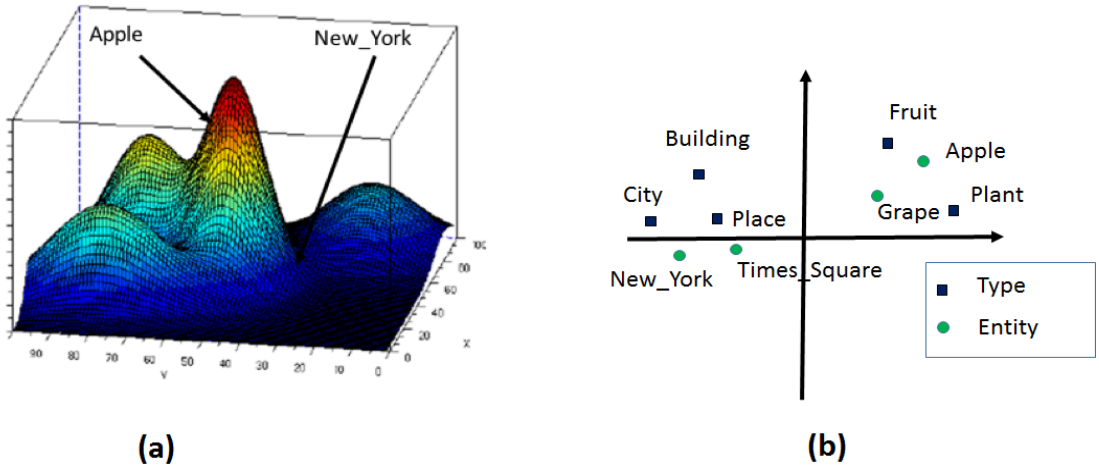


Figure 1: Generative story (a) and visual intuition behind semantic embeddings, and how such embeddings can co-exist with pre-generated entity embeddings in the same vector space (b).

instance-rich knowledge graph into sets of sequences of graph nodes by performing random walks or using graph kernels [12]. NLP algorithms like word2vec are applied on the sequences to embed entities, just like words in natural language sentences [23].

In the Semantic Web, the domain of discourse is typically expressed by a manually curated ontology. A basic element of an ontology is a *type*, also called a *class*. *Type assertion* statements relate entities (i.e. instances) in a knowledge base (KB) to the domain ontology, which can then be used to *infer* more facts about entities.

Given the crucial role played by types in mediating between domain ontologies and instance-rich KBs, a natural question is whether types can be embedded in the same semantic vector space as entities, and whether data-driven type embeddings can be used to reason about, and visualize, ontologies. For example, one could use these embeddings to ask whether the data adequately capture ontological semantics (Section 5.2), and to recommend types for new entities in the knowledge base (Section 6).

Unfortunately, type embeddings are difficult to directly derive from graphs because big knowledge graphs are sparse in type assertion statements compared to the number of unique instances and facts. In DBpedia, for example there are over 17 million (non-type) triples, and almost 4 million unique entities; the number of type assertion statements is also about 4 million, meaning that there is usually only one type assertion per entity. In many cases, the type assertions can be trivial (e.g. owl#Thing). Another problem is that types are typically asserted as objects, not subjects, in the KB; hence, a random walk cannot be initiated from a type node.

We propose a scalable solution to the problem of deriving type embeddings *from* entity embeddings in big graphs. A theoretical basis for the method is visualized in Figure 1 (a). Given a set of pre-generated entity embeddings, and a sparse collection of type assertion triples (a subset of the ABox), we are able to robustly generate embeddings for a set of types (e.g., *Fruit*, *Building* in Figure 1 (b)).

We validate the type embeddings derived using our method against a baseline that jointly embeds types and entities by combining the ABox and TBox. We show that our embeddings are able to capture TBox semantics (a level of the type hierarchy) much better than the baseline, despite having no knowledge of the TBox. Thus, in principle, our method can be used to embed types from different schemas and ontologies into the same space as a given set of entities. For example, both DBpedia types and schema.org types can be embedded (using our method) into the same embedding space as DBpedia entities.

We also illustrate an application for the proposed method, notably probabilistic type recommendation (e.g., recommending types such as *Fruit* and *Plant* for a new entity like *Grape*), and probabilistic type clustering over large graphs (Section 5). To the best of our knowledge, this is the first work that presents a *feature-agnostic* supervised typing of a large-scale semantic graph, DBpedia. Our algorithm is also extremely resource-efficient in terms of time and memory as it requires two linear passes over the full dataset. Empirical results on a partition of DBpedia, for example, show that our algorithm achieved run-time speedups by more than a factor of 15 on the type recommendation task compared to non-parametric nearest-neighbors baselines, with superior recall on two relevance criteria. The scalability of the model enabled us to probabilistically cluster

almost 4 million DBpedia instances into 415 types on a serial machine in under 50 hours.

This paper is an extended version of our recently published workshop paper [9]. Along with the contributions noted above, we list the following as additional contributions of this extended work:

1. A theoretical justification for the proposed embedding algorithm, and a section that explicitly presents methods for validating the model.
2. A significantly expanded related work section that places this work in a broader context than the previous paper.
3. Novel experimental results, especially for visualization and clustering. Unlike the workshop paper, we also compare the visualization against competitive baselines to illustrate the added value of our embedding generation.
4. More examples and intuition into the philosophy of the method, including an explicit motivation section.

2 Motivation

Latent space embeddings are not the only way to represent types. For example, one could represent a type by extracting features from the type, or alternatively, as a bag of relationships. However, such manual feature crafting methods, while not without merit, have two disadvantages. First, there is considerable manual effort involved in devising the 'right' features. This effort is well-recognized in the machine learning community for its ad-hoc, trial-and-error nature. Second, directly extracting features from the types do not make full use of the millions of facts in well-populated ABoxes.

In contrast, the latent space embeddings explored in this paper are *feature agnostic* i.e. they make use of the data in the ABox, along with a black box entity embedding algorithm like a neural network, to derive embeddings for types. Effort from a user, especially a user who may not be familiar with the Semantic Web or machine learning literature, is minimal as our algorithm takes few parameters as input. Effort is further minimized by the fact that many authors are increasingly choosing to publicly release entity embeddings that they have generated using their algorithms. For example, the authors of RDF2Vec [23] have released their embeddings for both Wikidata and DBpedia. While we do not use Wikidata for the experiments in this paper, we do make use of their publicly released DBPedia embeddings to show that re-generating entity embeddings is unnecessary both in theory and practice.

Also, because the type embeddings ultimately make use of the pre-generated entity embeddings, which are themselves derived by traversing the entire graph of ABox facts, the embeddings are *data driven*. In this sense, they are less prone to the biases of a feature engineer: the data dictates the features.

As further motivations for type embeddings, we illustrate several applications in which the embeddings can be used, most notably *type recommendation*. We also show that type embeddings can serve exploratory purposes e.g., we use them to visualize sub-type and super-type hierarchies. For example, in the clustering space, we find that types with a common super-type tend to be clustered closely together, while types with different super-types tend to be well separated. Such 'separation' is not well-achieved by naive baseline methods that seek to jointly embed entities and types using a single algorithm.

3 Related Work

Two broad research areas that are closely related to the work presented in this article are graph (including knowledge graph and network) embeddings, and entity typing. We provide coverage on both areas below. However, we note that, because embeddings are a very general framework, we use the embeddings not just for type embeddings but also visualization and online clustering, which cannot be handled by the other (special-purpose) type recommenders.

3.1 Graph Embeddings

Semantic vector space embeddings have witnessed much research in recent years, with neural word embedding algorithms (e.g. word2vec [16] and glove [18]) achieving state-of-the-art performance on a number of NLP tasks (e.g. dependency parsing) [2]. The success of word embedding approaches has led to a renewed interest in graph-based communities for embedding graphs. A famous example is DeepWalk [19], which applies word embedding techniques on random walk sequences on a graph to embed nodes in the graph to vectors.

In the Semantic Web, variants of this strategy were recently applied to DBpedia and Wikidata, and the embedded entities were used in several important problems, including content-based recommendation and node classification [23],[24]. Some other influential examples of such *knowledge graph embeddings* (KGEs), which is an active area of research, include (but are not limited to) systems such as TransE and TransH [11], [28], [1], [8]. An important aspect of this research is automatic knowledge base construction and completion

(AKBC), to which this work is related [27], [7]. A major difference is that, because of an additional layer of semantic abstraction (types vs. entities), we can afford to infer types without incrementally training the model such as in [10] or any other details of how the entity embeddings were derived. We also do not rely on natural language context of any kind [5].

In this article, we do not seek to develop a *new* learning algorithm for graph (including knowledge graph) or word embeddings; instead, the goal is to use an existing publicly available set of graph entity embeddings to extensionally model types. To the best of our knowledge, this is the first attempt to derive the embedding of *schema-level* elements (like types) directly using the embeddings of *instance-level* elements like entities. Because our method does not make underlying assumptions about the entity embeddings, it is general and can be applied to any set of entity embeddings.

3.2 Entity Typing

The type *recommendation* problem to which we apply the type models is closely connected to the type *prediction* problem studied in prior work, a good example being *Typifier* [13]. Unlike Typifier, which is not embedding-based and relies on manually devised features (e.g. data and pseudo-schema features [13]), our approach is feature-agnostic. Other examples of feature-centric type recommenders are the systems in [15], [17]. Due to the difficulty in automating feature construction, feature-agnostic systems are still quite rare; for example, in the Semantic Web, only a recent work achieved competitive performance at scale for feature-agnostic node classification [23].

In the Natural Language Processing (NLP) community, entity typing is an important problem. However, in the NLP community, the text context is very important in entity typing. For more details, we refer the reader to some recent work on entity typing in the NLP community [3], [29], [4], [21]. In the Semantic Web, a good example of a feature-rich entity typing system for entities in knowledge bases like DBpedia is *Tipalo* [6].

4 APPROACH

In this section, we detail our approach for generating type embeddings. First, we formally define our framework and introduce some related terminology (Section 4.1), followed by the actual algorithm in Section 4.2. A theoretical justification of the proposed model, as well as means for validating it, are covered in Sections 4.3 and 4.4 respectively.

4.1 Framework

We lay the groundwork in this section by formally defining a *typed* knowledge base (t-KB) and related terminology below:

Definition (typed knowledge base). Given a set $\mathcal{I}$ of Internationalized Resource Identifiers (IRIs), a set $\mathcal{B}$ of blank nodes and a set $\mathcal{L}$ of literals, a typed RDF Knowledge Base $\mathcal{T}$ is a set of RDF triples (i.e. $\subseteq \{\mathcal{I} \cup \mathcal{B}\} \times \mathcal{I} \times \{\mathcal{I} \cup \mathcal{B} \cup \mathcal{L}\}$) such that $\forall (s, p, o) \in \mathcal{T} \rightarrow \exists t \in \mathcal{I}, (s, : type, t) \in \mathcal{T}$, where $: type \in \mathcal{I}$ is a special type property (e.g. *rdf:type*).

We denote $(s, : type, t)$ as a *type assertion* statement, an arbitrary element s from the *entity set* $S = \{s|(s, p, o) \in \mathcal{T}\}$ as an *entity*, and the set $T(s) = \{t|(s, : type, t) \in \mathcal{T}\}$ as its set of asserted types[1]. Similar to the entity set S, we denote $T = \bigcup_{s \in S} T(s)$ as the *type set* of knowledge base $\mathcal{T}$. Finally, we denote a *type-only* KB (t-o-KB) $\mathcal{T}'$ as the subset of the typed knowledge base $\mathcal{T}$ that contains exactly the type assertions in $\mathcal{T}$.

In the literature, a knowledge base is typically meant to comprise an *ABox* and a *TBox*. Since the TBox contains the conceptualization of the knowledge base, s in the definition above would have to be a type for the type assertion (s,:type,t) to be included in the TBox. However, because we explicitly assume s to be an *entity*, the t-KB is necessarily a subset of the *ABox*. Although we do not use the TBox in the development of our framework, in Sections 4.4 and 5 and , we use the TBox to validate the methods presented in this article.

Although each entity s is *represented* by an IRI per the RDF data model, an alternate representation is as an *embedding* in a real-valued d-dimensional space:

Definition (entity embedding). A d-dimensional *entity embedding* representation of an entity s is a mapping from s to a real-valued vector $\overrightarrow{s}$ that is constrained to lie on the unit-radius hypersphere in d-dimensional space.

The constraint in the definition above ($\Sigma_i \overrightarrow{s}[i]^2 = 1$) ensures that the entity embedding is l2-normalized, which simplifies distance calculations considerably by equating cosine similarity between the two vectors with a dot product.

Concerning the actual learning of entity embeddings, we noted in Section 3 that recent work has successfully managed to learn embeddings (in spaces with only a few hundred dimensions) on large datasets like DBpedia by applying neural embedding algorithms like word2vec on graph node sequences [23]. Such embeddings may

[1] At present, we take an extensional (or instance-driven) view of a type by identifying it by its referents (the set of explicitly declared instances) of an entity s. We investigate an empirical relaxation of this condition in Section 5.

or may not include type assertions during training. A practical reason for not including types is that, in big knowledge graphs like DBpedia, the graphs $\mathcal{T}'$ and $\mathcal{T} - \mathcal{T}'$ are released as separate files (we provide links in Section 5), and it is more convenient to train embeddings over the latter. A more serious reason, pointed out in the introduction, is sparsity of assertions, data skew (a disproportionate number of entities are solely typed as owl:thing) and the observation that $T \cap S$ is typically empty.

To address problems of sparsity and robustness, we attempt to embed types into the *same* vector space as entities (thereby leveraging the enormous context of entities). Formally, we define a type embedding below.

Definition (type embedding). Given an entity set S and a type t, a d-dimensional *type embedding* representation of t is a mapping from t to a real-valued vector $\overrightarrow{t}$ that is constrained to lie on the unit-radius hypersphere in d-dimensional space.

Intuitively, a 'good' type embedding should have two elements: (1) be close in the vector space to entities that have that type. In Figure 1 (b), for example, *Fruit* is much closer to *Apple* than it is to *Times_Square*; (2) be closer to 'related' types than to types that are unrelated. In Figure 1 (b), *Building*, *City* and *Place* are all closer to one another than to either *Fruit* or *Plant*.

Clearly, the two elements above are related as they strongly rely on the data and on the context in which types are asserted or co-appear with entities that share context. In the next section, we explore how such robust embeddings can be scalably generated.

4.2 Generating Type Embeddings

Given the framework above, the primary research question addressed in this article is, *given a set of entity embeddings and a t-KB as inputs, how do we scalably generate a set of embeddings for the types in the ABox?*

Theoretically (and also empirically, as we illustrate in Section 5.2), there are at least two possible solutions to the problem. The first, which is not scalable, is to re-train the entity embeddings by combining the ABox and TBox (the *knowledge base*). One can do this in two ways. The first way is to ignore the given set of entity embeddings completely and re-run the embedding algorithm on the knowledge base. The second way is to use online machine learning optimization such as stochastic gradient descent (SGD) to update the original entity embeddings, and also generate new type embeddings. While the second way is more time-efficient than the first, the memory costs are extreme since all the neural network parameters have to be stored for the SGD to work.

In this article, we advocate a more *scalable* solution for the problem by treating the set of entity embeddings as a *black box*. In other words, we are agnostic to how they were generated, and we treat them as fixed in the rest of this discussion.

We propose a type embedding algorithm that is lightweight both in terms of run-time and memory. Algorithm 1 provides the pseudocode for our solution. Before describing the pseudocode, we describe the intuition as follows. A theoretical justification for the approach is described subsequently in Section 4.3.

Algorithm 1 relies on two assumptions. First, a type is ultimately described by its entities. This means that, with all things staying unchanged, a type should be close to as many entities having that type as possible. Second, a type should give more preference to entities that describe it exclusively. For example, suppose an entity s_1 has more than ten (explicitly declared) type assertions $\{t_1, t_2 \ldots t_{10}\}$ while the entity s_2 only has two type assertions $\{t_1, t_2\}$. Algorithm 1 is set up so that s_2 will contribute *more* to the derivation of t_1 and t_2 embeddings than s_1.

To operationalize these assumptions, while still being simple and scalable, Algorithm 1 computes a *weighted average* of entity embeddings to derive a type embedding. Specifically, in a first pass over a type-only KB $\mathcal{T}'$, the algorithm computes the number of types $|T(s)|$ asserted by the entity s. For each new type encountered, the algorithm initializes a zero vector for the type, in the same d-dimensional space as the entity embeddings. Even with millions of entities, this information can be stored in memory at little cost.

The second pass of the algorithm is *incremental*. For each triple $(s, :\ type, t)$, we update a type vector $\overrightarrow{t}$ (initialized to $\overrightarrow{0}$) using the equation:

$$\overrightarrow{t}_{new} = \overrightarrow{t}_{old} + \frac{1}{|T(s)|}\overrightarrow{s} \tag{1}$$

In the notation of the algorithm, $T_S[s] = T(s)$. Line 9 in Algorithm 1 shows a simple way of obtaining the final type embedding $\overrightarrow{t}$ by normalizing the 'final' mean vector $\overrightarrow{t}_{new}$ so that it lies on the unit-radius (d-dimensional) hypersphere. Normalizing ensures that the type embedding obeys the same constraints as the original entity embeddings and conforms to the type embedding definition earlier stated. A second reason for normalizing is that the computation of the cosine similarity between any vectors (whether type or entity) on the d-dimensional hypersphere reduces to the computation of the dot product between the vectors.

Algorithm 1 Generate Type Embeddings

Input: Sets S and $\vec{S}$ of entities and entity embeddings, type-only Knowledge Base $\mathcal{T}'$
Output: Type embedding $\overrightarrow{t}$ for each type t in $\mathcal{T}'$

1. Initialize empty dictionary T_S where keys are entities and values are type-sets
2. Initialize type-set T of $\mathcal{T}'$ to the empty set
   // First pass through $\mathcal{T}'$: collect entity-type statistics
3. **for all** triples $(s, : type, t) \in \mathcal{T}'$ such that $\overrightarrow{s} \in \vec{S}$ **do**
   Add t to T
   Add t to $T_S[s]$, if it does not already exist
4. **end for**
5. **for all** $s \in keys(T_S)$, set $T_S[s] = |T_S[s]|$ to save memory **end for**
   //Second pass through $\mathcal{T}'$ to derive type embeddings
6. Initialize Mean parameter dictionary M such that $keys(M) = T$, and each value in M is $\vec{0}$
7. **for all** triples $(s, : type, t) \in \mathcal{T}'$ such that $s \in S$ **do**
   Update $M[t]$ using Equation 1, using $T(s) = T_S[s]$
8. **end for**
   //Derive type embedding from $\overrightarrow{\mu_t}$
9. **for all** types $t \in keys(M)$ **do**
   Let type embedding $\overrightarrow{t}$ be the projection of $M[t]$ on d-dimensional hypersphere with unit radius (divide throughout by $||M[t]||_2$)
10. **end for**
11. **return** type embeddings derived in last step

4.3 Theoretical Justification: Generative Type Model

We now provide a theoretical basis for Algorithm 1 by proposing a generative story for the t-KB. Specifically, we will construct a generative type model or GTM for each type t. Intuitively, a generative type model 'explains', in a probabilistic sense, how an instance having type t was generated by the GTM of t.

We proceed as follows. Denoting GTM_t as the generative type model of type t and $|\mathcal{T}'(t)|$ as the number of t type assertions in the type-only KB $\mathcal{T}'$ provided during a *training phase*, one way to model GTM_t is to assume a non-parametric empirical distribution that assigns probability $1/|\mathcal{T}'(t)|$ to every entity $s \in S$ with asserted type $t \in T(s)$, and 0 probability to every other entity in S. Although the GTM model defined above is a valid probability distribution, it is problematic for various reasons.

First, the model does not consider the *number* of types that an entity participates in. In the real world, some entities are more unambiguously typed than others. For example, the entity *Antonio Conte* may be typed as a *Soccer Player*, *Soccer Manager* and as a *Person*. In an extensional sense[2], three different generative distributions (each corresponding to an asserted type) are contributing to the 'explanation' of the entity. In contrast, if the only asserted type of the entity *Brian Kerr* is *Soccer Manager*, he should contribute 'more' to the generative type model of *Soccer Manager*. The second problem with the model is that the number of parameters required to specify the GTM is proportional to the number of entities having that type; it is for this reason that the model is deemed to be *non-parametric*. The third challenge is that it is not obvious how such a model can be used to generalize to *unseen* data e.g. predicting or recommending types for a new entity during a *test phase*.

To address these challenges, we propose instead a parametric probabilistic distribution to express the GTM of a type. A simple, mathematically well-founded distribution is a multi-variate isometric Gaussian (for each type) in d-dimensional space. By isometric, it is meant that the covariance matrix $\vec{\Sigma}$ is diagonal (diag($\overrightarrow{\sigma_t^2}$)). Given this assumption, each GTM has $2d$ parameters (d-dimensional mean $\overrightarrow{\mu_t}$ and a d-dimensional variance vector $\overrightarrow{\sigma_t^2}$ with $\overrightarrow{\sigma_t^2}[i]$ representing the variance along the i^{th} dimension). We define the *single-type*-conditional probability $P(s|t)$ of an entity s with embedding $\overrightarrow{s}$ by the isometric Gaussian[3] $\mathcal{N}(\overrightarrow{x} =$

[2] Intensionally, the fact that Conte is a person can be derived using a domain ontology; however, neither of the other two classes can be derived from transitive closure.

[3] Recall that $\mathcal{N}(\overrightarrow{x}; \overrightarrow{\mu}, \vec{\Sigma}) = |2\pi\Sigma|^{-1/2} e^{-\frac{1}{2}(\overrightarrow{x}-\overrightarrow{\mu})'\Sigma^{-1}(\overrightarrow{x}-\overrightarrow{\mu})}$; in a slight abuse of notation, we denote $\vec{\Sigma}$ =diag($\overrightarrow{\sigma_t^2}$) as $\overrightarrow{\sigma_t^2}$.

$\overrightarrow{s}; \overrightarrow{\mu_t}, \overrightarrow{\sigma_t^2}$) and the *multi-type*-conditional probability as:

$$P(s|T(s)) = \Sigma_{t \in T(s)} \frac{1}{|T(s)|} \mathcal{N}(\overrightarrow{s}; \overrightarrow{\mu_t}, \overrightarrow{\sigma_t^2}) \quad (2)$$

Since $|T(s)|$ is not dependent on the summation, we can move it out:

$$P(s|T(s)) = \frac{1}{|T(s)|} \Sigma_{t \in T(s)} \mathcal{N}(\overrightarrow{s}; \overrightarrow{\mu_t}, \overrightarrow{\sigma_t^2}) \quad (3)$$

The equation above is a special case of a *Gaussian mixture model* (GMM) [20], which defines the probability as a weighted sum of Gaussian components, with the sum of the weights equating 1. In the general GMM model, the weights are unknown, whereas in the model expressed in Equation 3, the weights are equal. Earlier, Figure 1 (a) demonstrated a low-dimensional example ($P(s|Fruit, Plant)$).

One method to estimate the $2td$ parameter vector (denoted by $\overrightarrow{\theta}$) of the full type model is by *maximizing* a likelihood function over the the observed data $\mathcal{D}$ (the triples in $\mathcal{T}'$). Specifically, denoting a triple $(s, : type, t)$ as (s, t), the likelihood of $\mathcal{T}'$ is:

$$\mathcal{L}(\mathcal{D}; \overrightarrow{\theta}) = P((s,t)_1, \ldots, (s,t)_{|\mathcal{T}'|} | \overrightarrow{\theta}) \quad (4)$$

Assuming that entities are independently and identically distributed (i.i.d), and the type-conditional model in Equation 3, Equation 4 is expressed as:

$$\mathcal{L}(\mathcal{D}; \overrightarrow{\theta}) = \Pi_{s \in S} P(s, T(s) | \overrightarrow{\theta}) \quad (5)$$

$$\Pi_{s \in S} P(s, T(s) | \overrightarrow{\theta}) = \Pi_{s \in S} P(s|T(s), \overrightarrow{\theta}) P(T(s) | \overrightarrow{\theta}) \quad (6)$$

By setting $P(T(s) | \overrightarrow{\theta}) = P(T(s))$ to an empirically motivated frequency[4], we can solve for $\overrightarrow{\theta}$ by finding the values that maximize the log likelihood:

$$argmax_{\overrightarrow{\theta}} log(\mathcal{L}) = argmax_{\overrightarrow{\theta}} \Sigma_{s \in S} log(P(s|T(s), \overrightarrow{\theta})) \quad (7)$$

An optimal solution to Equation 7 is intractable as it has a strong non-linear dependence on the parameters. One option is to use a procedure like Expectation Maximization and derive iterative update equations. Unfortunately, iterative updates in high-dimensional parameter spaces are not feasible for millions of training points and hundreds of Gaussian components, as is the case in our setting. This is also true for other iterative algorithms that fix the parameters for all but one Gaussian component, optimize over that component and then repeat the procedure over all components till convergence.

The solution in Algorithm 1 is motivated by a similar intuition. Specifically, Algorithm 1 attempts a 2-pass approximation to Equation 7. To understand the intuition, suppose that $|T(s)| \leq 1$. Then, the GMM solution collapses to deriving Gaussian parameters (using maximum likelihood for optimizing) for each type independently of other types since two types never co-occur within the context of a single entity s (because $T(s)$ is constrained never to contain more than one type). The derivation has a known simple form: in particular, the mean of the entity vectors with type t yields the parameter μ_t.

Algorithm 1 uses this intuition to construct a weighted mean, as described earlier. The weight is analogous to the famous idf[5] term in information retrieval. Thus, if a type t participates in many type sets $T(s)$, it is assumed to not contribute significantly to any of them.

Algorithm 1 does not use the variance in any way. In principle, one could use both the mean and variance parameters of each GTM_t to infer a type embedding $\vec{t}$. However, in practice, such a formulation did not prove to be necessary. Simply using the mean μ_t as the type embedding $\vec{t}$ yielded excellent agreement with the TBox (see Section 4.4) in empirical evaluations, which implies that with sufficiently large knowledge bases, Algorithm 1 is robust enough that the variance ceases to matter.

4.4 Model Validation

An important question that arises concerning the treatments above is how the type embeddings derived using Algorithm 1 can be validated. Given that Algorithm 1, or the generative model in the previous section, did not use the TBox[6], we posit that a reasonable way of validating the *extrinsic* goodness of the type embeddings is to use the type hierarchy in the TBox. We do so in the following way. In many knowledge bases, instances are asserted in terms of very few types (out of hundreds of types) in the ABox. This is what we denote as *extensional* type semantics. Using the type hierarchy in the TBox, one can then perform transitive closure to infer more types for an instance *intensionally*. Since our type embedding algorithm does not have access to the TBox, it does not have access to the types obtained solely through transitive closure. Using this observation, the main validation question then is: does a set of *extensionally embedded* types with an immediate common super-type (i.e. a parent in the type taxonomy)

[4] A simple solution is to divide by $|S|$ the number of entities in S whose type-sets exactly equal $T(s)$.

[5] Inverse document frequency.

[6] The t-KB is a subset of the ABox, since the t-KB involves type assertions involving instances.

cluster together in the embedding space using a well-defined clustering/visualization measure? If they do, then the cluster represents the super-type: one could assert that the type hierarchy obtained *statistically* (via clustering) is in good agreement with a manually curated ontology. In the next section, we use this claim to empirically evaluate type embeddings derived using our method to those of a competitive baseline that jointly embeds the TBox and the ABox.

One could propose more complicated validation measures (e.g., computing a global probability for the full set of axioms in the TBox), or use an application (e.g., type recommendation) as a validation measure. We present some proof-of-concept results for the latter in a later section. Given that this is among the first attempts to propose the validation of derived type embeddings, we leave an investigation of the former for future work, as a theoretical justification of more complex validation metrics is beyond the scope of this article.

5 EXPERIMENTS

5.1 Preliminaries

Datasets: We construct five evaluation datasets by performing random stratified partitioning on the full set of DBpedia triples. We used the publicly available type-only KB[7] for our experiments from the October 2015 release of the English-language DBpedia. This file contains all the type assertions obtained only from the mapping-based extractions, without transitive closure using the ontology. Details of the five ground-truth datasets are provided in Table 1. Across the full five-dataset partition, there are 3,963,983 unique instances and 415 unique types. The five datasets are roughly uniform in their representation of the overall dataset, and not subject to splitting or dataset bias, owing to random stratified partitioning. In the experiments in Section 5.2, we derive type embeddings using the combined dataset (D1-5) and compare against two alternate type embedding methods in terms of the validation criteria outlined in Section 4.4. Subsequent to the empirical validation, we describe a potential entity typing application of our type embeddings. In a summary evaluation of the proof-of-concept evaluation, we use four partitions for training and another for testing. The DBpedia TBox which we use for the model validation in Section 5.2 may be dowloaded from the following link[8].

Entity Embeddings: The algorithms presented in this article assume that a set of entity embeddings has already been generated. Recently, [23] publicly made available two sets of 500-dimensional embeddings for DBpedia entities by using the word2vec algorithm on graph node sequences[9]. The word2vec model was trained using skip-gram, and was found to perform well on a range of node classification tasks. The authors referred to this algorithm as *RDF2Vec* in their paper. Rather than generate our own entity embeddings (which could potentially cause bias by overfitting to the type modeling task), we used those previously generated embeddings for all experiments in this article.

The difference between the two independently generated sets of entity embeddings that are used to evaluate the model in this article is that the first set, denoted *DB2Vec-types*, jointly embeds the TBox and ABox IRIs[10]. This was achieved by running RDF2Vec on the full knowledge base i.e. the combination of TBox and ABox assertions. The second set of embeddings, referred to as *DB2Vec*, only contains embeddings for entities in the ABox. No type embeddings were generated, since the authors removed all type assertions in the ABox before executing RDF2Vec. For validating our framework, we use both sets of embeddings for generating two sets of type embeddings (using Algorithm 1) and comparing both sets of results with the pre-generated type embeddings in *DB2Vec-types*. We respectively refer to these two sets of type embeddings as *DB2Vec-types-Alg1*[11] and *DB2Vec-Alg1*.

Implementation: All experiments in this article were run on a serial iMac with a 4 GHz Intel core i7 processor and 32 GB RAM. All code was written in the Python programming language. We used the *gensim* package[12] for accessing, manipulating and computing similarity on the entity embeddings. For details on the DB2Vec embeddings, we refer the reader to the empirical section of the RDF2Vec paper [23].

[7] Accessed at `http://downloads.dbpedia.org/2015-10/core-i18n/en/instance_types_en.ttl.bz2`

[8] `http://downloads.dbpedia.org/2015-10/dbpedia_2015-10.nt`

[9] Accessed at `http://data.dws.informatik.uni-mannheim.de/rdf2vec/models/DBpedia`

[10] However, we note that, because of word2vec parameter settings that the authors of [23] used to train the models, this does not guarantee that every IRI has an embeddings. IRIs that are too sparse will not be embedded as there is not enough data. This sparsity also affects evaluations, as we later illustrate.

[11] Note that, for this case, the original type embeddings (expressed by *DB2Vec-types*) are overwritten by the type embeddings generated by Algorithm 1. The embeddings are different from those in *DB2Vec-Alg1* because the entity embeddings are different (due to non-exposure of the latter to TBox type contexts).

[12] `https://pypi.python.org/pypi/gensim`

Table 1: Details of ground-truth datasets. The five datasets together comprise a partition of all (extensional) type assertion statements available for DBpedia.

Dataset	Num. triples	Num. unique instances	Num. unique types	Size on disk (bytes)
D-1	792,835	792,626	410	113,015,667
D-2	793,500	793,326	412	113,124,417
D-3	793,268	793,065	409	113,104,646
D-4	793,720	793,500	410	113,168,488
D-5	792,865	792,646	410	113,031,346

5.2 Experiment 1: Clustering and Visualizing the Extensional Model

Note that our methods never relied on the DBpedia ontology (the TBox) when deriving embeddings and generative model parameters. As described in Section 4.4, one reasonable way to validate the model is to use the intensional semantics of types (e.g. sub-class relationships in the ontology) to visualize the embeddings derived from extensional assertions (the mapping-based type assertion extractions). We perform two visualization experiments using the *unsupervised* t-SNE algorithm [14], a state-of-the-art tool for visualizing high-dimensional points on a 2-D plot. We compare three sets of derived type embeddings: *DB2Vec-types* (the baseline) and the two sets of type embeddings generated using Algorithm 1 (*DB2Vec-types-Alg1* and *DB2Vec-Alg1*) described in the *Entity Embeddings* section.

We conduct two experiments using each of these three sets of embeddings to answer two research questions: (1) Does exposure to the TBox actually 'matter' for Algorithm 1 to derive good type embeddings or does exposure to the ABox alone suffice? (2) Can Algorithm 1 perform as well as, or even outperform, *DB2Vec-types*, which *directly* generates type embeddings? We use two evaluation datasets to answer these questions. Since the model validation, and the goodness of the type embeddings, ultimately depends on how 'well' we capture the intensional semantics of the TBox, we use the t-SNE algorithm to visualize the performance of each of the embeddings [14]. Although very different from each other, both evaluation datasets involve five super-types in the DBpedia TBox.

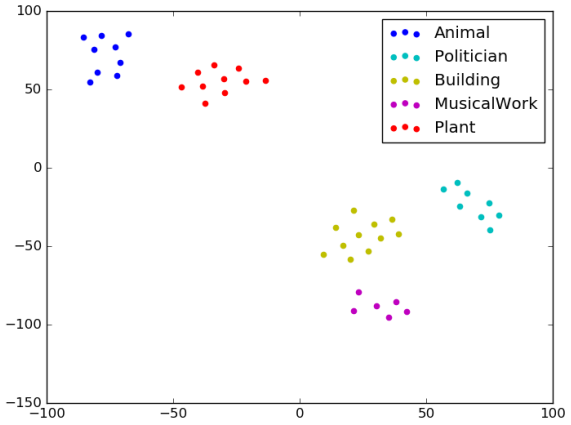
Dataset 1: We apply t-SNE on a matrix containing the type embedding vectors of all direct sub-types of five sampled types from the DBpedia ontology, namely *Animal, Politican, MusicalWork, Building* and *Plant*. The t-SNE algorithm takes the matrix as input and returns another matrix with the same number of rows but only 2 columns. We plot these points (rows) in Figure 2, by assigning labels (i.e. colors) to points[13] based on their super-type. The results in Figure 2 (a) and (b) show five well-separated clusters, with each cluster representing a super-type. In these cases, the extensional model is in excellent agreement with the intensional model. Since Figure 2 (a) and (b) respectively represent the results for *DB2Vec-Alg1* and *DB2Vec-types-Alg1*, this helps us answer the first research question: Algorithm 1 is relatively insensitive to whether entities were exposed to types during the execution of graph embedding. We also note that the clusters in Figure 2 (a) and (b) also demonstrate other interesting aspects not captured intensionally: e.g. *Building, Politician* and *MusicalWork* (artificial constructs) are much closer to each other, than they are to *Animal* and *Plant* (natural constructs), which form a separate 'super' cluster.

Cluster separation is much worse in Figure 2 (c), which represents the *DB2Vec-types* baseline. The type embeddings are not well separated, illustrating the lack of agreement with TBox semantics[14].

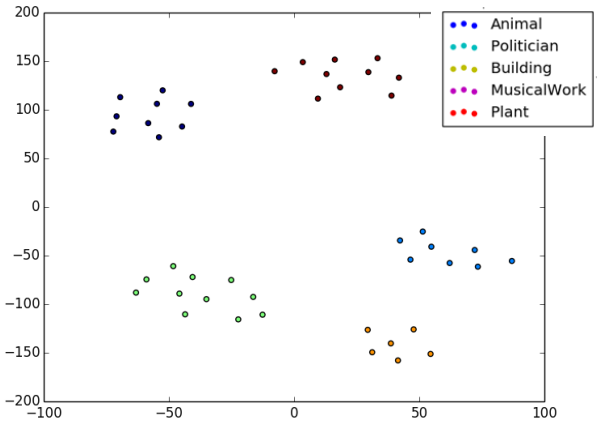
Dataset 2: We re-run the experiment but over sub-type embeddings of the types *SportsTeam, SportsLeague, Company, Organisation* and *EducationalInstitution*. Note that this set is much more *topically coherent* than the earlier set. The 2D visualization is illustrated in Figure 3; the topical coherence (there are now two clusters rather than five) is well-reflected in (a) and (b). The two purple 'outliers' on the left cluster are the embeddings for *SportsTeam* and *SportsLeague*, which are themselves sub-types of *Organisation*. Similar observations apply to (b); again, we note that exposure to the TBox during entity embedding generation did not affect the results of Algorithm 1. Because of data sparsity, little data was available from DB2Vec-types. The figures illustrate that our algorithm is able to deal more effectively with sparsity than the direct type

[13] Because t-SNE is unsupervised, it never accessed the labels during the clustering.

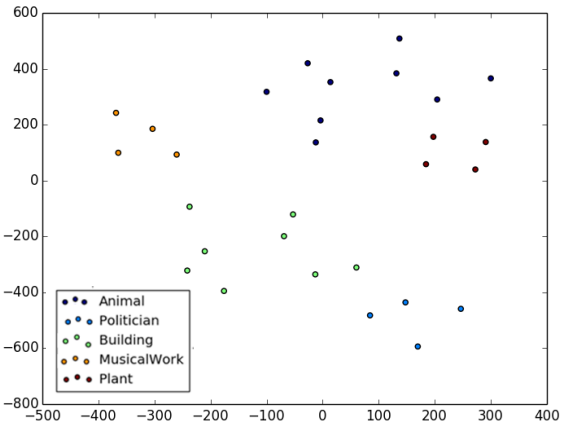
[14] Not all types in the TBox got embedded by RDF2Vec in the dataset released by the authors [23]. This may be due to pruning during model training e.g., in the word2vec model that underlies RDF2Vec, a common technique is to remove words that occur too few times in the dataset.



(a) DB2Vec-types-Alg1

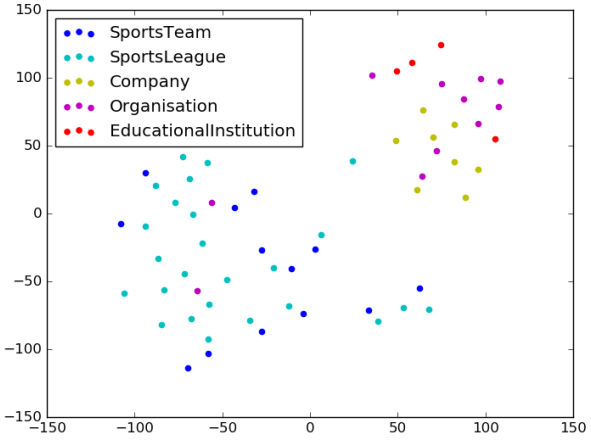


(b) DB2Vec-Alg1

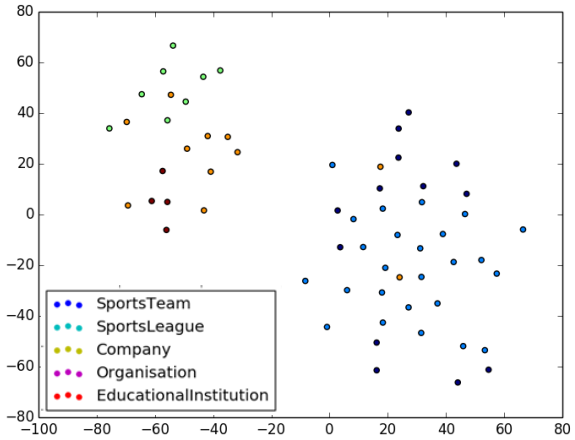


(c) DB2Vec-types

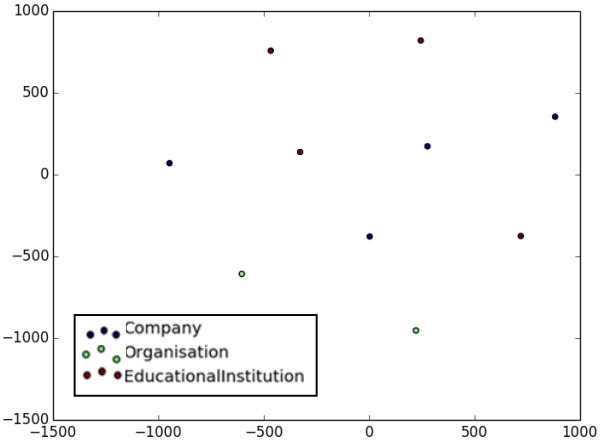
Figure 2: Visualization and clustering results on Dataset 1 using the t-SNE package. The dimensions in the figure do not have any intrinsic meaning.



(a) DB2Vec-types-Alg1



(b) DB2Vec-Alg1



(c) DB2Vec-types

Figure 3: Visualization and clustering results on Dataset 2 using the t-SNE package. The dimensions in the figure do not have any intrinsic meaning.

embedding generation. Combined with the observations from Dataset 1, the use of Algorithm 1 seems to be favored over direct type embedding generation.

5.3 Experiment 2: Probabilistic Typing of DBpedia Instances

The results of Experiment 2 in Section 6 illustrated that there are many types in the DBpedia ontology that are clearly related to an entity of interest, even if they are not super-types (or even sub-types) of the entity's type. Given an entity, we ideally want to assign a probability to *each* type. Such a clustering has much potential, including topic modeling of entities and fuzzy reasoning. To achieve a full fuzzy clustering over DBpedia, we used the union of all five datasets in Table 1 to compute our type embeddings, and then executed the fuzzy clustering algorithm over all DBpedia entities in the union of all five datasets. The algorithm scaled near-linearly and was able to finish executing over almost 4 million entities and 415 clusters (types) in about 50 hours, collecting over 100 GB of data[15]. To illustrate the quality of the probabilistic typing, we provide ten examples in Table 2, along with the top five types. We provide a formal evaluation, compared to a nearest neighbors baseline, in Section 6.

The results in Table 2 illustrate the advantage and robustness of fuzzy typing. For example, it is technically more appropriate for 'Animation' to be typed as an OWL *thing*, since it does not seem to have an evident type. However, the options listed in Table 2 make intuitive sense in that they contribute probabilistically to the definition of Animation. While the fuzzy clustering can be used for entity typing or type recommendation (see Section 6), we note that it has many other applications as well e.g., answering probabilistic queries or performing link prediction between entities by using type probabilities as *explanations* (not dissimilar to the generative theoretical justification of Algorithm 1 in Section 4.3).

6 Proof-of-Concept Application: Type Recommendation

In this experiment, we evaluate Algorithm 1 on the *probabilistic type recommendation* task. The input to the recommendation system is an entity, and the output must be a set of scored (or ranked) types that are *topically relevant* to the entity. The issue of relevance, by its nature, is subjective; we present a methodology for evaluating subjective relevance shortly.

[15] We will make this data available on a public server in the near future.

Baseline: We employ baselines based on weighted k Nearest Neighbors (kNN). The kNN algorithm is a strong baseline with some excellent theoretical properties: for example, even the 1NN algorithm is known to guarantee an error rate that is at most twice the Bayes error rate[16] in a given feature space and for a given distance function in the space. Because the entity embeddings are given, the kNN baseline, just like the embedding method, is *feature-agnostic*; to the best of our knowledge, it is the only baseline that has this property and is not super-linear. We consider three versions with k set to 1, 5 and 10 respectively.

Baseline Complexity: Compared to the embedding method, kNN has high time and memory complexity since it is *non-parametric* and requires storing all the (training) entities in memory. In contrast, after computing the type embeddings, the embedding method has to store $|T|$ vectors, one for each type. In terms of run-time, a full pass is required over the training dataset for *each* new test entity, regardless of the value of k. We explore the empirical consequence of this in our results.

Application Experiment 1: We perform five experiments, where in each experiment, four partitions (from Table 1) are used as the training set, and 5000 entities are randomly sampled from the remaining partition and used as the test set. We were forced to constrain the test set to 5000 for this initial experiment because of baseline complexity.

This experiment adopts an extremely *strict* definition of relevance: namely, the only relevant types are the ones that are *explicitly* asserted for the entity in the test partition. Thus, even a super-type (of the true asserted type) would be marked as 'irrelevant' if not explicitly asserted itself, since we do not consider transitive closures in the type-only KB. Although a strict measure, it provides a reasonable first benchmark, as it conforms to extensional semantics.

We evaluate performance using the *Recall@k*[17] measure from Information Retrieval. Recall@k computes, for each rank k in a ranked list of types, the ratio of true positives to the sum of true positives and false negatives. For each one of the five experimental runs, we compute a single Recall@k measure (for each k) by averaging the Recall@k over all 5000 entities. Finally, we compute the average across the experimental runs, and plot the results in Figure 4 (a). We also plotted individual plots for all five experimental runs, which turned out to be qualitatively very similar to Figure 4 (a). We omit those figures and a per-run analysis herein due

[16] This is the minimum possible error for a particular distribution of data.

[17] k in Recall@k should not be confused with k in kNN.

Table 2: Example results of probabilistic entity typing of DBpedia instances. Probabilities are not shown. Each entity in this representation is represented by a 415 dimensional type probability vector, since 415 TBox types in DBpedia have extensional participation (i.e. are asserted at least once in the ABox). In the last line, we list both *ComedyGroup* and *Manga*, since they were assigned equal probabilities (tied for fifth place).

Entity	Top 5 types
http://dbpedia.org/resource/Acid	[u'http://dbpedia.org/ontology/ChemicalCompound'] [u'http://dbpedia.org/ontology/Mineral'] [u'http://dbpedia.org/ontology/Drug'] [u'http://dbpedia.org/ontology/Protein'] [u'http://dbpedia.org/ontology/HumanGene']
http://dbpedia.org/resource/Asia	[u'http://dbpedia.org/ontology/Continent'] [u'http://dbpedia.org/ontology/Sea'] [u'http://dbpedia.org/ontology/EthnicGroup'] [u'http://dbpedia.org/ontology/Country'] [u'http://dbpedia.org/ontology/Gnetophytes']
http://dbpedia.org/resource/Albert_Sidney_Johnston	[u'http://dbpedia.org/ontology/MilitaryPerson'] [u'http://dbpedia.org/ontology/MilitaryUnit'] [u'http://dbpedia.org/ontology/MilitaryConflict'] [u'http://dbpedia.org/ontology/Senator'] [u'http://dbpedia.org/ontology/Governor']
http://dbpedia.org/resource/Aikido	[u'http://dbpedia.org/ontology/Sport'] [u'http://dbpedia.org/ontology/SumoWrestler'] [u'http://dbpedia.org/ontology/VoiceActor'] [u'http://dbpedia.org/ontology/AnimangaCharacter'] [u'http://dbpedia.org/ontology/OlympicEvent']
http://dbpedia.org/resource/Animation	[u'http://dbpedia.org/ontology/HollywoodCartoon'] [u'http://dbpedia.org/ontology/ComicsCreator'] [u'http://dbpedia.org/ontology/Comic'] [u'http://dbpedia.org/ontology/ComedyGroup', u'http://dbpedia.org/ontology/Manga']

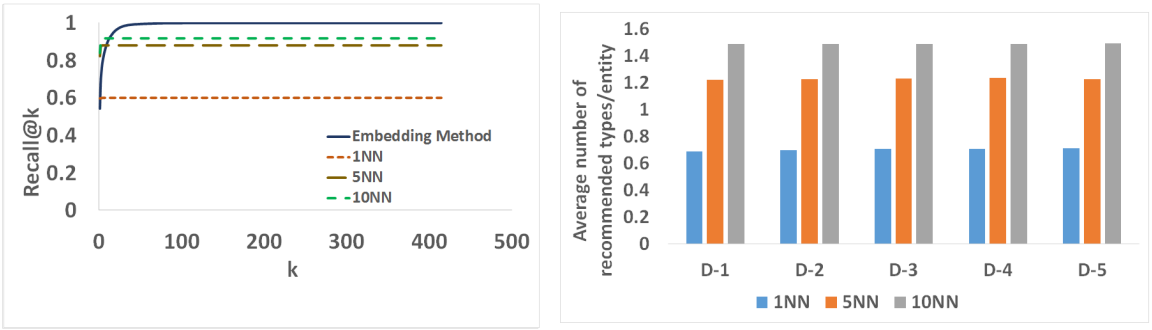
to space constraints.

Analysis: Figure 4 (a) shows that, even with the strict definition of relevance, although the embedding method starts out with low recall at the highest ranks, it converges with the other methods fairly quickly (between ranks 3-13). Figure 4 (b) shows that the baselines return very few non-zero recommendations per entity (fewer than 1.5) and the returned number is *sub-linear* in k: in this respect, the baselines perform 'hard' type predictions rather than graded recommendations. In contrast, the embedding method returns a more nuanced probability distribution over the 415 types (per entity), and is more apt for recommendations, as we show in the next experiment.

Application Experiment 2: Although Experiment 1 is appropriate for determining the extensional types of an entity, it takes an overly strict reflection of relevance. Considering the number of triples and unique instances in Table 1, there was usually only one extensional type asserted in the knowledge base. For a better judgment of relevance, we randomly sampled 100 instances from D-1 and pruned the ranked type lists for 10NN (clearly the best performing method in Figure 4 (a)) and the embedding method to 10. Because the number of returned types for 10NN was often fewer than 10, we randomly 'padded' the rest of the list with DBpedia types, and manually counted the number of topically relevant recommendations in each (10-element) list[18]. This allows us to compute a single-point Recall@10 score over the 100 sampled entities. Note that the random padding can only help, not hurt, the Recall@10 score of the baseline. We also asked the annotator to do a side-by-side comparison of the two lists (for each of the 100 entities) and mark the list that is more *topically useful* overall. The annotator was not given the details of the two ranking algorithms; also, all annotations were conducted within a single short time-span.

Analysis: The single-point Recall@10 was 0.4712 for the embedding method (averaged across the 100

[18] The annotations were performed externally, not by the authors.



(a) (b)

Figure 4: Application Experiment 1 results over all five datasets in Table 1. (a) plots the average Recall@k across all five experimental runs, while (b) illustrates average number of recommended types per entity for each of the datasets and baselines.

Table 3: Top 3 type recommendations for the embedding method and 10NN for five entities from the 100-sample dataset (Experiment 2)

Entity	Embedding Method Rec.	10NN Rec.
Shenyang_J-13	Aircraft, Weapon, Rocket	Aircraft, Weapon, Photographer
Amtkeli_ River	River, BodyOfWater, Lake	River, Dam, Mountain
Melody_ Calling	Album, Single, Band	Album, Single, PublicTransitSystem
Esau_ (judge_royal)	Monarch, Loyalty, Noble	Noble, Journalist, AdministrativeRegion
Angus_ Deayton	Comedian, ComedyGroup, RadioProgram	Person, Actor, TelevisionShow

manually annotated samples) and 0.1423 for the 10NN with standard deviations of 0.2593 and 0.0805 respectively. Some representative results for both 10NN and the embedding method are presented in Table 3. The table presents some intuitive evidence that types recommended by the embedding method are more topically relevant than the types recommended by 10NN. The annotator found the embedding method to be topically more useful for 99 of the 100 entities, with 10NN more useful on only a single entity.

Run-times: Concerning the empirical run-times, all baseline methods required about 1 hour to process 5000 test instances for the 4-partition training set while the embedding method only required 4 minutes. If the training set is fixed, all methods were found to exhibit linear dependence on the size of the test set. This illustrates why we were forced to sample 5000 test instances (per experimental run) for evaluating kNN, since predicting types on only one of the full five partitions in Table 1 would take about 150 hours, which is untenable, even if only strict type predictions (i.e. assertions) are of interest.

7 Conclusion

In this article, we developed a framework for deriving type embeddings in the same space as a given set of entity embeddings. We devised a scalable data-driven algorithm for inferring the type embeddings using only the assertions in the ABox. We evaluated the algorithm using the DBpedia ABox, against the DBpedia TBox, by comparing to a competitive baseline that infers type embeddings from scratch by re-executing the RDF2Vec on the entire knowledge base. Visualizations using the t-SNE algorithm illustrate the potential of the method. Furthermore, we also applied the algorithm to a probabilistic type recommendation task on five DBpedia partitions. Compared to a kNN baseline, the algorithm yields better results on various relevance criteria, and is also significantly faster.

Future Work. There are many avenues for future work that we have already started exploring. First, we are using the methods in this article to embed entire ontologies (collections of types and properties) into vector spaces, to enable a combination of distributional and ontological semantic reasoning. Second, we are exploring more applications of embedded types, such as an enhanced version of semantic search, and semantically guided information extraction from

structured data. Last, but not least, we are conducting broader empirical studies e.g. on datasets other than DBpedia and using knowledge graph embeddings other than the DeepWalk-based RDF2Vec to test the robustness of our type embedding approach to such variations. We are also testing the hypothesis that deriving type embeddings from entity embeddings yields higher quality typing than treating types as part of a knowledge graph and jointly deriving entity and type embeddings. We are also looking to carry out a broader user study than the preliminary study in Application Experiment 2.

REFERENCES

[1] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[2] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks." in *EMNLP*, 2014, pp. 740–750.

[3] L. d. Corro, A. Abujabal, R. Gemulla, and G. Weikum, "Finet: Context-aware fine-grained named entity typing," in *Proceedings of the Conference on Empirical Methods on Natural Language Processing*. Assoc. for Computational Linguistics, 2015.

[4] G. Durrett and D. Klein, "A joint model for entity analysis: Coreference, typing, and linking," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 477–490, 2014.

[5] M. v. Erp and P. Vossen, "Entity typing using distributional semantics and dbpedia," in *Under Review*, 2016.

[6] A. Gangemi, A. Nuzzolese, V. Presutti, F. Draicchio, A. Musetti, and P. Ciancarini, "Automatic typing of dbpedia entities," *The Semantic Web–ISWC 2012*, pp. 65–81, 2012.

[7] P. Groth, S. Pal, D. McBeath, B. Allen, and R. Daniel, "Applying universal schemas for domain specific ontology expansion," *Proceedings of AKBC*, pp. 81–85, 2016.

[8] S. Guo, Q. Wang, B. Wang, L. Wang, and L. Guo, "Semantically smooth knowledge graph embedding." in *ACL (1)*, 2015, pp. 84–94.

[9] M. Kejriwal and P. Szekely, "Supervised typing of big graphs using semantic embeddings," *arXiv preprint arXiv:1703.07805*, 2017.

[10] S. Li, J. Zhu, and C. Miao, "Psdvec: A toolbox for incremental and scalable word embedding," *Neurocomputing*, 2016.

[11] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion." in *AAAI*, 2015, pp. 2181–2187.

[12] U. Lösch, S. Bloehdorn, and A. Rettinger, "Graph kernels for rdf data," in *Extended Semantic Web Conference*. Springer, 2012, pp. 134–148.

[13] Y. Ma, T. Tran, and V. Bicer, "Typifier: Inferring the type semantics of structured data," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 206–217.

[14] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[15] A. Melo, H. Paulheim, and J. Völker, "Type prediction in rdf knowledge bases using hierarchical multilabel classification," in *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, 2016.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[17] H. Paulheim and C. Bizer, "Type inference on noisy rdf data," in *International Semantic Web Conference*. Springer, 2013, pp. 510–525.

[18] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–43.

[19] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[20] C. E. Rasmussen, "The infinite gaussian mixture model," in *Advances in neural information processing systems*, 2000, pp. 554–560.

[21] X. Ren, A. El-Kishky, C. Wang, F. Tao, C. R. Voss, and J. Han, "Clustype: Effective entity recognition and typing by relation phrase-based clustering," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 995–1004.

[22] K. Riesen and H. Bunke, *Graph classification and clustering based on vector space embedding*. World Scientific Publishing Co., Inc., 2010.

[23] P. Ristoski and H. Paulheim, "Rdf2vec: Rdf graph embeddings for data mining," in *International Semantic Web Conference*. Springer, 2016, pp. 498–514.

[24] J. Rosati, P. Ristoski, T. Di Noia, R. d. Leone, and H. Paulheim, "Rdf graph embeddings for content-based recommender systems," in *CEUR workshop proceedings*, vol. 1673. RWTH, 2016, pp. 23–30.

[25] M. Sahlgren, "The distributional hypothesis," *Italian Journal of Linguistics*, vol. 20, no. 1, pp. 33–54, 2008.

[26] J. Turian, L. Ratinov, and Y. Bengio, "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2010, pp. 384–394.

[27] Q. Wang, B. Wang, and L. Guo, "Knowledge base completion using embeddings and rules." in *IJCAI*, 2015, pp. 1859–1866.

[28] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes." in *AAAI*. Citeseer, 2014, pp. 1112–1119.

[29] Y. Yaghoobzadeh and H. Schütze, "Corpus-level fine-grained entity typing using contextual information," *arXiv preprint arXiv:1606.07901*, 2016.

AUTHOR BIOGRAPHIES

Mayank Kejriwal is a research scientist at the Information Sciences Institute, USC Viterbi School of Engineering, and is the corresponding author. He graduated in 2016 with his PhD from the University of Texas at Austin. He has been actively involved in researching, testing and integrating machine learning and Semantic Web applications in the Domain-specific Insight Graph (DIG) architecture, most notably entity resolution, information extraction, and entity-centric information retrieval. DIG is widely used by US law enforcement agencies to combat human trafficking. He is currently funded under 3 DARPA projects, and is co-authoring a textbook on knowledge graphs with Pedro Szekely and Craig Knoblock.

Pedro Szekely is a Research Associate Professor at ISI/USC. He received his PhD from Carnegie Mellon University and is currently the principal investigator on the DARPA MEMEX program. His research expertise is in rapid and robust construction of domain-specific knowledge graphs, for which he has won multiple best paper awards from the Semantic Web community.