Type Inference on Noisy RDF Data

Heiko Paulheim and Christian Bizer

University of Mannheim, Germany
Research Group Data and Web Science
{heiko,chris}@informatik.uni-mannheim.de

Abstract. Type information is very valuable in knowledge bases. However, most large open knowledge bases are incomplete with respect to type information, and, at the same time, contain noisy and incorrect data. That makes classic type inference by reasoning difficult. In this paper, we propose the heuristic link-based type inference mechanism *SDType*, which can handle noisy and incorrect data. Instead of leveraging T-box information from the schema, *SDType* takes the actual use of a schema into account and thus is also robust to misused schema elements.

Keywords: Type Inference, Noisy Data, Link-based Classification

1 Introduction

Type information plays an important role in knowledge bases. Axioms stating that an instance is of a certain type are one of the atomic building blocks of knowledge bases, stating, e.g., that *Thomas Glavinic* is an instance of type *Writer*. Many useful queries to a knowledge base use type information, e.g., *Find all* ***writers*** *from Vienna*, *Is Night Work a* ***novel*** *or a* ***short story****?*, etc.

In many knowledge bases, type information is incomplete for different reasons. For instance, in a crowd-sourced knowledge base, the problem may be simply that no one may have entered the type(s) for a certain instance. When using open world semantics, as in many semantic web knowledge bases, this is not a problem from a logic point of view – however, it drastically limits the usefulness of the knowledge base.

Cross-domain knowledge bases, unlike closed-domain knowledge bases, most often contain a large variety of types. Since it is often not feasible to manually assign types to all instances in a large knowledge base, automatic support creating type information is desirable. Furthermore, since open, crowd-sourced knowledge bases often contain noisy data, logic-based reasoning approaches are likely to multiply errors.

In this paper, we show how type information can be generated heuristically by exploiting other axioms in a knowledge base, in particular links between instances. Unlike classic reasoning approaches, we use a weighted voting approach taking many links into account, which avoids the propagation of errors from single wrong axioms.

The rest of this paper is structured as follows. Section 2 motivates our work by showing typical problems of reasoning on real-world datasets. Section 3 introduces the *SDType* approach, which is evaluated in Sect. 4 in different experimental settings. In Sect. 5, we show how *SDType* can be applied to solve a real-world problem, i.e., the completion of missing type information in DBpedia. We conclude our paper with a review of related work in Sect. 6, and a summary and an outlook on future work.

2 Problems with Type Inference on Real-world Datasets

A standard way to infer type information in the Semantic Web is the use of reasoning, e.g., standard RDFS reasoning via entailment rules [20]. To illustrate the problems that can occur with that approach, we have conducted an experiment with *DBpedia* knowledge base [2]. We have used the following subset of entailment rules:

- `?x a ?t1. ?t1 rdfs:subClassOf ?t2` entails `?x a ?t2`
- `?x ?r ?y . ?r rdfs:domain ?t` entails `?x a ?t`
- `?y ?r ?x . ?r rdfs:range ?t` entails `?x a ?t`

We have applied these three rules to the instance `dbpedia:Germany`. These rules in total induce 23 types for `dbpedia:Germany`, only three of which are correct. The list of inferred types contains, among others, the types *award*, *city*, *sports team*, *mountain*, *stadium*, *record label*, *person*, and *military conflict*.

A reasoner requires only one false statement to come to a wrong conclusion. In the example of `dbpedia:Germany`, at most 20 wrong statements are enough to make a reasoner infer 20 wrong types. However, there are more than 38,000 statements about `dbpedia:Germany`, i.e., an error rate of only 0.0005 is enough to end up with such a completely nonsensical reasoning result. In other words: even with a knowledge base that is 99.9% correct, an RDFS reasoner will not provide meaningful results. However, a correctness of 99.9% is difficult, if not impossible, to achieve with real-world datasets populated either (semi-)automatically, e.g., by information extraction from documents, or by the crowd.

In the example above, the class `Mountain` in the above is induced from a single wrong statement among the 38,000 statements about `dbpedia:Germany`, which is `dbpedia:Mže dbpedia-owl:sourceMountain dbpedia:Germany`. Likewise, the class `MilitaryConflict` is induced from a single wrong statement, i.e., `dbpedia:XII_Corps_ (United_Kingdom) dbpedia-owl:battle dbpedia:Germany`.

These problems exist because traditional reasoning is only useful if a) both the knowledge base and the schema do not contain any errors and b) the schema is only used in ways foreseen by its creator [4]. Both assumptions are not realistic for large and open knowledge bases. This shows that, although reasoning seems the straight forward approach to tackle the problem of completing missing types, it is – at least in its standard form – not applicable for large, open knowledge bases, since they are unlikely to have correct enough data for reasoning to

Table 1. Type distribution of the property `dbpedia-owl:location` in DBpedia

Type	Subject (%)	Object (%)
`owl:Thing`	100.0	88.6
`dbpedia-owl:Place`	69.8	87.6
`dbpedia-owl:PopulatedPlace`	0.0	84.7
`dbpedia-owl:ArchitecturalStructure`	50.7	0.0
`dbpedia-owl:Settlement`	0.0	50.6
`dbpedia-owl:Building`	34.0	0.0
`dbpedia-owl:Organization`	29.1	0.0
`dbpedia-owl:City`	0.0	24.2
...	...	...

produce meaningful results. What is required is an approach for inducing types which is tolerant with respect to erroneous and noisy data.

3 Approach

An RDF knowledge base consists of an A-box, i.e., the definition of instances and the relations that hold between them, and a T-box, i.e., a schema or ontology. The *SDType* approach proposed in this paper exploits links between instances to infer their types using weighted voting. Assuming that certain relations occur only with particular types, we can heuristically assume that an instance should have certain types if it is connected to other instances through certain relations. For example, from a statement like `:x dbpedia-owl:location :y`, we may conclude with a certain confidence that `:y` is a place.

3.1 Link-based Type Inference

SDType uses links between resources as indicators for types, i.e., we propose a *link-based object classification* approach [6]. The basic idea is to use each link from and to a instance as an indicator for the resource's type. For each link, we use the statistical distribution (hence the name ***SD****Type*) of types in the subject and object position of the property for predicting the instance's types.

For each property in a dataset, there is a characteristic distribution of types for both the subject and the object. For example, the property `dbpedia-owl:location` is used in 247,601 triples in DBpedia. Table 1 shows an excerpt of the distribution for that property.[1]

Based on that example distribution, we can assign types with probabilities to `:x` and `:y` when observing a triple like `:x dbpedia-owl:location :y`. Given the distribution in table 1, we could assign $P($`?x a dbpedia-owl:Place`$) = 0.698$, $P($`?y a dbpedia-owl:Place`$) = 0.876$, etc.

More formally, the basic building blocks of *SDType* are conditional properties measuring how likely a type T is, given a resource with a certain property p, expressed as $P\left(T(r)|(\exists p.\top)(r)\right)$, where p may be an incoming or an outgoing

[1] All DBpedia examples in this paper use version 3.8.

property. Furthermore, each property is assigned a certain weight w_p, which reflects its capability of predicting the type (see below). With those elements, we can compute the confidence for a resource r having a type t as

$$\text{conf}\,(T(r)) := \frac{1}{N} \cdot \sum_{\text{all properties } p \text{ of } r} P\,(T(r)|(\exists p.\top)(r)), \tag{1}$$

where N is the number of properties that connects a resource to another one. By using the average probabilities of each type, we address the problem of faulty links, since they do not contribute too much to the overall probability.

In the example with `dbpedia:Germany` used above, the class `Mountain` was inferred due to one wrong statement out of 38,000. With the above definition, that relation would only be weighted with $\frac{1}{38{,}000}$, thus, the type `Mountain` would receive a comparably small overall confidence.

By looking at the *actual* distribution of types co-occurring with a property, instead of the *defined* domains and ranges, properties which are "abused", i.e., used differently than conceived by the schema creator, do not cause any problems for *SDType*. As long as a property is used more or less consistently throughout the knowledge base, the inferences will always be consistent as well. Single inconsistent usages, just like single wrong statements, do not contribute too much to the overall result. Furthermore, when looking at the actual usage of a schema, the results can be more fine-grained than when using the schema only. For example, on the *MusicBrainz* dataset[2], `foaf:name` is always used as a property of `mo:MusicArtist`. While RDFS entailment rules could not infer any specific type from the `foaf:name` property, since it has no explicit domain defined.[3]

While using the actual distribution instead of defined domains and ranges eliminates those problems, it can induce new ones when a dataset is heavily skewed, i.e., the extensions of some classes are several orders of magnitude larger than others. This is a problem in particular with general purpose properties, such as `rdfs:label` or `owl:sameAs`, which are rather equally distributed in the overall knowledge base. If that knowledge base is heavily skewed (e.g., a database about cities and countries which contains 10,000 cities per country on average), and it contains many of such general purpose properties, there is a danger of overrating the more frequent types. Thus, we define a weight w_p for each property (note that p and p^{-1} are treated independently and are each assigned an individual weight), which measures the deviation of that property from the apriori distribution of all types:

$$w_p := \sum_{\text{all types t}} \left(P(t) - P\,(t|\exists p.\top)\right)^2 \tag{2}$$

With those types, we can refine the above definition to

$$\text{conf}\,(T(r)) := \nu \cdot \sum_{\text{all properties } p \text{ of } r} w_p \cdot P\,(T(r)|(\exists p.\top)(r)), \tag{3}$$

[2] `http://dbtune.org/musicbrainz/`

[3] The defined domain of `foaf:name` is `owl:Thing`, see `http://xmlns.com/foaf/spec/`

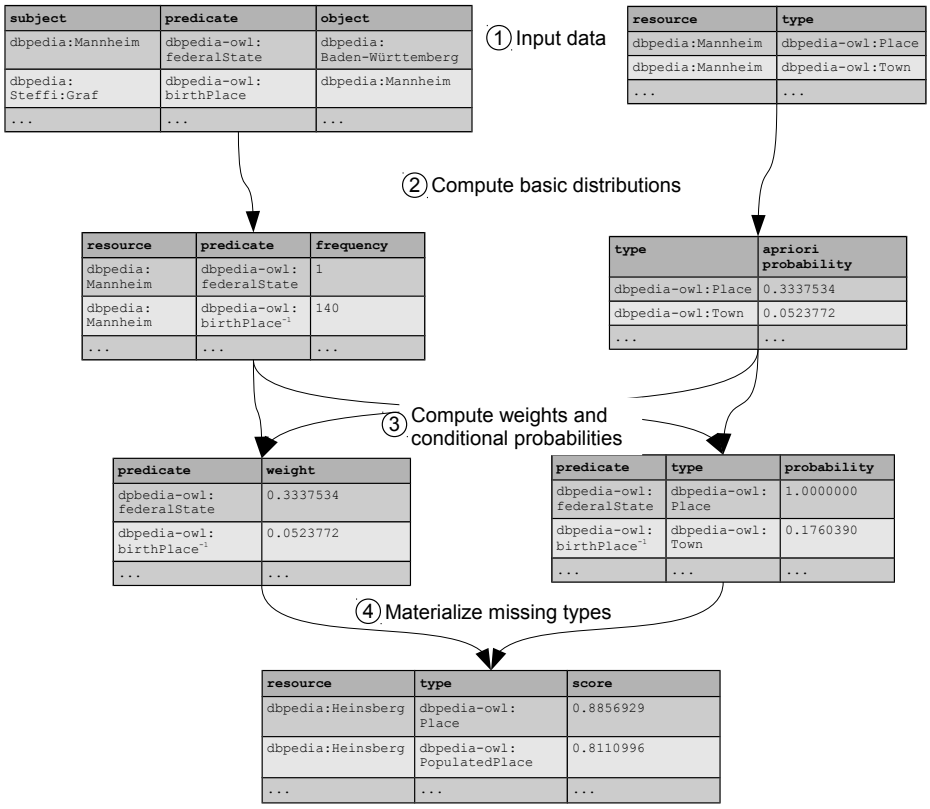

Fig. 1. Implementation of the type completion prototype as a sequence of table creation operations

with the normalization factor ν defined as

$$\nu = \frac{1}{\sum_{\text{all properties } p \text{ of } r} w_p} \tag{4}$$

Intuitively, *SDType* implements a weighted voting approach, where for each link, a vote consisting of a distribution of types is cast. The weights reflect the discriminate power of the individual links' properties.

Looking at these weights in DBpedia, for example, we can observe that the maximum weight is given to properties that only appear with one type, such as `dbpedia-owl:maximumBoatLength`, which is only used for `dbpedia-owl:Canal`. On the other end of the spectrum, there are properties such as `foaf:name`, which, in DBpedia, is used for persons, companies, cities, events, etc.

Consider, for example, the triples `:x dbpedia-owl:location :y . :x foaf:name "X"`, and an apriori probability of `dbpedia-owl:Person` and `dbpedia-owl:Place` of 0.21 and 0.16, respectively. With those numbers and distributions such

as in table 1, definition (1) would yield a confidence score for `:x a dbpedia-owl:Person` and `:x a dbpedia-owl:Place` of 0.14 and 0.60, respectively.[4]

When using weights, the numbers are different. In our example from DBpedia, the obtained weights for `dbpedia-owl:location` and `foaf:name` are 0.77 and 0.17, hence, the overall confidence scores for `:x a dbpedia-owl:Person` and `:x a dbpedia-owl:Place` in that example, using definition (3), are 0.05 and 0.78, respectively. This shows that the weights help reducing the influence of general purpose properties and thus assigning more sensible scores to the types that are found by *SDType*, and in the end help reducing wrong results coming from skewed datasets.

In summary, we are capable of computing a score for each pair of a resource and a type. Given a reasonable cutoff threshold, we can thus infer missing types at arbitrary levels of quality – thresholds between 0.4 and 0.6 typically yield statements at a precision between 0.95 and 0.99.

3.2 Implementation

SDType has been implemented based on a relational database, as shown in Fig. 1. The input data consists of two tables, one containing all direct property assertions between instances, the other containing all direct type assertions.

From these input files, basic statistics and aggregations are computed: the number of each type of relation for all resources, and the the apriori probability of all types, i.e., the percentage of instances that are of that type. Each of those tables can be computed with one pass over the input tables or their join.

The basic statistic tables serve as intermediate results for computing the weights and conditional probabilities used in the formulas above. Once again, those weights and conditional probabilities can be computed with one pass over the intermediate tables or their joins.

In a final step, new types can be materialized including the confidence scores. This can be done for all instances, or implemented as a service, which types an instance on demand. Since of each of the steps requires one pass over the database, the overall complexity is linear in the number of statements in the knowledge base.

4 Evaluation

To evaluate the validity of our approach, we use the existing type information in two large datasets, i.e., DBpedia [2] and OpenCyc [9], as a gold standard,[5] and let *SDType* reproduce that information, allowing us to evaluate recall, precision, and F-measure.

[4] The actual numbers for DBpedia are: $P(Person|foaf\#name) = 0.273941$, $P(Place|foaf\#name) = 0.314562$, $P(Person|dbpedia\#location) = 0.000236836$, $P(Place|dbpedia\#location) = 0.876949$.

[5] In the case of DBpedia, the dataset is rather a silver standard. However, it provides the possibility of a larger-scale evaluation. A finer-grained evaluation with manual validation of the results by an expert can be found in Sect. 5.

Table 2. Characteristics of the datasets used for evaluation

	DBpedia	OpenCyc
Number of instances	3,600,638	193,049
Number of distinct classes	359	119,941
Number of distinct properties	1775	18,526
Average depth of leaf classes in the class hierarchy	2.4	10.7
Average number of type statements per (typed) instance	5.6	59.9
Average number of instances per type	38,003.3	755.2
Average number of ingoing properties per instance	8.5	4.8
Average number of outgoing properties per instance	8.8	4.0

4.1 Datasets

DBpedia is generated automatically from Wikipedia infoboxes, and has a large coverage, at the price of reduced precision, e.g., due to parsing errors or mistakes in Wikipedia itself. OpenCyc, on the other hand, is more focused on precise data allowing for exact reasoning, but has a lower coverage than DBpedia. The DBpedia dataset contains all types from the infobox types dataset (i.e., DBpedia ontology, schema.org, and UMBEL).[6]

While DBpedia has all type information for the DBpedia ontology fully materialized w.r.t. `rdfs:subClassOf`, we manually materialized all direct types in OpenCyc, using simple RDFS-like inference for subclasses and subproperties (the latter are not used at all in the DBpedia ontology). Table 2 lists some relevant characteristics of the datasets.

It can be observed that the class hierarchy of OpenCyc is several orders of magnitude larger and more fine-grained than the class hierarchy of DBpedia. At the same time, the average number of instances in each class is much smaller for OpenCyc. Since the average number of properties per instance is also lower, the problem of inferring types with *SDType* on OpenCyc is harder for two reasons: there is les evidences for each instance, and the number of classes to predict is higher.

For both datasets, we have used random samples of 10,000 instances. Furthermore, we restrict our approach to using only *ingoing* properties. The reason is that classification based on outgoing properties would oversimplify the problem. In DBpedia, outgoing properties and types are generated in the same step, so the correct type can be trivially predicted from outgoing properties. The same holds for OpenCyc, which uses per class templates for populating instance data [17]. Furthermore, when trying to infer *missing* types, the instances with missing types most often have no outgoing properties.

4.2 Results

Figure 2 shows the results of *SDType* on DBpedia.[7] While it can be observed that *SDType* works sufficiently well on the overall dataset (i.e., instances that have at

[6] `http://dbpedia.org/Downloads38`

[7] The predicted types include those defined in the DBpedia ontology, schema.org, and UMBEL, as well as `owl:Thing`.

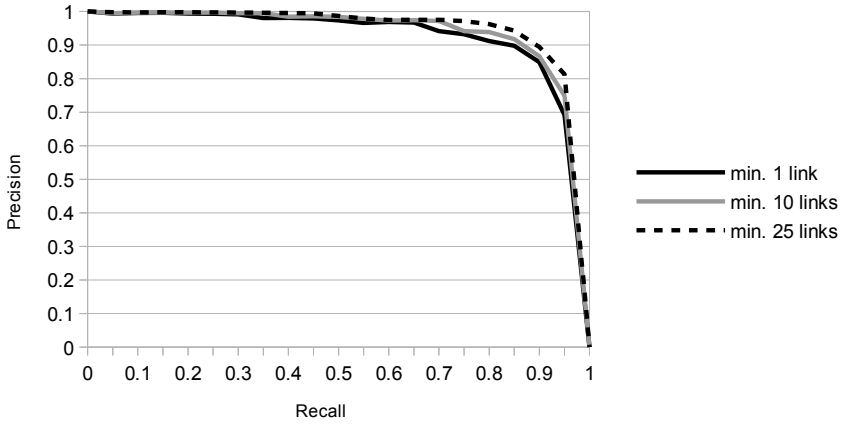

Fig. 2. Precision/recall curves of *SDType* on DBpedia, for instances with at least one, at least 10, and at least 25 incoming links

least one ingoing link), achieving an F-measure of 88.5%, the results are slightly better on instances that have at least 10 or 25 ingoing links, with an F-measure of 88.9% and 89.9%, respectively. The differences show more significantly in the precision@95% (i.e. the precision that can be achieved at 95% recall), which is 0.69 (minimum one link), 0.75 (minimum ten links), and 0.82 (minimum 25 links), respectively.

Figure 3 depicts the corresponding results for OpenCyc. The first observation is that the overall results are not as good as on DBpedia, achieving a maximum F-measure of 60.1% (60.3% and 60.4% when restricting to instances that have at least 10 or 25 ingoing links). The second observation is that the results for instances with different numbers of ingoing properties do not differ much – in fact, most of the differences are too small to be visible in the figure. While 95% recall cannot be reached on OpenCyc with *SDType*, the precision@90% is 0.18 (minimum one link), 0.23 (minimum ten and 25 links), respectively.

The strong divergence of the results between DBpedia and OpenCyc, as discussed above, was to be expected, since OpenCyc has on the one hand more (and more specific) types per instance, on the other hand less evidence per instance, since the number of properties connecting instances is smaller.

As the diagrams show, looking at instances with more links improves the results on DBpedia, but not on OpenCyc (apart from a small improvement in precision at a recall of around 0.9). The reason for that is that DBpedia, with its stronger focus on coverage than on correctness, contains more faulty statements. When more links are present, the influence of each individual statement is reduced, which allows for correcting errors. OpenCyc, on the other hand, with its stronger focus on precision, benefits less from that error correction mechanism.

Since we assume that it is more difficult to predict more specific types (such as *Heavy Metal Band*) than predicting more general ones (like *Band* or even *Organization*), we have additionally examined the best F-measure that can be

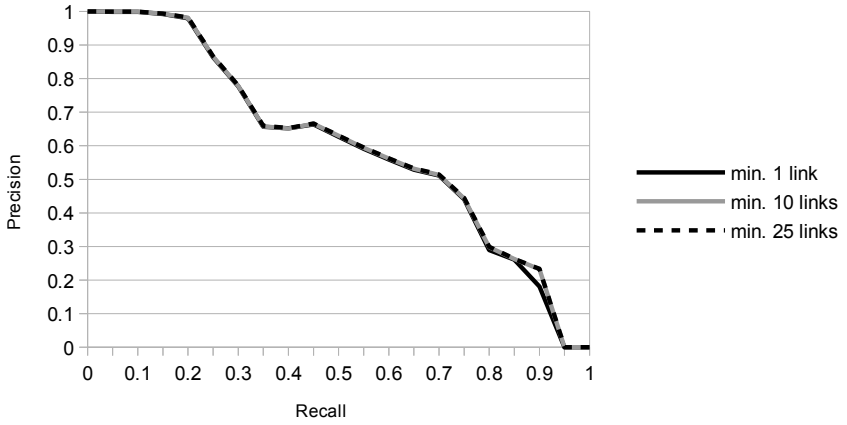

Fig. 3. Precision/recall curves of *SDType* on OpenCyc, taking into account only incoming, only outgoing, and both incoming and outgoing properties

achieved when restricting the approach to a certain maximum class hierarchy depth. The results are depicted in Fig. 4. It can be observed that *SDType* in fact works better on more general types (achieving an F-measure of up to 97.0% on DBpedia and 71.6% on OpenCyc when restricting the approach to predicting only top-level classes). However, the effects are weaker than we expected.

5 Application: Completing Missing Types in DBpedia

In the following, we apply *SDType* to infer missing type information in DBpedia. While DBpedia has a quite large coverage, there are millions of missing type statements. To infer those missing types, we have combined the approach sketched above with a preclassification step separating typeable from untypeable resources in order to reduce false inferences.

5.1 Estimating Type Completeness in DBpedia

Aside from the type information in DBpedia using the DBpedia ontology, which is generated using Wikipedia infoboxes, resources in DBpedia are also mapped to the YAGO ontology [18]. Those mappings are generated from Wikipedia page categories. Thus, they are complementary to DBpedia types – an article may have a correct infobox, but missing category information, or vice versa. Both methods of generating type information are prone to (different types of) errors. However, looking at the overlaps and differences of type statements created by both methods may provide some approximate estimates about the completeness of DBpedia types.

To estimate the completeness of type information in DBpedia, we used a partial mapping between the YAGO ontology [18] and the DBpedia ontology.[8]

[8] `http://www.netestate.de/De/Loesungen/DBpedia-YAGO-Ontology-Matching`

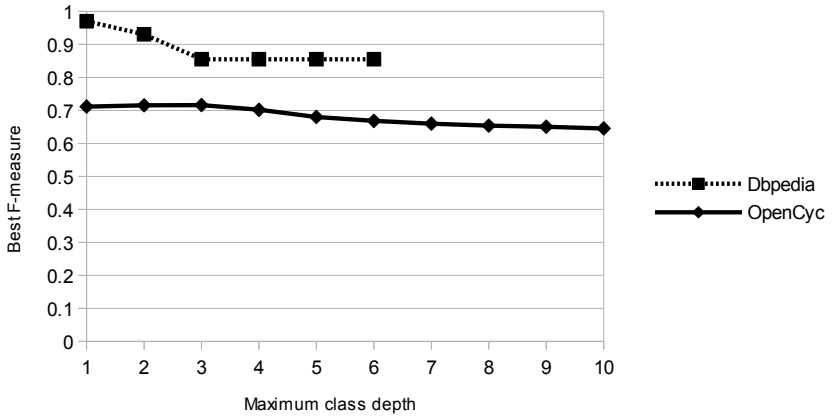

Fig. 4. Maximum achievable F-measure by maximum class depth for DBpedia and OpenCyc. The graph depicts the maximum F-measure that can be achieved when restricting the approach to finding classes of a maximum hiearchy depth of 1, 2, etc.

Assuming that the YAGO types are at least more or less correct, we can estimate the completeness of a DBpedia type `dbpedia#t` using the mapped YAGO type `yago#t` by looking at the relation of all instances of `dbpedia#t` and all instances that have at least one of the types `dbpedia#t` and `yago#t`:

$$completeness(\texttt{dbpedia\#t}) \leq \frac{|\texttt{dbpedia\#t}|}{|\texttt{dbpedia\#t} \cup \texttt{yago\#t}|} \tag{5}$$

The denominator denotes an estimate of all instances that *should* have the type `dbpedia#t`. Since the actual number of resources that should have that type can be larger than that (i.e., neither the DBpedia nor the YAGO type is set), the completeness can be smaller than the fraction, hence the inequation.

Calculating the sum across all types, we observe that DBpedia types are at most 63.7% complete, with at least 2.7 million missing type statements (while YAGO types, which can be assessed accordingly, are at most 53.3% complete). The classes the most missing type statements are shown in Fig. 5

Classes that are very incomplete include

- `dbpedia-owl:Actor` (completeness $\leq$ 4%), with 57,000 instances missing the type, including, e.g., Brad Pitt and Tom Hanks
- `dbpedia-owl:Game` (completeness $\leq$ 7%), with 17,000 instances missing the type, including Tetris and Sim City
- `dbpedia-owl:Sports` (completeness $\leq$ 5.3%), with 3,300 instances missing the type, including Beach Volleyball and Biathlon

A similar experiment using the classes `dbpedia-owl:Person` and `foaf:Person` (assuming that each person should have both types) yielded that the class `dbpedia-owl:Person` is at most 40% complete. These examples show that the problem of missing types in DBpedia is large, and that it does not only affect

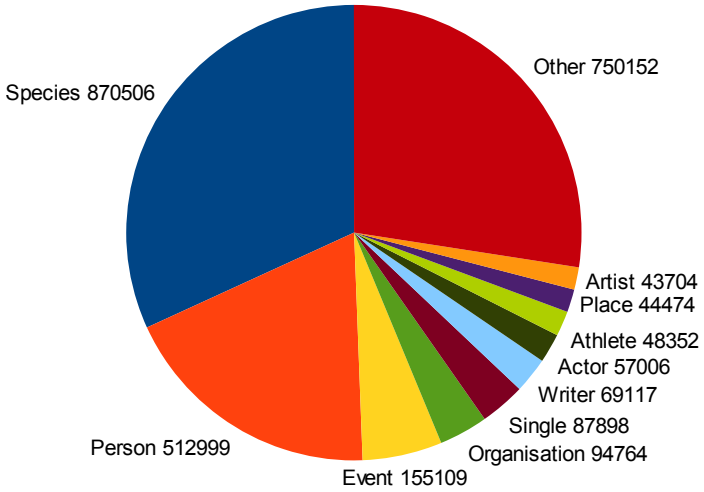

Fig. 5. Largest number of (estimated) missing type statements per class

marginally important instances. In DBpedia, common reasons for missing type statements are

- Missing infoboxes – an article without an infobox is not assigned any type.
- Too general infoboxes – if an article about an actor uses a person infobox instead of the more specific actor infobox, the instance is assigned the type `dbpedia-owl:Person`, but not `dbpedia-owl:Actor`.
- Wrong infobox mappings – e.g., the videogame infobox is mapped to `dbpedia-owl:VideoGame`, not `dbpedia-owl:Game`, and `dbpedia-owl:VideoGame` is not a subclass of `dbpedia-owl:Game` in the DBpedia ontology.
- Unclear semantics – some DBpedia ontology classes do not have clear semantics. For example, there is a class `dbpedia-owl:College`, but it is not clear which notion of *college* is denoted by that class. The term *college*, according to different usages, e.g., in British and US English, can denote private secondary schools, universities, or institutions within universities.[9]

5.2 Typing Untyped Instances in DBpedia

In our second experiment, we have analyzed how well *SDType* is suitable for adding type information to untyped resources. As discussed above, resources may be missing a type because they use no infobox, an infobox not mapped to a type, or are derived from a Wikipedia red link. In particular in the latter case, the only usable information are the incoming properties.

Simply typing all untyped resources with *SDType* would lead to many errors, since there are quite a few resources that should not have a type, as discussed

[9] see `http://oxforddictionaries.com/definition/english/college`

in [1]. Examples are resources derived from list pages,[10] pages about a category rather than an individual,[11] or general articles.[12]

In order to address that problem, we have manually labeled 500 untyped resources into typeable and non-typeable resources. For those resources, we have created features using the *FeGeLOD* framework [13], and learned a ruleset for classifying typeable and non-typeable resources using the *Ripper* rule learner [3]. The resulting rule set has accuracy of 91.8% (evaluated using 10-fold cross validation).

From all 550,048 untyped resources in DBpedia, this classifier identifies 519,900 (94.5%) as typeable. We have generated types for those resources and evaluated them manually on a sample of 100 random resources. The results for various thresholds are depicted in Fig. 6. It can be observed that 3.1 types per instance can be generated with a precision of 0.99 at a threshold of 0.6, 4.0 types with a precision of 0.97 at a threshold of 0.5, and 4.8 types with a precision of 0.95 at a threshold of 0.4.[13]. In contrast, RDFS reasoning on the test dataset generates 3.0 types per instance with a precision of 0.96, which shows that *SDType* is better in both precision and productivity.

With those thresholds, we can generate a total of 2,426,552 and 1,682,704 type statements, respectively, as depicted in Table 3. It can be observed that with the higher threshold guaranteeing higher precision, more general types are generated, while more specific types such as *Athlete* or *Artist*, are rarely found. In most cases, the generated types are consistent, i.e., an *Artist* is also a *Person*, while contradicting predictions (e.g., *Organization* and *Person* for the same instance) are rather rare.

6 Related Work

The problems of inference on noisy data in the Semantic Web has been identified, e.g., in [16] and [8]. While general-purpose reasoning on noisy data is still actively researched, there have been solutions proposed for the specific problem of type inference in (general or particular) RDF datasets in the recent past, using strategies such as machine learning, statistical methods, and exploitation of external knowledge such as links to other data sources or textual information.

[11] use a similar approach as ours, but on a different problem: they try to predict possible predicates for resources based on co-occurrence of properties. They report an F-measure of 0.85 at linear runtime complexity.

Many ontology learning algorithms are capable of dealing with noisy data [19]. However, when using the learned ontologies for inferring missing information using a reasoner, the same problems as with manually created ontologies occur.

[10] e.g., `http://dbpedia.org/resource/Lists_of_writers`
[11] e.g., `http://dbpedia.org/resource/Writer`
[12] e.g., `http://dbpedia.org/resource/History_of_writing`
[13] A web service for DBpedia type completion, as well as the code used to produce the additional types, is available at `http://wifo5-21.informatik.uni-mannheim.de:8080/DBpediaTypeCompletionService/`

Table 3. Results for typing untyped resources, including main types found. The table lists all types which were predicted for at least 1% of the instances in the test set.

Threshold	0.4	0.6
Estimated precision	≥ 0.95	≥ 0.99
Total typed instances	440,849	373,366
Total type statements	2,426,552	1,682,704
Average types per typed instance	5.5	4.5
Distinct types assigned	144	121
Main types:		
Person	236,608 (53.7%)	173,944 (46.6%)
– Athlete	71,226 (16.2%)	544 (<0.1%)
– Artist	21,219 (4.8%)	22 (<0.1%)
– Musical Artist	10,533 (2.4%)	21 (<0.1%)
– Writer	4,973 (1.1%)	0 (0.0%)
Place	79,115 (17.9%)	72,593 (19.4%)
– Settlement	52,622 (11.9%)	23,060 (6.2%)
– Natural Place	4,846 (1.1%)	2,293 (1.0%)
Organization	73,148 (16.6%)	46,988 (12.6%)
– Company	25,077 (5.7%)	21,509 (5.8%)
– Sports Team	15,176 (3.4%)	14,635 (3.9%)
– Record Label	13,444 (3.0%)	13,158 (3.5%)
– Band	12,770 (2.9%)	6 (<0.1%)
Creative Work	15,542 (3.5%)	13,130 (3.4%)
– Album	12,516 (2.8%)	191 (<0.1%)
Species	8,249 (1.8%)	7,988 (2.1%)
– Animal	7,815 (1.7%)	6,744 (1.8%)

One of the first approaches to type classification in relational data is discussed in [10]. The authors train a machine learning model on instances that already have a type, and apply it to the untyped instances in an iterative manner. The authors report an accuracy of 0.81, treating type completion as a single-class problem (i.e., each instance is assigned exactly one type).

The work discussed in [12] assumes that for many instances, there are *some*, but not *all* types. Association rule mining is employed to find common patterns of the type *if type A and B are set, type C is also set*, and apply them to the knowledge base. The authors report that they can add around 3 additional types to an average instance at a precision of 85.6%.

In [1], an approach is introduced which first exploits cross-language links between DBpedia in different languages to increase coverage, e.g., if an instance has a type in one language version and does not have one in another language version. Then, they use nearest neighbor classification based on different features, such as templates, categories, and bag of words of the corresponding Wikipedia article. On existing type information, the authors report a recall of 0.48, a precision of 0.91, and an F-measure of 0.63.

The *Tipalo* system [5] leverages the natural language descriptions of DBpedia entities to infer types, exploiting the fact that most abstracts in Wikipedia follow similar patterns. Those descriptions are parsed and mapped to the WordNet and

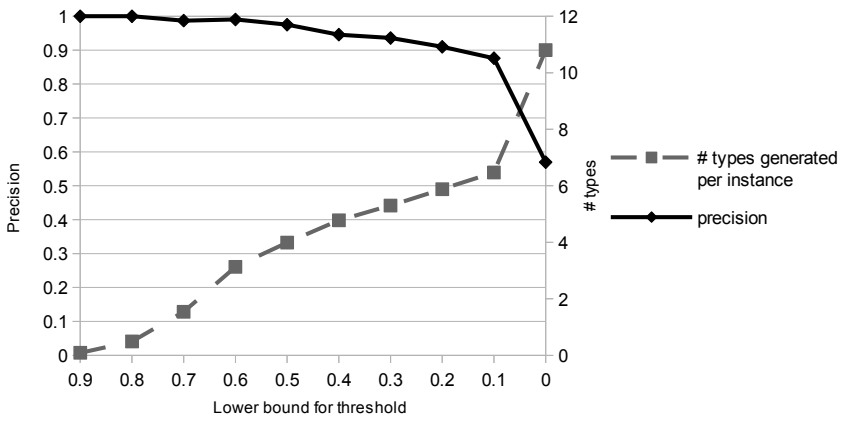

Fig. 6. Precision and average number of type statements per resource generated on untyped resources in DBpedia

DOLCE ontologies in order to find appropriate types. The authors report an overall recall of 0.74, a precision of 0.76, and an F-measure of 0.75.

The authors of [7] exploit types of resources derived from linked resources, where links between Wikipedia pages are used to find linked resources (which are potentially more than resources actually linked in DBpedia). For each resource, they use the classes of related resources as features, and use k nearest neighbors for predicting types based on those features. The authors report a recall of 0.86, a precsion of 0.52, and hence an F-measure of 0.65.

The approach discussed in [15] addresses a slightly different problem, i.e., the mapping DBpedia entities to the category system of OpenCyc. They use different indicators – infoboxes, textual descriptions, Wikipedia categories and instance-level links to OpenCyc – and apply an a posteriori consistency check using Cyc's own consistency checking mechanism. The authors report a recall of 0.78, a precision of 0.93, and hence an F-measure of 0.85.

The approaches discussed above, except for [12], are using specific features for DBpedia. In contrast, *SDType* is agnostic to the dataset and can be applied to any RDF knowledge base. Furthermore, none of the approaches discussed above reaches the quality level of *SDType* (i.e., an F-measure of 88.5% on the DBpedia dataset).

With respect to DBpedia, it is further noteworthy that *SDType* is also capable of typing resources derived from Wikipedia pages with very sparse information (i.e., no infoboxes, no categories, etc.) – as an extreme case, we are also capable of typing instances derived from Wikipedia red links only by using information from the ingoing links.

7 Conclusion and Outlook

In this paper, we have discussed the *SDType* approach for heuristically completing types in large, cross-domain databases, based on statistical distributions. Unlike traditional reasoning, our approach is capable of dealing with noisy data as well as faulty schemas or unforeseen usage of schemas.

The evaluation has shown that *SDType* can predict type information with an F-measure of up to 88.9% on DBpedia and 63.7% on OpenCyc, and can be applied to virtually any cross-domain dataset. For DBpedia, we have furthermore enhanced *SDType* to produce valid types only for untyped resources. To that end, we have used a trained preclassifier telling typeable from non-typeable instances at an accuracy of 91.8%, and are able to predict 2.4 million missing type statements at a precision of 0.95, or 1.7 million missing type statements at a precision of 0.99, respectively. We have shown that with these numbers, we outperform traditional RDFS reasoning both in precision and productivity.

The results show that *SDType* is good at predicting higher-level classes (such as *Band*), while predicting more fine-grained classes (such as *Heavy Metal Band*) is much more difficult. One strategy to overcome this limitation would be to use *qualified relations* instead of only relation information, i.e., a combination of the relation and the type of related objects. For example, links from a music group to an instance of *Heavy Metal Album* could indicate that this music group is to be classified as a *Heavy Metal Band*. However, using such features results in a much larger feature space [13] and thus creates new challenges with respect to scalability of *SDType*.

The type statements created by *SDType* are provided in a web service interface, which allows for building applications and services at a user-defined trade-off of recall and precision, as sketched in [14].

The statistical measures used in this paper cannot only be used for predicting missing types. Other options we want to explore in the future include the validation of existing types and links. Like each link can be an indicator for a type that does not exist in the knowledge base, it may also be an indicator that an existing type (or the link itself) is wrong.

In summary, we have shown an approach that is capable of making type inference heuristically on noisy data, which significantly outperforms previous approaches addressing this problems, and which works on large-scale datasets such as DBpedia. The resulting high precision types for DBpedia have been added to the DBpedia 3.9 release and are thus publicly usable via to the DBpedia services.

Acknowledgements. The authors would like to thank Christian Meilicke for his valuable feedback on this paper.

References

1. Alessio Palmero Aprosio, Claudio Giuliano, and Alberto Lavelli. Automatic expansion of dbpedia exploiting wikipedia cross-language information. In *10th Extended Semantic Web Conference (ESWC 2013)*, 2013.
2. Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics*, 7(3):154–165, 2009.
3. William W. Cohen. Fast effective rule induction. In *12th International Conference on Machine Learning*, 1995.

4. Dieter Fensel and Frank van Harmelen. Unifying Reasoning and Search. *IEEE Internet Computing*, 11(2):96,94–95, 2007.
5. Aldo Gangemi, Andrea Giovanni Nuzzolese, Valentina Presutti, Francesco Draicchio, Alberto Musetti, and Paolo Ciancarini. Automatic typing of dbpedia entities. In *11th International Semantic Web Conference (ISWC 2012)*, 2012.
6. Lise Getoor and Christopher P Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.
7. Andrea Giovanni, Aldo Gangemi, Valentina Presutti, and Paolo Ciancarini. Type inference through the analysis of wikipedia links. In *Linked Data on the Web (LDOW)*, 2012.
8. Qiu Ji, Zhiqiang Gao, and Zhisheng Huang. Reasoning with noisy semantic data. In *The Semanic Web: Research and Applications (ESWC 2011), Part II*, pages 497–502, 2011.
9. Cynthia Matuszek, John Cabral, Michael Witbrock, and John DeOliveira. An introduction to the syntax and content of cyc. In *Proceedings of the 2006 AAAI spring symposium on formalizing and compiling background knowledge and its applications to knowledge representation and question answering*, 2006.
10. Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.
11. Eyal Oren, Sebastian Gerke, and Stefan Decker. Simple algorithms for predicate suggestions using similarity and co-occurrence. In *European Semantic Web Conference (ESWC 2007)*, pages 160–174. Springer, 2007.
12. Heiko Paulheim. Browsing linked open data with auto complete. In *Semantic Web Challenge*, 2012.
13. Heiko Paulheim and Johannes Fürnkranz. Unsupervised Feature Generation from Linked Open Data. In *International Conference on Web Intelligence, Mining, and Semantics (WIMS'12)*, 2012.
14. Heiko Paulheim and Jeff Z. Pan. Why the semantic web should become more imprecise. In *What will the Semantic Web look like 10 years from now?*, 2012.
15. Aleksander Pohl. Classifying the wikipedia articles in the opencyc taxonomy. In *Web of Linked Entities Workshop (WoLE 2012)*, 2012.
16. Axel Polleres, Aidan Hogan, Andreas Harth, and Stefan Decker. Can we ever catch up with the web? *Semantic Web Journal*, 1(1,2):45–52, 2010.
17. Purvesh Shah, David Schneider, Cynthia Matuszek, Robert C. Kahlert, Bjørn Aldag, David Baxter, John Cabral, Michael J. Witbrock, and Jon Curtis. Automated population of cyc: Extracting information about named-entities from the web. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, pages 153–158. AAAI Press, 2006.
18. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 697–706. ACM, 2007.
19. Johanna Völker and Mathias Niepert. Statistical schema induction. In *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications - Part I*, pages 124–138, Berlin, Heidelberg, 2011. Springer-Verlag.
20. W3C. RDF Semantics, 2004. `http://www.w3.org/TR/rdf-mt/`.