Author's version

Using Crowdsourcing for Fine-Grained Entity Type Completion in Knowledge Bases*

Zhaoan Dong[1], Ju Fan[1], Jiaheng Lu[2], Xiaoyong Du[1], and Tok Wang Ling[3]

[1] DEKE, MOE and School of Information, Renmin University of China
fanj@ruc.edu.cn
[2] Department of Computer Science, University of Helsinki, Finland
[3] School of Computing, National University of Singapore, Singapore

Abstract. Recent years have witnessed the proliferation of large-scale Knowledge Bases (KBs). However, many entities in KBs have *incomplete* type information, and some are totally untyped. Even worse, fine-grained types (e.g., `BasketballPlayer`) containing rich semantic meanings are more likely to be incomplete, as they are more difficult to be obtained. Existing machine-based algorithms use predicates (e.g., `birthPlace`) of entities to infer their missing types, and they have limitations that the predicates may be insufficient to infer fine-grained types. In this paper, we utilize crowdsourcing to solve the problem, and address the challenge of controlling crowdsourcing cost. To this end, we propose a hybrid machine-crowdsourcing approach for fine-grained entity type completion. It firstly determines the types of some "*representative*" entities via crowdsourcing and then infers the types for remaining entities based on the crowdsourcing results. To support this approach, we first propose an embedding-based influence for type inference which considers not only the distance between entity embeddings but also the distances between entity and type embeddings. Second, we propose a new difficulty model for entity selection which can better capture the uncertainty of the machine algorithm when identifying the entity types. We demonstrate the effectiveness of our approach through experiments on real crowdsourcing platforms. The results show that our method outperforms the state-of-the-art algorithms by improving the effectiveness of fine-grained type completion at affordable crowdsourcing cost.

Keywords: Crowdsourcing · Entity Type Completion · Knowledge Base

1 Introduction

The last decades have witnessed the booming of large-scale and open-accessible Knowledge Bases (KBs) such as DBpedia [10], Freebase [2], and YAGO [20]. These KBs contain thousands of millions of real-world *entities* that fall into different *types*, e.g., `Person`, `Place`, `Sport`. Due to their large coverage and high quality, the KBs have been successfully used to support many applications, such

* This work is supported by NSFC (No.61602488, No.61632016 and No.61472427).

as web search, question answering and entity linking. However, many entities in the KBs have *incomplete* type information, and some are even totally untyped [17]. In DBPedia, for example, there are over 4 million entities assigned with about 4 million types, which means that only one type per entity in average [8]. Even worse, many entities in DBPedia only have *coarse-grained* types such as `Thing`, while *fine-grained* types such as `GolfPlayer` are missing since they are difficult to be obtained when constructing the KBs. This incompleteness of entity types affects the usefulness and usability of the KBs. For example, entity `Hedy_Lamarr`[4] in DBpedia is assigned with only general types including `Thing`, `Agent` and `Person` but none of the fine-grained ones: neither `Actor` nor `Inventor`. Therefore, `Hedy_Lamarr` is bound to be missed in the answer of following question: *Who is not only a famous* ***Actor****, but also an* ***Inventor****?*

To complete the type information in KBs, some *machine-based* approaches have been proposed. For example, *SDType*, which is reported as the state-of-the-art method [13, 18], exploits the predicates, i.e., links between entities, to infer missing types. Intuitively, if a predicate occurs in entities of one specific type, it would be assigned with a large weight. In contrast, if it occurs in entities of many different types, it will be assigned with a low weight. Obviously, high weighted predicates are more likely to identify fine-grained types. For example, in DBPedia, predicates `teachingStaff` and `numberOfClassrooms` have a high weight to infer type `School`. *SDType* computes a confidence score for every possible type of an entity based on predicate weights. Then, if confidence of a type is larger than some threshold, say 0.5, it completes the type for the entity.

However, since the KBs are often incomplete and noisy, it is difficult for *SDType* to infer the correct *fine-grained* types if an entity misses the correct highly weighted predicates or has some wrong predicates. For example, in DBPedia with version 3.8, the `BasketballPlayer` entity `Ron_Harper` has only 4 predicates, `college`, `draftTeam`, `birthPlace` and `nationality`. Given the entity denoted by x, *SDType* computes a score of "*inference ability*" for each p of these predicates to infer type C, which is denoted as $Prob(C(x)|p)$. For example, as shown in Table 1, $Prob$`(BasketballPlayer(x)|draftTeam)`$=0.281$ means 28.1% entities having predicate `draftTeam` belong to type `BasketballPlayer`. We can see that the scores of the four predicates of `Ron_Harper` are quite small, which is insufficient to infer type `BasketballPlayer` for the entity. A detailed analysis on computing $Prob(C(x)|p)$ and limitations of *SDType* is referred to Section 3.

To overcome the limitation of machine-based approaches, we propose to utilize crowdsourcing that leverages intelligence of the crowd to solve the entity type completion problem. The main motivation is that human is much better than machine to identify entity types, even though predicates of entities may be missing. For example, Fig. 1 shows an example crowdsourcing task for `Ron_Harper`. We can see that it is not difficult for human to identify the correct type(s) for the given entities. This is also verified by our experiments that the crowdsourcing result is much better than machine-based approaches.

[4] http://dbpedia.org/page/Hedy_Lamarr

Table 1. Statistics of top 10 predicates linked to `BasketballPlayer`

	p	$Prob(C(x)\|p)$	w_p
1	draftTeam	0.281	0.078
2	highschool	0.241	0.337
3	college	0.224	0.422
4	nationality	0.035	0.005
5	ceo	0.023	0.002
6	coach	0.023	0.002
7	alumni	0.015	0.004
8	league	0.014	0.246
9	birthPlace	0.009	0.021
10	formerTeam	0.006	0.248

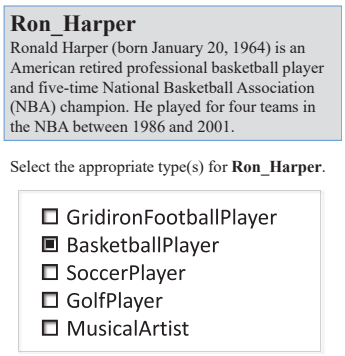


Fig. 1. A example of micro-task

Unfortunately, the challenge in utilizing crowdsourcing for entity type completion is the crowdsourcing cost, since we need to pay rewards to the crowd for their answers. Especially, in a large-scale knowledge base, it would be extremely expensive if all the entities with their possible types are crowdsourced. Therefore, we devise a hybrid framework which combines the intelligence of crowdsourcing workers with the algorithmic approaches. We firstly select some *"representative"* entities. Next, we publish the selected entities for crowdsourcing. At last, we infer and determine the entities' types based on the crowdsourcing results.

To support the hybrid framework, we develop the following techniques. First, we propose an embedding-based influence for type inference which considers not only the distance between entities but also the distance between entities and types when inferring entity types. Second, we propose a new difficulty model for entity selection which can better describe the uncertainty of the machine algorithm to determine the correct types of entities. We demonstrate the effectiveness of the method through experiments on real datasets. The results show that our hybrid method outperforms the baseline machine algorithm by recalling more fine-grained entity types with small extra crowdsourcing cost.

The remainder of this paper is organized as follows: Section 2 presents an overview of our approach. Sections 3, 4 and 5 introduce the techniques on generating candidate types, type inference and entity selection respectively. The experimental results are reported in Section 6 and related works are reviewed in Section 7. We finally conclude the paper in Section 8.

2 An overview of Entity Type Completion

This section presents an overview of entity type completion in KBs. We first formally define the problem and then introduce our crowdsourcing-based approach.

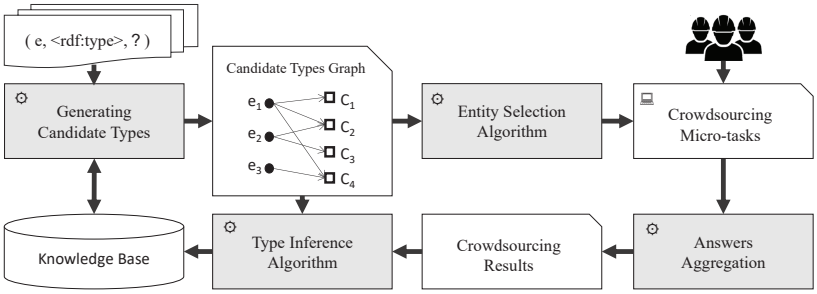


Fig. 2. An overview of the hybrid method

2.1 Problem Formalization

We denote a knowledge base $\mathcal{K} = \{\mathcal{E}, \mathcal{T}, \mathcal{R}, \mathcal{K}_\mathcal{R}, \mathcal{K}_\mathcal{T}\}$, where $\mathcal{E}$ is the set of entities (e.g., `Ron_Harper`), $\mathcal{T}$ is the set of entity types (e.g., `BasketballPlayer`) and $\mathcal{R}$ is the set of predicates (e.g., `birthPlace`). $\mathcal{K}_\mathcal{R} = \{p(e,o) \,|\, e,o \in \mathcal{E} \wedge p \in \mathcal{R}\}$ contains the known predicate instances (e.g., `birthPlace`(`Ron_Harper`,`Dayton`)). $\mathcal{K}_\mathcal{T} = \{C(e) \,|\, e \in \mathcal{E} \wedge C \in \mathcal{T}\}$ is a set of known type assertions indicating that entity e is an instance of type C (e.g., `Person`(`Ron_Harper`)). With the notations, we formulate the entity type completion problem as follows:

Definition 1 (Entity Type Completion). *Given a set of entities $\mathcal{A} \subseteq \mathcal{E}$ and a set of entity types $\mathcal{T}$, it determines whether an entity-type pair (e, C) is true or not, where $e \in \mathcal{A}$, $C \in \mathcal{T}$ and $C(e) \notin \mathcal{K}_\mathcal{T}$. Then, if (e, C) is true, we can add the new found type assertion $C(e)$ to $\mathcal{K}_\mathcal{T}$: $\mathcal{K}_\mathcal{T} \leftarrow \mathcal{K}_\mathcal{T} \cup \{C(e)\}$.*

2.2 Framework of Our Crowdsourcing-Based Approach

To address the problem, we propose a crowdsourcing-based approach that utilizes the intelligence of crowdsourcing workers. Moreover, as the crowdsourcing budget is limited, we devise a machine-crowdsourcing hybrid framework that is illustrated in Figure 2.

The approach takes as input a knowledge base $\mathcal{K}$ that contains all the known predicate instances $\mathcal{K}_\mathcal{R}$ and type assertions $\mathcal{K}_\mathcal{T}$. Given a set of entities missing types, it first employs a machine-based algorithm to generate the candidate types for them. Then, an *Entity Selection Algorithm* selects the most "*representative*" entities under a given crowdsourcing budget. On the one hand, the selected entities should be "*uncertain*" for the algorithm to identify the correct types. On the other hand, the selected entities should be more useful to infer more type assertions for unselected entities. Next, we generate micro-tasks for the selected entities and publish them to crowdsourcing platform, e.g., Amazon Mechanical Turk (AMTurk)[5], where human workers could help to identify the right types. Then, the answers collected from the crowds will be aggregated. Finally, the *Type Inference Algorithm* infers the types for all target entities based on both the crowdsourcing results and the inferred results.

[5] https://www.mturk.com

We note that there are some existing studies [5,9,12,14] that devise machine-crowdsourcing hybrid approaches in other applications, such as, web table matching and information extraction (more details can be referred to Section 7). However, we are the first to devise the hybrid framework in entity type completion. The variety of knowledge bases requires us to develop new techniques on the following components. We discuss how to generate candidate types in Section 3, introduce the inference algorithm in Section 4, and present how to select "*representative*" entities in Section 5.

3 Generating Candidate Types

We use the state-of-the-art machine-based type completion method *SDType* [18] to generate candidate types. The basic building blocks of SDType are conditional probabilities, e.g., the probability of an entity being of type C if it has a predicate p. Additionally, each predicate p is assigned a weight w_p which reflects the discriminating power of the predicate. Note that p is treated differently with its reverse predicate p^{-1}, i.e., they are assigned different weights respectively. According to the probability distribution of all predicates associated with each entity, *SDType* computes a confidence score for each entity-type pair.

$$w_p = \sum_{C \in \mathcal{T}} (Prob(C) - Prob(C|p))^2 \tag{1}$$

$$score(e_i, C_j) = \frac{\sum_{p \in Pred(e_i)} Prob(C_j(e_i)|p) \cdot w_p}{\sum_{p \in Pred(e_i)} w_p} \tag{2}$$

where $Prob(C_j(e_i)|p)$ indicates how likely an entity e_i having predicate p is of class C_j, and $Prob(C|p)$ indicates the prior probability of type C, i.e, how many entities that belongs to the type C with predicate p.

Limitations of *SDType*. We utilize the example in Table 1 to analyze limitations of *SDType*. In DBPedia, the `BasketballPlayer` entity `Ron_Harper` has only 4 predicates, `college`, `draftTeam`, `birthPlace` and `nationality`. All of them appear in the top-10 predicates linked with type `BasketballPlayer` respect to the conditional probability. As is shown in Table 1, $Prob(C(x)|p)$ indicates how likely an entity having predicate p is of class C, and w_p is the weight of predicate p. For example, `Prob(BasketballPlayer(x)|draftTeam)`=0.281 means 28.1% entities having predicate `draftTeam` belong to `BasketballPlayer`. Although the predicate `college` has the maximum weight as 0.422, it is hard to determine the correct type `BasketballPlayer` because the conditional probability is just 0.224. As a result, the confidence value of the entity-type pair <`Ron_Harper`, `BasketballPlayer`>is just 0.233. In fact, if the confidence threshold is set to 0.5, Table 1 indicates that none of the `BasketballPlayer` entities could obtain the correct type via *SDType* because no predicate has a conditional probability greater than 0.5.

Candidate Types Graph. The target entities and their candidate types are represented in a Candidate Types Graph. A *Candidate Types Graph* is a bipartite

Algorithm 1 *Type Inference Algorithm*

Input
$\mathcal{G} = \{\mathcal{A}, \mathcal{T}^*, \Pi\}$*: a initial Candidate Types Graph.*
$\mathcal{S}^q = \{< e^q, C >\}$*: the approved entity-type pairs for selected entities* $\mathcal{A}^q$*.*
Output
$\mathcal{G}' = \{\mathcal{A}, \mathcal{T}^*, \Pi'\}$*: the updated Candidate Types Graph.*
1: Initialize $\Pi' = \Phi$
2: **for** *each* $\pi =< e, C, s > \in \Pi$ **do**
3: **if** $e \in \mathcal{A}^q$ **then**
4: Update the score of π with crowdsourcing answer, i.e., $s' = 1.0$
5: **else**
6: Update the score of π: $s' = \alpha \cdot s + (1 - \alpha) \cdot Inf(\pi|\mathcal{A}^q)$
7: **end if**
8: $\Pi' \leftarrow \Pi' \cup \{< e, C, s' >\}$
9: **end for**
10: *return* $\mathcal{G}' = \{\mathcal{A}, \mathcal{T}^*, \Pi'\}$

graph $\mathcal{G} = \{\mathcal{A}, \mathcal{T}^*, \Pi\}$, where $\mathcal{A}$ is the set of target entities, $\mathcal{T}^*$ is the set of candidate types and $\Pi = \{< e_i, C_j, s_{i,j} > \mid e_i \in \mathcal{A} \wedge C_j \in \mathcal{T}^*\}$ is the set of all possible entity-type pairs. And $\pi_{i,j} =< e_i, C_j, s_{i,j} >$ indicates the probability that the type assertion $C_j(e_i)$ holds is $s_{i,j}$, where $1 \leq i \leq |\mathcal{A}|$ and $1 \leq j \leq |\mathcal{T}^*|$. We also represent all candidate types of entity e as $\mathcal{C}(e)$. For example, Fig. 3 is a toy Candidate Types Graph that consists of 3 entities, 4 candidate types and 7 candidate entity type pairs.

4 Inferring Types Using Crowdsourcing Results

4.1 Type Inference Algorithm

We adopt the concept determination algorithm proposed in [5] as our type inference algorithm. As shown in Algorithm 1, it takes the Candidate Types Graph $\mathcal{G}$ and the crowdsourcing validated entity-type pairs $\mathcal{S}^q$ as input. For each entity-type pair, the algorithm updates its score based on two evidences. One is the initial score from *SDType* whose prior probability is denoted as α. The other is the influences from the approved entity-type pairs by crowdsourcing with prior probabilities $1 - \alpha$. Finally, the algorithm outputs updated Candidate Types Graph $\mathcal{G}'$ with the revised confidence scores for each entity-type pairs. The major difference is that we propose a new method to compute the influences $Inf(\pi|\mathcal{A}^q)$ in line 6 of Algorithm 1.

4.2 Influence between Entity-Type Pairs

In [5], the authors proposed a concept-based method to compute the *inter-table influence* between two columns as the cosine similarity of their concept vectors. The same idea can be adopted for entity type completion tasks, that is, similar entities are more likely to belong to the same type.

Type Vector Based Influence: Specifically, we generate a type vector for each entity e based on Candidate Types Graph, which is denoted as $\overrightarrow{e}$. Each dimension in the vector represents a candidate type and the value is the confidence score $s_{i,j}$ of the edge $<e_i, C_j>$. Therefore, the influence from an entity-type pair $< e_m, C >$ to $< e_n, C >$ can be computed as the cosine similarity of two entities' type vectors, i.e., $\overrightarrow{e_m}$ and $\overrightarrow{e_n}$.

$$Inf(< e_n, C > | < e_m, C >) = CosineSimilarity(\overrightarrow{e_m}, \overrightarrow{e_n}) \tag{3}$$

Embedding-based Influence: Unlike the type vectors which are constructed based on the edges in Candidate Types Graph, embedding is a latent representation for knowledge bases. Embedding-based algorithms, such as TransE [3], embed entities and relations into relatively low dimensional representations (i.e., embeddings) so that semantic related entities can be close to each other. For example, entities of the same type are usually close to each other in the embeddings space. Additionally, entity types can also be embedded into the same space, so that one type embedding can be close to the entities from that type [8]. Therefore, we propose an embedding-based method to compute the influences between entity-type pairs, e.g., from an entity-type pair $< e_m, C >$ to $< e_n, C >$.

$$Inf(< e_n, C > | < e_m, C >) = \begin{cases} \frac{1}{1+\|\overrightarrow{e_m}, \overrightarrow{e_n}\|_2} & if \|\overrightarrow{e_m}, \overrightarrow{C}\|_2 \geq \|\overrightarrow{e_n}, \overrightarrow{C}\|_2 \\ 0 & if \|\overrightarrow{e_m}, \overrightarrow{C}\|_2 < \|\overrightarrow{e_n}, \overrightarrow{C}\|_2 \end{cases} \tag{4}$$

where $\overrightarrow{e_m}$, $\overrightarrow{e_n}$ denote the embeddings of e_m and e_n respectively, and $\overrightarrow{C}$ is the type embedding of C in the same embedding space. We use 2-norm to measure the distance between entity and type embedding. Unlike type vector based influence which only considers the similarity between two entities, embedding based influence not only considers the distance between embeddings of two entities but also the distances from type embedding to each entity.

Specifically, there are two meanings in Equation 4. On the one hand, if the type of an entity is already determined to be one type, we can infer the type of entity closer to the same type. For example, in Fig. 4, if a belongs to type T, $Inf(< b, T > | < a, T >) = 1/(1 + d_3)$, which depends on the distance between a and b. So that, $Inf(< b, T > | < a, T >) > Inf(< x, T > | < a, T >)$ because b is closer to a than x. On the other hand, we cannot infer the type of entity far away from the type. For example, in Fig. 4, we cannot infer the type of a based on b, i.e., $Inf(< a, T > | < b, T >) = 0$.

4.3 Aggregating the Influences

We use the same method used in [5] to aggregate the influences from the selected entities, where $\xi(e_m)$ denotes all the candidate type assertions of entity e_m and $\xi(\mathcal{A}^q)$ represents the approved entity-type pairs of selected entities $\mathcal{A}^q$. Firstly, we assume that the influences from those edges in $\xi(e_m)$ to $\pi_{n,j}$ are independent with each other. Therefore, the influence from an entity to an entity-type pair

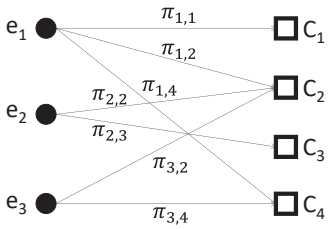


Fig. 3. An example of Candidate Types Graph

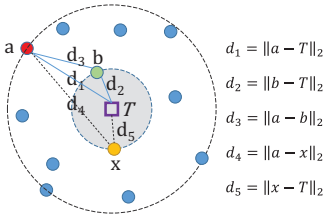


Fig. 4. Embedding based influence

is aggregated as:

$$Inf(\pi_{n,j}|e_m) = 1 - \prod_{\pi_{m,i} \in \xi(e_m)} (1 - Inf(\pi_{n,j}|\pi_{m,i})) \tag{5}$$

which could be interpreted as the probability that $\pi_{n,j}$ is influenced by at least one edge of $\xi(e_m)$. Then we compute the influence from selected entities to an entity-type pair as:

$$Inf(\pi_{n,j}|\mathcal{A}^q) = 1 - \prod_{\pi_{m,i} \in \xi(\mathcal{A}^q)} (1 - Inf(\pi_{n,j}|\pi_{m,i})) \tag{6}$$

Similarly, the influence from an entity e_m to e_n indicates the probability that at least one edge in $\xi(e_n)$ is influenced by entity e_m.

$$Inf(e_n|e_m) = 1 - \prod_{\pi_{n,j} \in \xi(e_n)} (1 - Inf(\pi_{n,j}|e_m)) \tag{7}$$

Finally, we obtain the influence from selected entities $\mathcal{A}^q$ to an entity e_n as:

$$Inf(e_n|\mathcal{A}^q) = 1 - \prod_{\pi_{n,j} \in \xi(e_n)} (1 - Inf(\pi_{n,j}|\mathcal{A}^q)) \tag{8}$$

5 Selecting Entities for Crowdsourcing

The fundamental challenge in the hybrid approach is to determine which entities should be selected for crowdsourcing. In [5], the authors proposed an expected utility function which considers both task difficulty and influence. They developed a greedy-based algorithm based on the expected utilities. Similarly, for entity type completion tasks, we define the expected utility of the selected entities $\mathcal{A}^q$ as:

$$E[(\mathcal{A}^q)] = \sum_{e \in \mathcal{A}} \mathcal{D}(e) \cdot Inf(e|\mathcal{A}^q) \tag{9}$$

We modify the greedy algorithm proposed in [5] and apply it to select entities for entity type completion. The major difference is that we use our new difficulty model and embedding based influence model when computing the utilities of the selected entities. Thus, we mainly introduce our new difficulty model below.

5.1 Entity Difficulty Model

The entity difficulty model illustrates how certain the machine algorithm is when identifying the types for one entity. For example, the entropy of types distribution is a common measure of uncertainty which is already used in [5]. The intuition is that, the more uncertain the machine algorithm is, the more likely it will be to make mistakes. Therefore, we need crowdsourcing to complete the types of those entities which are most uncertain for machine algorithm.

For *SDType*, we find it is more uncertain because: (1) the probabilities of an entity's candidate types are almost identical; (2) the weights of related predicates are very low; (3) the maximum of scores is very low; (4) the entity has too many candidate types. Based on the observations, we propose a new difficulty model which takes all of the above factors into consideration.

Entropy. Similar to [5], we firstly consider the entropy which reflects the distribution of the probabilities of an entity's candidate types. On the one hand, if a type has clearly higher score than others, the entropy is low. On the other hand, if it is close to a uniform distribution, the entropy is high.

$$\mathcal{D}_1(e) = -\sum_{C \in \mathcal{C}(e)} \frac{score(e,C)}{S} \cdot \log \frac{score(e,C)}{S} \tag{10}$$

where $S = \sum_{C \in \mathcal{C}(e)} score(e,C)$ is used for normalization.

Average Weight of Predicates. *SDType* uses all predicates linked with the entity as indicators for its types. If each predicate in $Pred(e)$ has a large weight w_p, i.e., each of them has a great discriminating power, then it is easy to identify correct types for entity e. Otherwise, it is difficult.

$$\mathcal{D}_2(e) = \frac{1}{|Pred(e)|} \cdot \sum_{p \in Pred(e)} w_p \tag{11}$$

Max Score of Candidate Types. The intuition is that, if one type has significant higher score than that of other types, it is easy to determine the answer, otherwise, difficult.

$$\mathcal{D}_3(e) = 1 - \max\{score(e,C) | t \in \mathcal{C}(e)\} \tag{12}$$

Number of Candidate Types. Obviously, it is difficult to infer the correct type if there are too many candidate types. Thus,

$$\mathcal{D}_4(e) = |\mathcal{C}(e)| \tag{13}$$

Finally, the difficulty of an entity e is defined as follows:

$$\mathcal{D}(e) = \mathcal{D}_1(e) \times \mathcal{D}_2(e) \times \mathcal{D}_3(e) \times \mathcal{D}_4(e) \tag{14}$$

6 Experiments

We implement the algorithms using `Scala 2.11` including *SDType*, the *Entity Selection Algorithm* and *Type Inference Algorithm*. The codes run in a pseudo-distributed `spark-2.0.0` on a single PC with a 2.6GHz Intel core i5 processor and 16GB RAM.

6.1 Datasets

We firstly extract $9,970,687$ predicate instances of 629 predicates about $2,283,173$ entities from DBPedia3.8 [6]. For types, we extract $7,727,665$ type assertions and transform the hierarchical structure in DBpedia Ontology[7] into a tree structure[8] where the root type ($Level = 0$) is "`Thing`". It should be mentioned that, in this paper, we mainly focus on the fine-grained entity type completion. In particular, we evaluate completion for types with $Level = 4$.

DBP-904: We randomly extract about 1000 type assertions, i.e., $<e, C>$pairs from $7,727,665$ types where the type C is from $Level = 4$ and has at least 100 instances. Then we filter out some entities according to the pruning strategy for candidate types (see Section 6.2). At last, 904 entities with their fine-grained types are retained.

DBP-4987: We first extract predicates having at least 1000 instances and their subjects and objects must have types of $Level = 4$. For each extracted predicate, we sample 50 instances, then we obtain 5324 entities. We also filter out those entities whose ground truth does not appear in the candidate type list. Finally, we obtain 4987 entities with their type assertions. There are 7054 predicate instances of 107 predicates in total.

Embeddings of Entities and Types: We generate entity embeddings using Fast-TransE[9] which is an efficient implementation of TransE. We run the code with the parameters: *nepoches*=5000, *nbatches*=100, *dimension*=250, *alpha*=0.001, and *threads*=8. The remaining parameters have the default values as presented in the source code. Based on the entity embeddings, we use the algorithm proposed in [8] to generate type embeddings in the same semantic vector space.

6.2 Crowdsourcing on Amazon Mechanical Turk

Pruning Candidate Types: In *SDType*, the number of candidate types for different entities is quite different. For example, some may have more than 100 candidate types since one predicate often provides many candidate types. Unfortunately, this will affect the quality of crowdsourcing because human workers are easy to be bored with such a long list.

[6] http://wiki.dbpedia.org/services-resources/datasets/previous-releases/dataset-38
[7] http://wiki.dbpedia.org/services-resources/ontology
[8] http://mappings.dbpedia.org/server/ontology/classes/
[9] https://github.com/thunlp/Fast-TransX

To tackle this problem, we set a threshold η for $Prob(C(e)|p)$ to prune the entity-type pairs with low probabilities. For example, when $\eta = 0$, we get 92 candidate types for entity `Hirooka_Station`. When $\eta = 0.1$, only 3 of them are retained. Thus, it is easy for human workers to select the right types. In our experiments, we empirically set $\eta = 0.05$ which can keep the number of candidate types within an acceptable range. After being pruned, the average number of the candidate types is 6 and the maximum is 15.

Micro-Tasks and Answer Aggregation: We generate micro-tasks for selected entities and publish them on Amazon Mechanical Turk (AMTurk). In order to reduce the crowdsourcing cost, a Human Intelligence Task (HIT) is designed to contain 10 micro-tasks. In our experiments, each HIT is assigned to 5 workers and we spend \$0.1 for each assignment. As is shown in Fig. 1, each micro-task contains a short description of an entity and a list of candidate types, human workers are asked to select correct types from the list. For crowdsourcing answers aggregration, we employ the codes of *Get-Another-Label* algorithm[10], which is a variation of *Expectation-Maximization* (EM) [4].

6.3 Evaluation on Entity Difficulty Model

We evaluate our entity difficulty model on DBP-904. Firstly, all testing entities are sorted according to their difficulty in ascending order and then equally separated into 10 buckets. Fig. 5 shows that pure crowdsourcing method obtains high and stable recall of entity-type pairs, while the *SDType* algorithm performs worse on two datasets when the difficulty increases. This shows that the proposed entity difficulty model can effectively capture the uncertainty of entities.

6.4 The Prior Probability α

To set an appropriate prior probability α in Algorithm 1, we perform an experiment on DBP-4987. We set the α from 0.0 to 1.0 with step 0.1 and examine the F-Measure with different amount of randomly selected entities, because F-Measure considers both the Recall and Precision of the algorithm. On one hand, we hope to recall more positive entity type pairs. On the other hand, we want the false positives to be as few as possible. As shown in Fig. 6, we find that $\alpha = 0.8$ is the best for type vector based inference while $\alpha = 0.7$ for embedding based method.

6.5 Comparison of Influence Models

To evaluate the embedding based influence, we first randomly select $x\%$ of entities and publish them on AMTurk for crowdsourcing. Then, we infer the types of all target entities using the following three methods respectively: (1). *No-Influence* does not consider the influences, i.e., it directly merges the crowdsourcing results for selected entities and that of *SDType* for unselected. (2).

[10] https://github.com/ipeirotis/Get-Another-Label/wiki

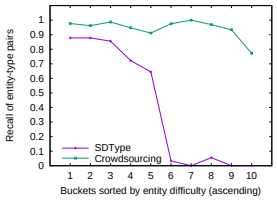
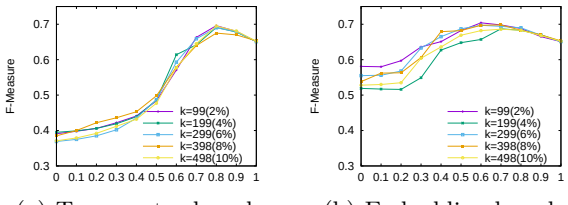
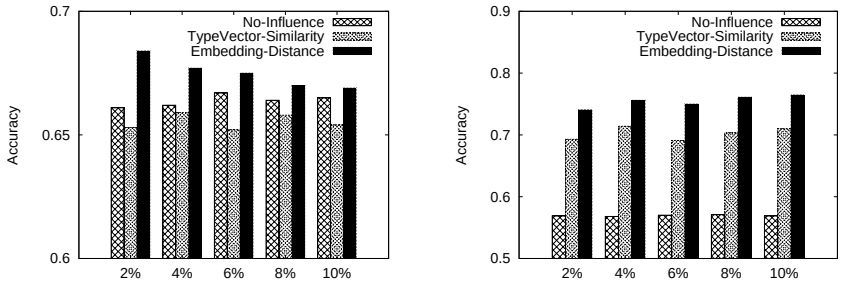


Fig. 5. Evaluation on the entity difficulty model.



(a) Type vector based (b) Embedding based

Fig. 6. Determining the prior probability α.



(a) DBP-904 (b) DBP-4987

Fig. 7. Comparison of different influence models.

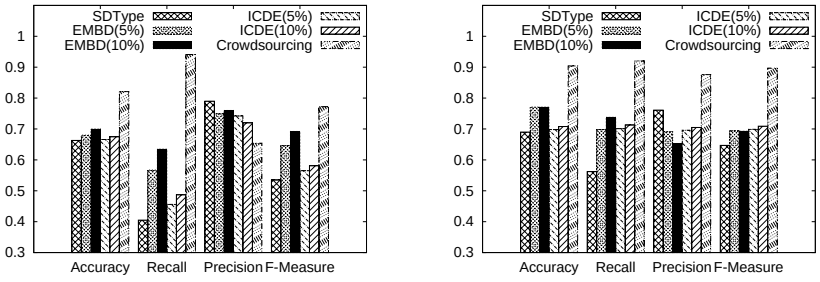
TypeVector-Similarity: Inferring with the type vector based influence. (3). *Embedding-Distance*: Inferring with the embeddings based influence.

As shown in Fig. 7, *Embedding-Distance* method outperforms the other two methods in Accuracy. The increasement is significant on DBP-4987. Compared with *No-Influence*, it is increased by 30% in average across various number of selected entities and about 8% compared with *TypeVector-Similarity*. This improvement is mainly attributed to the embedding-based representation learned, which can better capture global structural information in the KBs.

6.6 Comparison with the Existing Methods

In this section, we compare our hybrid method with three existing methods. Firstly, we briefly describe the methods to be compared as follows. (1) *SDType*: the state-of-the-art pure machine algorithm [13,18]. (2) *ICDE*: the hybrid method proposed in [5] for web table matching. We implement this algorithm for entity type completion task. In comparison, we examine the performance separately when 5% and 10% entities are selected. (3) *EMBD*: our hybrid method for entity type completion based on *embedding based influence*. Similarly, we examine the performance with two budgets 5% and 10%, separately. (4) *Crowdsourcing*: the pure crowdsourcing method, i.e., all the target entities are crowdsourced.

As shown in Figure 8, the pure crowdsourcing method achieves the best performance. For example, the values of Recall are greater than 0.9 on both DBP-904 and DBP-4987. However, it is too expensive to crowdsource all the



(a) Performance on DBP-904 (b) Performance on DBP-4987

Fig. 8. Comparison of different inference methods.

entities. The only special case is that on DBP-904, its Precision is lower than others. This is mainly because we allow human workers to select multiple types for one entity. As a result, some false positive entity-type pairs occur in the crowdsourcing results.

On DBP-904, EMBD significantly outperforms *SDType* and ICDE. For example, compared with *SDType*, EMBD(5%) increases about 40% (from 0.40 to 0.57) on the Recall and about 21% (from 0.54 to 0.65) on F-Measure. Compared with ICDE(5%), EMBD(5%) increases about 24% (from 0.46 to 0.57) on the Recall and about 14% (from 0.57 to 0.65) on F-Measure. On DBP-4987, although the advantage of EMBD is not so significant on F-Measure, it still outperforms *SDType* on Accuracy and Recall. For example, EMBD(5%) increases about 12% (from 0.69 to 0.77) on the Accuracy and about 24% (from 0.56 to 0.70) on Recall. In a word, EMBD outperforms *SDType* and ICDE on the whole by recalling more fine-grained entity type assertions with affordable crowdsourcing cost.

7 Related work

Entity Type Completion is an important subproblem of knowledge base completion. Paulheim et.al [17] classified the related methods into two categories: *internal methods* and *external methods*.

Internal methods only use the data in current knowledge base as input. A straightforward internal method is classical *ontology reasoning*. However, *RDFS reasoning* is prone to propagate errors since most knowledge bases are usually incomplete and noisy [18]. Paulheim et.al found that types can be inferred via the links between entities. For example, an entity is likely to be a `Movie`, if it has a predicate `director`. Based on the observation, they propose *SDType* [18, 19]. *SDType* performs like a weighted voting, where each predicate casts a vote on its object's type, using the statistical distributions to weight its votes.

External methods take advantage of sources outside the knowledge bases, e.g., text corpora or links to other knowledge bases. For example, Nuzzolese et al. exploit the Wikipedia links to predict entity types [16]. Tipalo system [6] parses the abstracts of entities which often follow similar patterns and map them to the WordNet and DOLCE ontologies to find types. Aprosio et al. [1]

exploit cross-language links between different language versions of DBpedia as features for type completion. Sleeman et.al [21] use SVM to predict entity types in DBpedia and Freebase. Crowdsourcing can be viewed as an external method because it makes use of human knowledge which is a kind of data source beyond the current knowledge base.

Hybrid Machine-Crwodsourcing Methods have attracted many attentions in recent years. For example, Kamar [7] studied how to combine the complementary strengths of humans and machine for solving consensus crowdsourcing tasks. Lofi et al. [12] extensively investigated the commonly reoccurring challenges and solutions for hybrid algorithmic-crowdsourcing workflows and propose a set of design patterns. Kondreddi [9] proposed Higgins, a novel system architecture that effectively integrates combines Human Computing (HC) inputs with machine based Information Extraction (IE). Mozafari et al. [14] advocated integrating machine learning into crowdsourced databases. Fan et al. [5] proposed a hybrid framework which assigns the most "beneficial" column-to-concept matching tasks to human workers and then infer the best matches for the remain columns utilizing the crowdsourcing results. In this paper, we extend their algorithms with new features including an embedding-based influence model and a new entity difficulty model and apply them to entity type completion.

Knowledge Graph Embedding, in recent years, has become an active area of research for knowledge base construction and completion. One of the most successful model is TransE [3], which learns the embeddings of entities and relations in a neural-based approach. Various methods such as TransR [11], HolE [15] are also proposed. Since TransE is the most simple and popular method, it is chosen to train the embeddings.

8 Conclusion

In this paper, we have addressed the problem of *fine-grained* entity type completion in Knowledge Bases. We proposed a hybrid method integrating the intelligence of crowdsourcing and the speed of machine algorithm. To discover more type assertions with affordable crowdsourcing cost, we proposed an embedding-based influence for type inference which considers not only the distances between entities but also the distances between entities and types. And we also proposed a new difficulty model for crowdsourcing entity selection. The experimental results on two real datasets illustrated the potential of our hybrid method.

References

1. Aprosio, A.P., Giuliano, C., Lavelli, A.: Automatic expansion of dbpedia exploiting wikipedia cross-language information. In: Extended Semantic Web Conference. pp. 397–411 (2013)
2. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase:a collaboratively created graph database for structuring human knowledge. In: SIGMOD Conference. pp. 1247–1250 (2008)

3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: International Conference on Neural Information Processing Systems. pp. 2787–2795 (2013)
4. Dawid, A.P., Skene, A.M.: Maximum likelihood estimation of observer error-rates using the em algorithm. Journal of the Royal Statistical Society **28**(1), 20–28 (1979)
5. Fan, J., Lu, M., Ooi, B.C., Tan, W.C., Zhang, M.: A hybrid machine-crowdsourcing system for matching web tables. In: IEEE International Conference on Data Engineering. pp. 976–987 (2014)
6. Gangemi, A., Nuzzolese, A.G., Presutti, V., Draicchio, F., Musetti, A., Ciancarini, P.: Automatic typing of dbpedia entities. In: International Conference on the Semantic Web. pp. 65–81 (2012)
7. Kamar, E., Hacker, S., Horvitz, E.: Combining human and machine intelligence in large-scale crowdsourcing. In: AAMAS. pp. 467–474 (2012)
8. Kejriwal, M., Szekely, P.: Supervised typing of big graphs using semantic embeddings p. 3 (2017)
9. Kondreddi, S.K., Triantafillou, P., Weikum, G.: Combining information extraction and human computing for crowdsourced knowledge acquisition. In: ICDE. pp. 988–999 (2014)
10. Lehmann, J.: Dbpedia: A large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web **6**(2), 167–195 (2015)
11. Lin, Y., Liu, Z., Zhu, X., Zhu, X., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Twenty-Ninth AAAI Conference on Artificial Intelligence. pp. 2181–2187 (2015)
12. Lofi, C., Maarry, K.E.: Design patterns for hybrid algorithmic-crowdsourcing workflows. In: CBI. pp. 1–8 (2014)
13. Melo, A., Völker, J., Paulheim, H.: Type prediction in noisy rdf knowledge bases using hierarchical multilabel classification with graph and latent features. International Journal on Artificial Intelligence Tools **26**(2), 1760011 (2017)
14. Mozafari, B., Sarkar, P., Franklin, M.J., Jordan, M.I., Madden, S.: Scaling up crowd-sourcing to very large datasets: A case for active learning. Proceedings of the VLDB Endowment (PVLDB) **8**(2), 125–136 (2014)
15. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Thirtieth AAAI Conference on Artificial Intelligence. pp. 1955–1961 (2016)
16. Nuzzolese, A.G., Gangemi, A., Presutti, V., Ciancarini, P.: Type inference through the analysis of wikipedia links. In: WWW2012 Workshop on Linked Data on the Web, Lyon, France, 16 April, 2012 (2012), `http://ceur-ws.org/Vol-937/ldow2012-paper-13.pdf`
17. Paulheim, H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic web (Preprint), 1–20 (2016), survey
18. Paulheim, H., Bizer, C.: Type inference on noisy rdf data. In: International Semantic Web Conference. pp. 510–525. Springer (2013)
19. Paulheim, H., Bizer, C.: Improving the quality of linked data using statistical distributions. International Journal on Semantic Web & Information Systems **10**(2), 63–86 (2014)
20. Rebele, T., Suchanek, F., Hoffart, J., Biega, J., Kuzey, E., Weikum, G.: Yago: A multilingual knowledge base from wikipedia, wordnet, and geonames. In: International Semantic Web Conference. pp. 177–185 (2016)
21. Sleeman, J., Finin, T.: Type prediction for efficient coreference resolution in heterogeneous semantic graphs. In: IEEE Seventh International Conference on Semantic Computing. pp. 78–85 (2014)