# Improving the Quality of Linked Data Using Statistical Distributions

**Heiko Paulheim* and Christian Bizer**
*University of Mannheim, Germany*
*Data and Web Science Group*

## ABSTRACT

Linked Data on the Web is either created from structured data sources (such as relational databases), from semi-structured sources (such as Wikipedia), or from unstructured sources (such as text). In the latter two cases, the generated Linked Data will likely be noisy and incomplete. In this paper, we present two algorithms that exploit statistical distributions of properties and types for enhancing the quality of incomplete and noisy Linked Data sets: *SDType* adds missing type statements, and *SDValidate* identifies faulty statements. Neither of the algorithms uses external knowledge, i.e., they operate only on the data itself. We evaluate the algorithms on the DBpedia and NELL knowledge bases, showing that they are both accurate as well as scalable. Both algorithms have been used for building the DBpedia 3.9 release: With SDType, 3.4 million missing type statements have been added, while using SDValidate, 13,000 erroneous RDF statements have been removed from the knowledge base.

***Keywords:*** Data Quality, Type Completion, Error Detection, Noisy Data, Semi-Structured Data, DBpedia, NELL

## INTRODUCTION

Many of the data sets that are published as Linked Data on the Web (Bizer et al. 2009a) have been created from structured sources such as relational databases and are thus *strongly structured* (Bizer and Cyganiak, 2006). In addition, Linked Data is also extracted from *semi-structured sources*, such as Wikipedia (Bizer et al. 2009b, Lehmann et al. 2014) or from *unstructured sources* such as free text (Ramakrishnan et al. 2006, Augenstein et al. 2012, Gerber and Ngonga Ngomo 2012).

Linked Data which has been extracted from semi- and unstructured sources is likely to contain noise in the form of wrong RDF statements (Dutta et al., 2014). The data is also likely to be rather incomplete with respect to its schema. For example, Linked Data sets which have been generated from relational databases usually contain type information for each resource since such information is present in most databases. This does not hold for data sets that were extracted from semi-structured and unstructured sources, where that information may be missing in the original source, or the information extraction system was not able to extract it. Furthermore, heuristically extracted data sets are more likely to contain a certain level of noise in the form of wrong statements.

In order to improve the quality of such noisy and incomplete Linked Data sets, this article proposes the *SDType* method for adding missing type information to a data set, as well as the *SDValidate* method for identifying possibly wrong statements which were generated by the information extraction system. Both methods do not use any external knowledge, i.e., they exploit solely the data set itself. The proposed methods rely on statistical distributions of types and properties, i.e., characteristic distributions of the types of a property's subjects and objects. We show that both algorithms have a high accuracy and scale to large knowledge bases such as DBpedia and NELL. Both algorithms have been used to improve the quality of the DBpedia 3.9 release: With SDType, we have added 3.4 million missing type statements, while with SDValidate, 13,000 wrong statements have been identified and removed.

The rest of the article is structured as follows. We first discuss data quality issues that are particular to Linked Data sets which have been extracted from semi- and unstructured sources, i.e., noisy and incomplete data sets, and describe the two datasets used for evaluation in this article, i.e., DBpedia, and a Linked Data version of NELL. Then, we introduce the idea of using statistical distributions of types and properties for quality improvement, which underlies both algorithms discussed in this article, and we further define the two the algorithms *SDType* and *SDValidate*. For both algorithms, we discuss the evaluation on the two datasets, and describe how they have been deployed for the DBpedia release 3.9. We present an implementation of the algorithms in a relational database system, and, based on that implementation, discuss their complexity. We conclude the paper with a review of related work, a summary, and an outlook on future work.

Parts of the work presented in this article have been published as part of the conference paper "Type Inference on Noisy RDF Data" (Paulheim and Bizer, 2013). While that conference paper only discusses the SDType algorithm, this article extends the conference paper by introducing the complementary SDValidate algorithm including its evaluation, and further evaluates the differences between SDType and type inference using classical ontology reasoning. Furthermore, it compares the results achieved with both algorithms on an additional dataset, i.e., a Linked Data version of NELL.

## DATA QUALITY ISSUES WITH NOISY AND INCOMPLETE LINKED DATA SETS

Data quality is not a single measure, but has multiple dimensions. Pipino et al. (2002) list several of those dimensions, ranging from accessibility to completeness. In addition, many of those dimensions cannot be assessed in a context-free manner, but depend on the task at hand, such as relevance. Thus, data quality is generally conceived as "fitness for use" (Wang et al., 1996), i.e., the capability of data to fit the requirements of a specific user given a certain use case.

Linked data sets created from semi-structured or unstructured sources face certain data quality problems that are unique to that class of data sets. The first difference is concerned with the *completeness of type information*: Due to missing type information in the semi-structured or unstructured sources or due to errors happening in the information extraction process type information is often missing for a relevant part of the described resources.

Although completeness in Linked Data is not a problem from a logic point of view, given the formal semantics of RDF (the open world assumption allows for missing knowledge), having complete type information is essential for many use cases. For example, a query for all cities lo-

cated in a country will only return useful results if a sufficient number of instances has the type `dbpedia-owl:City`.[1]

Furthermore, datasets created from semi-structured or unstructured data sets may contain *noise*. While relational databases may also contain errors, they are usually *factual errors*, e.g., a wrong capital or population number for a country. In contrast, the noise created in information extraction processes often comprises different sorts of errors, such as a building or a person being defined as the capital of a country.

In this paper, we use two noisy and incomplete datasets. The first one is DBpedia (Bizer et al. 2009b, Lehmann et al. 2014), the second one is a Linked Data version of NELL (Zimmermann et al., 2013).

DBpedia is a large-scale, multilingual, cross-domain knowledge base. It uses data from infoboxes in Wikipedia, which are mapped to an ontology in a crowd-sourced process. A number of different extractors read and transform the information from the infoboxes, which is provided as Linked Open Data. In the past years, DBpedia has become one of the central and most widely used datasets in the Linked Open Data cloud (Bizer et al., 2009a). Due to its wide popularity and coverage, we use it as an example for looking at data quality in more detail.

For the experiments presented in this article, we used DBpedia version 3.9 (released September 2013). This version of DBpedia[2] contains information about 4.0 million things, 3.2 of which are assigned one or more types in the DBpedia ontology, as well as other ontologies, such as schema.org,[3] UMBEL,[4] and YAGO (Hoffart et al., 2013). For example, there are 832,000 persons, 639,000 places, 209,000 organizations, etc. Furthermore, DBpedia extracts information from Wikipedia in 119 languages in total.

While offering a broad coverage, DBpedia is by no means free of errors. There are various sources of those errors, ranging from factual errors in Wikipedia over wrongly used Wiki markup (e.g., using wrong types of infoboxes) to bugs and limitations in the DBpedia data extraction code, e.g., in dealing with many degrees of freedom for encoding numbers in Wikipedia (Wienand and Paulheim, 2014).

Wikipedia itself may contain factual errors, as well as noise (e.g., a statement that a soccer player's team is the country Germany, while the statement's object should be the German national soccer team, not the country). Weaver et al. (2006) estimate the fraction of wrong statements to be 2.8%.[5]

---

[1] The following namespace conventions are used throughout this paper: dbpedia=http://dbpedia.org/resource/, dbpedia-owl=http://dbpedia.org/ontology/, yago=http://dbpedia.org/class/yago/, foaf=http://xmlns.com/foaf/0.1/Person, owl=http://www.w3.org/2002/07/owl#, rdfs=http://www.w3.org/2000/01/rdf-schema#

[2] Unless otherwise indicated, all statements about the DBpedia knowledge base refer to version 3.9.

[3] http://www.schema.org/

[4] http://www.umbel.org/

[5] Since the authors estimate the correctness of all links in a Wikipedia page, not only links in infoboxes, not every error would result in a wrong statement in DBpedia. However, the fraction can be seen as a rough estimate of the information quality in DBpedia.

In previous work (Paulheim and Bizer 2013), we have performed the analysis of another quality dimension in DBpedia, i.e., completeness. While it is difficult, if not impossible, to make a statement of the overall completeness of DBpedia (and Wikipedia), we have attempted an approximation of the completeness with respect to one particular type of information in DBpedia, i.e., direct types. We have estimated the number of missing type statements in DBpedia to at least 2.7 million.

By design, DBpedia cannot contain any information that is not contained in Wikipedia. While providing an ontology with the data in theory allows for reasoning to infer missing information, classical ontology reasoning often fails on DBpedia, since the A-box data (i.e., the facts about single instances) is too noisy. We have shown that even in a knowledge base where 99.9% of the facts are correct, ontology reasoning can lead to completely skewed and nonsensical results (Paulheim and Bizer, 2013). Thus, classical ontology reasoning, although one of the main promises and selling points of the Semantic Web, is not a remedy to that problem.

NELL (Never ending language learning) is a system that, based on a small set of examples and seed patterns, learns language patterns and instantiations for relations. For example, the sentence *Bavaria Munich keeper Manuel Neuer talks about his future ambitions* contains an instantiation of the relation *plays_for_team* with the subject *Manuel Neuer* and the object *Bavaria Munich*, as well as the pattern *Y keeper X*, which is an indicator for such instantiations. NELL aims at learning both new patterns as well as new instances in an iterative process, using a web crawl as its base corpus. In each iteration, the patterns and instantiations with the highest confidence are added to the knowledge base, after checking for consistency using an ontology (Carlson et al., 2010).

For our purposes, we use a linked data version of NELL, which represents facts extracted by NELL in RDF, together with a rich OWL ontology (Zimmermann et al., 2013).[6] The version used in this paper contains 262,000 locations, 195,000 organizations, 176,000 persons, etc.

In contrast to DBpedia, there is not too much work on the data quality of NELL. In the original publications, the authors state a precision of at most 90% in the extracted statements (e.g Carlson et al, 2010). Although we do not know the number of missing type statements, we have observed that the fraction of instances that do not have any type is similar to DBpedia, and the class hierarchies have a comparable complexity (see below). Thus, we assume that the problem of missing types is of comparable severity in both datasets.

With respect to correctness, there are differences. DBpedia is extracted from the Wikipedia infoboxes without checking consistency or plausibility constraints. In contrast, NELL is designed in a rather defensive way that only promotes patterns and facts with high confidence values, and exploits some basic reasoning to avoid the inclusion of wrong facts (Carlson et al., 2010). Thus, there is already some implicit quality control in the NELL dataset, and we expect a lower number of errors. In our evaluation, we will discuss how these quality aspects influence our approach.

In our experiments, we have observed one frequent source of errors, i.e., the handling of homonymy, which is obviously hard for NELL, and hence, homonymous resources often merged into one resource. One such example identified by SDValidate is *concrete* as a building material, and the Thomas Bernhard novel *Concrete*, which are both represented by the same resource.

[6] We use version 08m.690 of the dataset, accessed on March 29th, 2014

Table 1 depicts some basic characteristics of the datasets. First of all, it can be observed that both datasets have a decent fraction of untyped instances. This makes them good candidates for applying methods for automatic type completion. Furthermore, by construction, both datasets are likely to contain noise, thus, we also expect them to benefit from automatic error correction.

| | DBpedia | NELL |
|---|---|---|
| Number of instances | 3,600,638 | 1,475,390 |
| Number of type assertions | 13,225,156 | 5,688,414 |
| Number of relation assertions | 10,645,207 | 81,527 |
| Number of distinct classes | 359 | 270 |
| Number of distinct properties | 1775 | 548 |
| Average depth of class hierarchy | 2.4 | 3.9 |
| Fraction of untyped instances | 19.9% | 20.8% |
| Average number of types per typed instance | 5.6 | 4.9 |
| Average number of instances per class | 38,003 | 21,547 |
| Average number of ingoing properties per instance | 8.5 | 2.7 |
| Average number of outgoing properties per instance | 8.8 | 1.7 |

Table 1: Statistics about the datasets used for the evaluation

## STATISTICAL DISTRIBUTIONS OF PROPERTIES AND TYPES

Both algorithms discussed in this article, i.e., *SDType* and *SDValidate*, use statistical distributions of statements connecting pairs of A-box resources, and of `rdf:type` statements. In particular, for each property, we determine the number of instances of a certain type appearing in the property's subject and object position. The resulting distributions characterize the property.

Table 2 shows an example distribution of the DBpedia property `dbpedia-owl:location`. The table reads as follows: given the set of triples that have the predicate `dbpedia-owl:location`, 100% of the subjects are of type `owl:Thing`, 69.8% of the subjects are of type `dbpedia-owl:Place`, and so on. For the objects, 88.6% are of type `owl:Thing`, 87.6% are of type `dbpedia-owl:Place`, and so on.

A noteworthy observation is that not all objects are of type `owl:Thing`. This is due to the fact that types in DBpedia (including `owl:Thing`) are only generated if an infobox is present. While there is an infobox on the page from which the subject was created (otherwise, the statement would not exist at all), not all objects are generated from pages having an infobox. Thus, the number is lower.

The distribution of a property can be used for several purposes. The basic idea of the SDType algorithm is that when observing a certain property in the predicate position of a statement, we can infer the types of the statement's subject and object with certain probabilities. Furthermore, if

there is a large deviation between actual types of the subject and/or object and the apriori probabilities given by the distribution, we can mark the statement as possibly wrong – that is the basic idea of the SDValidate algorithm.

| 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 |
| ... | ... | ... |

Table 2: Distribution of subject and object types for the property `dbpedia-owl:location`. The percentages of the types of subjects and objects do not sum up to 100% because each resource can have multiple types, so that the sum is larger than 100%.

In addition to the distributions per statement, we also use the apriori distributions of each type, i.e., the percentage of resources that have a certain type. These apriori probabilities can be used to assess the value of a certain property in deriving information from a statement using that property. If the distribution of a property is close to the apriori distribution, the information gained from the statement is less valuable.

## THE SDTYPE ALGORITHM FOR TYPE COMPLETION

As discussed above, type information is useful, as well as often incomplete. The SDType algorithm is designed to use information about statistical distributions for assigning types to untyped resources, and thus to increase the completeness of `rdf:type` statements in the knowledge base.

### Approach of SDType

SDType uses properties that connect two resources as indicators for their types, i.e., it is a *link-based object classification approach* (Getoor and Diehl, 2005). The basic idea is to use each ingoing and outgoing properties of a a resource as an indicator for that resource's type. For each property, we use the **s**tatistical **d**istribution (hence the name *SDType*) of types in the subject and object position of the property for predicting the instance's types. SDType can be seen as a weighted voting approach, where each property can cast a vote on its object's types, using the statistical distribution to weigh its votes.

Consider, for example, the distribution shown in Table 2. Given that we observe a triple like

```
:x dbpedia-owl:location :y ,
```

we can assign the following probabilities:

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 probabilities measuring how likely a type t is, given a resource with a certain property *prop*, expressed as $P(t|(\exists prop.T))$, where *prop* may be an ingoing or an outgoing property (with usually different probabilities) These conditional probabilities are the probabilities that a resource with a certain (ingoing or outgoing) property have a given type, as shown in Table 2. If multiple statements with the same property exist, the respective conditional probabilities are taken into account multiple times.

These conditional probabilities are obtained directly from the statistical distributions shown above. Furthermore, each property is assigned a certain weight $w_{prop,}$ which reflects its predictive power. Since that predictive power may be different for predicting types in the subject and in the object position, the weights are different in both cases, i.e., in the example above, there are two weights $w_{\text{dbpedia-owl:location}}$ and $w_{\text{dbpedia-owl:location}^{-1}}$.

The motivation of using weights for properties to avoid problems with skewed knowledge bases, i.e., knowledge bases in which the extension of some types are several orders of magnitude larger than that of others. In such cases, using only statistical distributions without any correction can lead to false type predictions, which are biased towards the majority types. This is a problem in particular with general purpose properties, such as `rdfs:label` or `owl:sameAs`, which are more or less equally distributed in the overall knowledge base.

To avoid those problems, we define property weights $w_p$, which measure the deviation of the property's distribution to the apriori distribution of all types in the knowledge base. The stronger this deviation, the higher we assess the property's predictive power. Formally, $w_{prop}$ is defined as

$$w_{prop} := \sum_{\text{all types } t} \left(P(t) - P(t|(\exists prop.T))\right)^2$$

To compute the weights, both the statistical distributions per property as well as the apriori probabilities P(t) are used. Note that as discussed above, we define individual weights for p and $p^{-1}$, i.e., different weights are used for predicting the types, depending on whether the resource is used in the predicate's subject or object position.

Using the conditional probabilities and the property weights, we implement a weighted voting approach, where for each property, a vote for the types in the property's statistical distribution is cast, assigning a likelihood to each type. The overall predicted type distribution for a resource is then given by the weighted sum of all likelihoods. Thus, we define the confidence that a resource r has the type t as

$$\sum_{\text{all properties prop of } r} P(t(r)|(\exists prop.T)(r)),$$

with the normalization factor $v$ defined as

$$v := \frac{1}{\sum_{\text{all properties prop of } r} w_{prop}}.$$

In those formulas, “all properties prop of r” includes properties of all statements that have *r* in the subject *or* the object position. As discussed above, subject and object position properties are handled separately.

When implementing the approach, a *confidence threshold t* is applied, i.e., all type statements with a confidence larger than t are assumed to be correct.

Looking at the 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 the type `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.

To illustrate the effect of weights, we consider the following example:

```
x dbpedia-owl:location y .
x foaf:name z,
```

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, the confidences of types for x without property weights (i.e., setting all weights $w_p$ to 1 in the definition) would be 0.14 for `dbpedia-owl:Person`, and 0.6 for `dbpedia-owl:Place`.

When using weights, the numbers are different. In the above examples, the weights for `dbpedia-owl:location`$^{-1}$ and `foaf:name`$^{-1}$ are 0.77 and 0.17, respectively. Using these weights, the confidence scores are 0.05 for `dbpedia-owl:Person` and 0.78 for `dbpedia-owl:Place`. 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 lead to a reduction of wrong results originating from skewed datasets.[7]

## Evaluation of SDType

We have performed two evaluations of SDType on each dataset. The first evaluation uses instances that already have types assigned, taking that type information as a gold standard. We randomly sampled three datasets, each containing 10,000 instances, and tried to reconstruct their types. The three datasets were sampled to contain instances with at least one, at least 10, and at least 25 ingoing properties. For the experiment with DBpedia, we used the mapping-based properties and the infobox types dataset, the latter not only containing types from the DBpedia ontology, but also from UMBEL and schema.org.

In a second experiment, we attempt to assign types to non-typed instances. For that experiment, we sampled 100 random untyped instances from each dataset, and manually inspected the types generated. To make this inspection easier on the NELL dataset, we restricted ourselves to instances that have links to Wikipedia pages (for DBpedia instances, the Wikipedia page is always given implicitly). For that second experiment, we report precision as well the number of types that can be generated.

In both experiments, we use only ingoing properties to predict types. Since types and outgoing properties are generated in the same process in DBpedia (i.e., using mappings from the infobox to the DBpedia ontology), using outgoing properties for type prediction would oversimplify the problem in the first experiment. In the second experiment, we only consider untyped in-

[7] The actual numbers for DBpedia are: P(dbpedia-owl:Person|foaf:name$^{-1}$) = 0.273941, P(dbpedia-owl:Place|foaf:name$^{-1}$) = 0.314562, P(dbpedia-owl:Person|dbpedia-owl:location$^{-1}$) = 0.000236836, P(dbpedia-owl:Place|dbpedia-owl:location$^{-1}$) = 0.876949.

stances – those instances mostly do not have infoboxes and hence no outgoing properties. In order to make the results comparable, we applied the same restriction on the NELL dataset.

Figures 1 and 2 show the results of SDType for the first experiment. For DBpedia, it can be observed that SDType works sufficiently well on the overall dataset (i.e., instances that have at least one ingoing property), achieving an F-measure of 88.5%, the results are slightly better on instances that have at least 10 or 25 ingoing properties, with an F-measure of 88.9% and 89.9%, respectively (in each of the three cases, the optimal confidence threshold is t=0.34). The differences show more significantly in the precision@95% (i.e. the precision that can be achieved at 95% recall), which is 0.69 (min. one property), 0.75 (min. ten properties), and 0.82 (min. 25 properties), respectively. The observations are similar on the NELL dataset, with a maximum F-measure of 93.2% (min. one property, at a confidence threshold of t=0.26), 94.2% (min. 10 properties, at a confidence threshold of t=0.27), and 95.7% (min. 25 properties, at a confidence threshold of 0.35), and a precision@95% of 0.88 (min. one property), 0.92 (min. ten properties), and 0.96 (min. 25 properties).

These numbers indicate that instances with more ingoing properties can be typed more accurately. Using the average of the predicted type vectors for all ingoing properties, the influence of single wrong statements is reduced if more properties are present.

To contrast SDType with ontology reasoning, which is a standard method for inferring types on RDF data, we have run the reasoner Pellet (Sirin et al., 2007) on the same datasets. For typing an instance with Pellet, we load the T-box of the ontology, add all statements which involve the resource at hand, and let Pellet determine the types. The comparison of that reasoning approach with SDType is shown in Figures 3 and 4.

It can be observed that both for DBpedia and for NELL, SDType outperforms the standard reasoning approach, both in terms of recall and precision, in most cases. On the DBpedia dataset, the precision of the reasoning approach drops with more ingoing statements, while the precision of SDType remains stable and even slightly rises, at an overall higher level of recall. The reason is that the reasoning approach *accumulates* errors from single wrong statements, while SDType *reduces* the influence of single wrong statements as the number of statements increases. On the NELL dataset, the picture for the standard reasoning approach is a bit different. Here the precision drops with a larger amount of ingoing properties. The likely explanation is the consistency checking in NELL, which leads to removal of much noise. The remaining statements are very likely to be correct, which also eases the reasoning on them. By using ontology-based consistency checking, resources involved in more statements have undergone more consistency checks are thus easier to type for a reasoner.

In our second experiment, we have analyzed how well SDType is suitable for adding type information to *untyped* resources. The results for various thresholds are depicted in Figures 5 and 6. It can be observed that on the DBpedia dataset, 4.4 types per instance can be generated with a precision of 0.99, and 5.4 types with a precision of 0.95 at a threshold of 0.4. The results on the NELL dataset are similar, with the precision dropping slightly faster. The bulge at a threshold of 0.5 is caused by errors made on only two instances, for which SDType made wrong predictions at a high confidence level, and thus not significant.

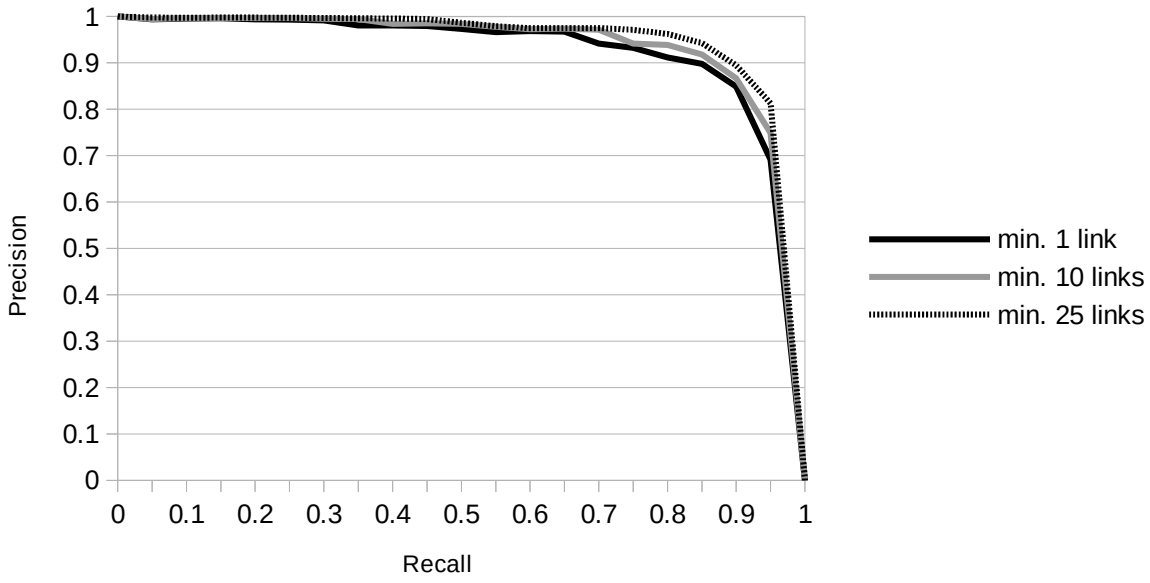


*Figure 1: Results of SDType, using typed instances in DBpedia as a gold standard.*

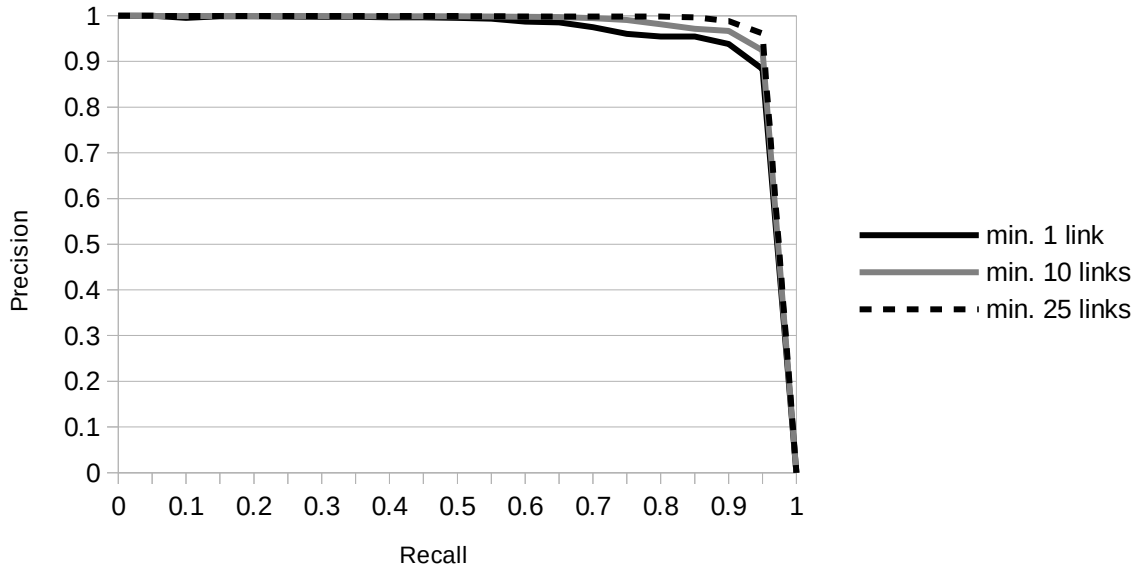


*Figure 2: Results of SDType on NELL, using typed instances in NELL as a gold standard.*

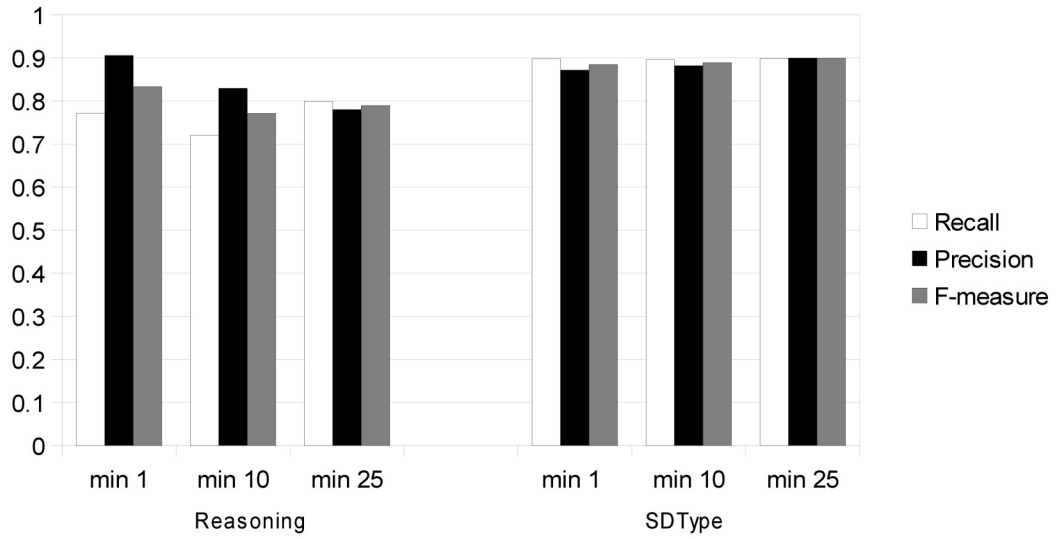


*Figure 3: Comparison of standard reasoning and SDtype on the DBpedia dataset*

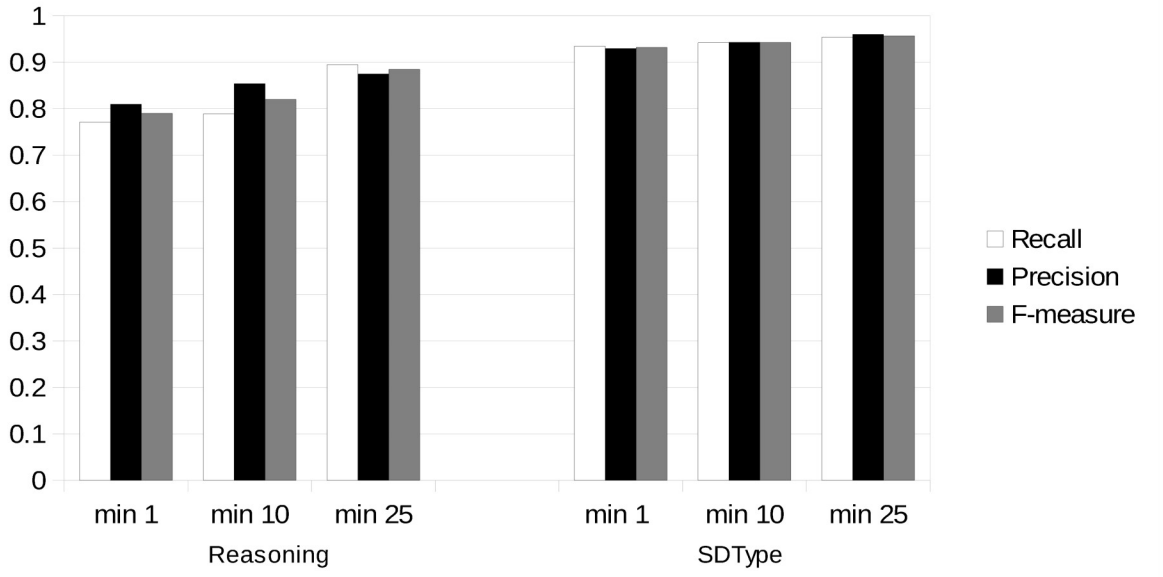


*Figure 4: Comparison of Pellet reasoning and SDType on the NELL dataset*

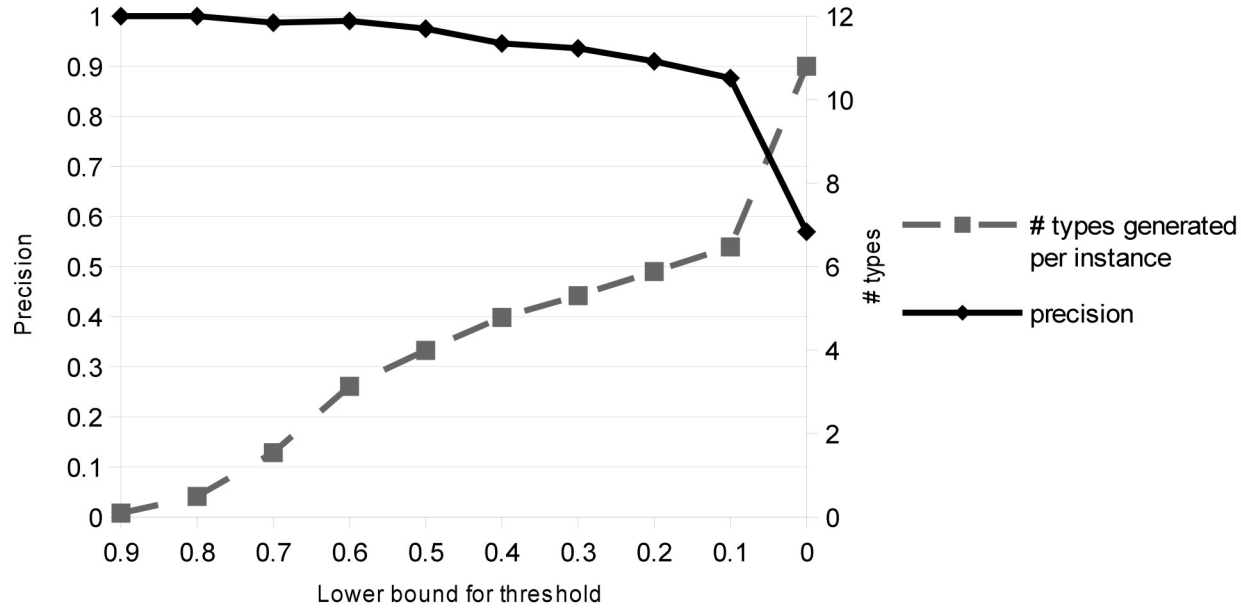


*Figure 5: Results of SDType on untyped instances in DBpedia*

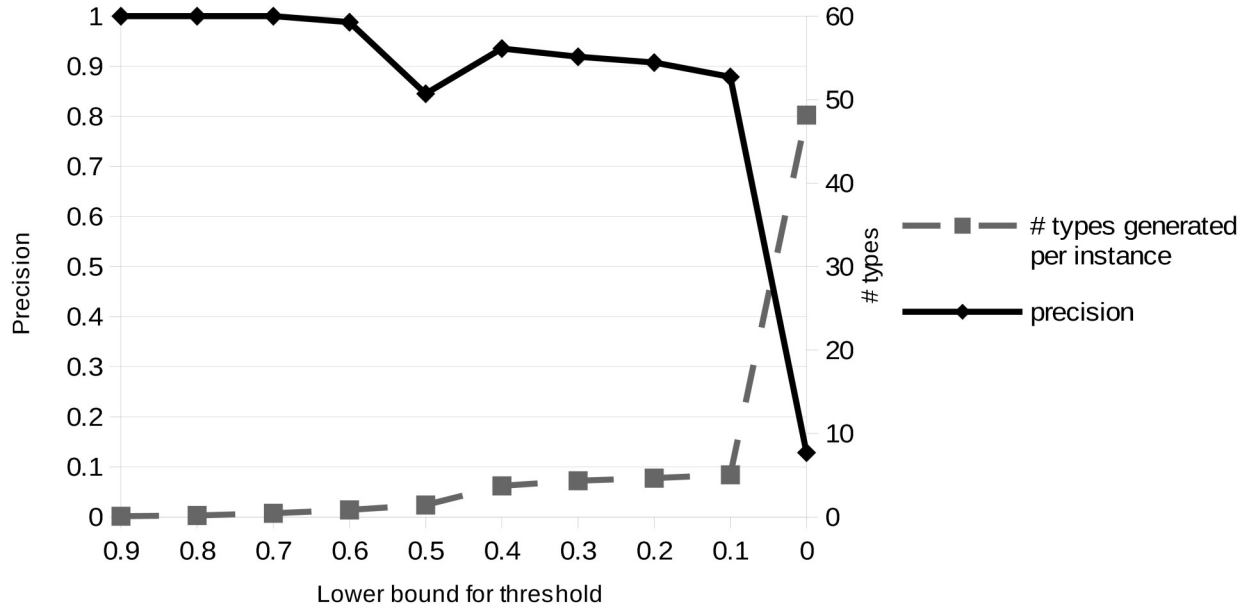


*Figure 6: Results of SDType on Untyped Instances in the NELL dataset*

### Deployment in DBpedia

From the second experiment, we have observed that for achieving an overall precision of 0.95 and 0.99 on untyped instances on DBpedia, we have to set a threshold on the confidence level of 0.4 and 0.6, respectively. With those thresholds, we can generate a total of 3.4 million and 2.2 million type statements in DBpedia, respectively, as depicted in Table 3. With the higher threshold guaranteeing higher precision, more general types such as `dbpedia-owl:Person` are generated, while more specific types such as `dbpedia-owl:Athlete` or `dbpedia-owl:Artist`, are rarely found. In most cases, the generated types are consistent, i.e., an artist is also a person, and, since the types in the DBpedia ontology are fully materialized, all super-types are automatically included in the prediction. In contrast, contradicting predictions (e.g., `dbpedia-owl:Organization` and `dbpedia-owl:Person` for the same instance) are rather rare, but since there are only few statements about disjointness in the DBpedia ontology (see below), those cases are hard to detect automatically.

We have added the dataset that results from applying a confidence threshold of t=0.4 to the DBpedia 3.9 release. The dataset adds another 3.4 million type statements to the 15.9 million that already exist in the knowledge base, and hence increases the number of type statements by 21%. For example, we have identified 293,000 additional persons (an increase of 35%), 115,000 additional places (an increase of 18%), and 120,000 additional organizations (an increase of 57%).

| Threshold | 0.4 | 0.6 |
|---|---|---|
| Estimated Precision | 0.95 | 0.99 |
| Newly typed instances | 626,662 | 510,838 |
| Total type statements | 3,402,539 | 2,228,647 |
| Avg. types per instance | 5.4 | 4.4 |
| Distinct classes | 159 | 130 |
| Main types: | | |
| Person | 292,398 (46.7%) | 191,490 (37.5%) |
| Organization | 119,123 (19.0%) | 67,872 (13.3%) |
| Place | 114,093 (18.2%) | 99,437 (19.5%) |
| Work | 37,176 (5.9%) | 33,007 (6.5%) |

Table 3: Results of SDType for assigning types to untyped instances in DBpedia 3.9. Besides basic measures, the table reports the number of type statements found for large classes, and the relative number of all newly typed instances that belong to those classes.

## THE SDVALIDATE ALGORITHM FOR ERROR DETECTION

While SDType uses statistical distributions for predicting types, the basic idea of the SDValidate algorithm is to use the statistical distributions to assess the correctness of statements which relate two resources.[8] SDValidate assigns a confidence score to each statement which reflects the devi-

[8] Statements whose object is not an entity, but a literal (e.g., a number) are not handled by SDValidate.

ation of the types predicted by the statement from the statement's object's actual types set in the knowledge base.

### Approach of SDValidate

SDValidate follows a three step approach for identifying incorrect statements. The first step computes the *relative predicate frequency (RPF)* for each statement, which describes the frequency of a predicate/object combination. All statements which have a low RPF are selected for further inspection. The rationale behind that approach is that statements with a frequent predicate/object combination are more likely to be correct than a small number of "outlying" statements with an infrequent predicate/object combination. The RPF of a statement *s* is defined as the probability that a statement sharing the object with *s* also shares the relation with *s:*

$$RPF(s) := P(pred \mid obj) = \frac{|statements\ with\ pred(s) \wedge obj(s)|}{|statements\ with\ obj(s)|}$$

For example, the DBpedia resource `dbpedia:Germany` has around 38,000 ingoing statements. Among those, there is the statement

```
dbpedia:CORSIKA dbpedia-owl:author dbpedia:Germany .
```

Since this is the only ingoing statement to `dbpedia:Germany` with the property `dbpedia-owl:author`, the RPF of that property (w.r.t. `dbpedia:Germany`) would be 0.000026. If we filter by a significance level of 0.05 or 0.01, this statement would be selected in the first step.

The second step uses the statistical distributions of properties and types to assign a score to each of the selected statements. For each property, there is a prediction vector, assigning probabilities to the predicate's subject and object. The SDValidate algorithm compares these vectors to the respective resources' actual types. The *cosine similarity* of the two vectors is used as a confidence score for the statement.

As discussed above, the type information of a statement's object are more likely to be error-prone than those of a statement's subject when applying heuristic methods for dataset creation. Thus, we constrain the algorithm to compare the predicate's type distribution for the object position with a statement's object's types.

Given a statement s with predicate prop and object o, we define the confidence in the statement being true as

$$conf(s) := \frac{\sum_{all\ types\ t} p\left(t \mid prop^{-1}\right) \cdot d(t,o)}{\sqrt{\sum_{all\ types\ t} p\left(t \mid prop^{-1}\right)^2} \cdot \sqrt{\sum_{all\ types\ t} d(t,o)^2}},$$

where d(t,o) denotes whether resource o has type t, i.e.,

$$d(t,o) := \begin{cases} 1 & if\ o\ has\ type\ t \\ 0 & otherwise \end{cases}$$

In the third step, we apply a threshold $\tau$ above which statements are regarded to be true. Statements with a confidence below that threshold are marked as being potentially wrong. In the example above, the actual types for `dbpedia:Germany` comprise types such as Place and Country, while the predicted types given the property `dbpedia-owl:author` include Person and

Writer. Thus, the cosine similarity of both vectors is rather low at 0.105[9], and the statement would be discarded given a higher threshold.

## Evaluation of SDValidate

For evaluating SDValidate, we have run the algorithm on random subsets of DBpedia statements, and evaluated the precision manually by inspecting the statements marked as wrong by SDValidate. Evaluating recall, on the other hand, would be much more time consuming, since it requires manual inspection of all the statements in the test set, not only those marked wrong by our approach. Therefore, we do not report recall.

We evaluate three different values for p for the statement selection step by relative predicate frequency: $s \leq 1.0$ (which corresponds to selecting all statements for inspection), as well as $s \leq 0.05$ and $s \leq 0.01$, for selecting only the statements whose predicate is not used by a significant subset of the statements. To conduct these evaluations, we used three test sets. For the case of $s \leq 1.0$, we randomly sampled 10,000 resources in DBpedia. For the test sets of $s \leq 0.05$ and $s \leq 0.01$, we randomly sampled 10,000 resources with 20 and 100 ingoing properties, respectively, to make sure that it is possible to select any statements in the first step. We then ordered the results by confidence in ascending order, and inspected the top results from the list.

The results of SDValidate on the DBpedia dataset are depicted in Figure 7. It can be observed that the application without a pre-selection provides results that lack sufficient precision. The reason is that while SDType uses the average of several statements and thus provides results based on a larger number of statements, SDValidate takes fewer information into account for rating the confidence of a statement, and hence requires additional indicators. The evaluation with a pre-selection at significance levels of 0.05 and 0.01 provides a different picture. It can be observed that in both cases, a precision above 0.9 (0.92 for significance level of 0.01, 0.93 for 0.05) can be achieved.

On the NELL dataset, the results are much worse, as shown in Figure 8. Here, a maximum precision of only 0.69 can be reached with prefiltering by a significance threshold of 0.01. When comparing the curves with the results achieved on DBpedia, it is evident that no statements are marked with very low confidence scores, i.e., scores below 0.05. This hints at the fact that the NELL dataset is already pre-checked for consistency, as discussed above, i.e., most of the obvious errors have already been removed. However, at lower thresholds, we are still able to discover wrong statements, but at the price of more false negatives.

Since both datasets come with ontologies that include disjointness statements, ontology reasoning can be used for checking the validity of statements by reasoning (see section "Related Work"). In order to use reasoning as a baseline, we loaded the respective ontologies into the Pellet reasoner (Sirin et al., 2007), added the statement to check together with its subject's and object's type statements, and had Pellet validate the consistency of the resulting model. However, that approach did not find any wrong statements for both datasets.

The reason why reasoning does not work as an alternative to SDValidate are different for both datasets: On the DBpedia dataset, the reason is probably the very low number of disjointness axioms (there are only 17 such axioms between very high-level classes). For the NELL dataset, reasoning is also employed for consistency checking when compiling the dataset, so that no addi-

[9] It is not 0 because both vectors share the common type `owl:Thing`.

tional conflicts can be detected with reasoning afterwards. That said, it is remarkable that SDValidate is capable of identifying additional errors on the NELL dataset, which are not found by classic ontology reasoning.

In order to obtain an impression of the types of errors identified by SDValidate, we performed an additional analysis of the errors. For DBpedia, we can track the Wikipedia page from which each statement was extracted. This allows for a detailed error analysis. We classified the errors identified on the DBpedia dataset into the following categories:

- Links in longer texts. This type of error occurs when the infobox entry from which the statement is extracted contains a text expression which is not linked as a whole, but only a part of which is linked. An example is the statement `dbpedia:Claire_Marshall dbpedia-owl:education dbpedia:Devon .`
  Here, Devon is a county in England and not an educational institution. The corresponding infobox entry in Wikipedia says "Blundell's School, Devon", where "Blundell's School" is linked to the page about the school, and "Devon" is linked to the county. The DBpedia extraction framework creates two statements out of this, one with the school as its object (which is correct), one with the county (which is not in line with the expected object type of an educational institution). SDValidate is capable of identifying the wrong one out of the two and eliminates it, while keeping the correct object.
- Wrong links in Wikipedia itself. A typical example is the statement `dbpedia:Dallas_Kruse dbpedia-owl:instrument dbpedia:Rhodes .`
  This reflects the exact link given in the Wikipedia infobox, however, `dbpedia:Rhodes` is the concept which denotes the Greek island named *Rhodes*, while the correct Wikipedia article to be linked from the infobox would be *Rhodes_piano*.
- Wrong infobox key. One example is the statement `dbpedia:The_Great_Cat dbpedia-owl:occupation dbpedia:Neo-classical_metal .`
  Here, the infobox key "genre" should have been used instead of "occupation".
- Links with hashtags. One example is the statement `dbpedia:Michael_Sim dbpedia-owl:award dbpedia:PGA_Tour_of_Australasia .`
  This statement is extracted from an infobox object actually being `http://en.wikipedia.org/wiki/PGA_Tour_of_Australasia#Order_of_Merit_winners`, which refers to an award. During the extraction, the URL fragment is removed, which leads to a statement making less sense.

The source of the remaining identified errors is either unknown, or they represent false negatives, i.e., statements identified as wrong which are actually correct.

The distribution of errors across the different categories is depicted in Figure 9. The largest portion of errors is produced in cases where the infobox entry from which the statement is extracted contains a text expression which is not linked as a whole, but only a part of which is linked. This is in line with the findings by Zaveri et al. (2013), who report that the majority of accuracy problems in DBpedia goes back to objects being incorrectly or incompletely extracted. In particular, it is noteworthy that Zaveri et al. claim that detecting this type of error cannot be automatized, while SDValidate is a heuristic algorithm which automatically detects at least a cer-

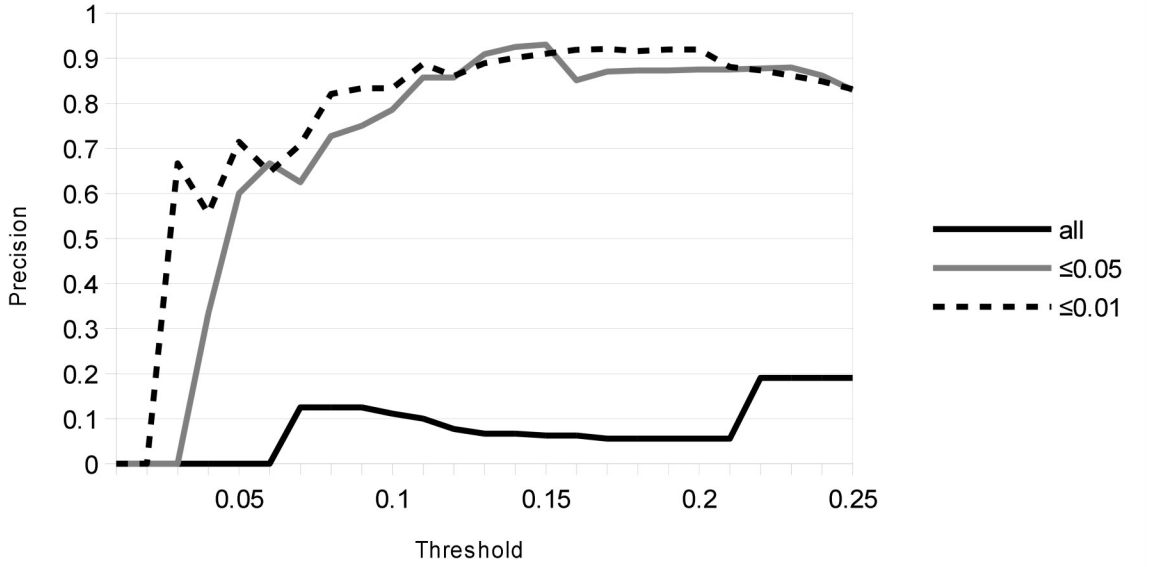


*Figure 7: Performance of SDValidate on the DBpedia dataset, using 10,000 random instances and ingoing relations only. The figure depicts the precision that is achieved using different thresholds. The three lines show the results for applying SDValidate to all statements in DBpedia, as well as to subsets obtained by filtering by significance level of 0.05 and 0.01.*

tain portion of such errors. The second major source of errors are wrong links in Wikipedia itself. Curiously, this source of errors is not in the list of common errors listed by Zaveri et al.

The NELL dataset does not provide provenance information on statement-level as facts are not extracted from a single source, but the result of patterns learned from different sources. This makes it hard to pinpoint the exact reason for a wrong statement. However, as discussed above, one frequent source of errors is the presence of homonymous resources, which are obviously hard to distinguish for NELL, and are hence often merged into one resource. Examples identified by SDValidate include *MS* being the abbreviation for *Microsoft* as well as for *multiple sclerosis*, the American football team *Atlanta Falcons* and the animal *falcon*, *concrete* as a building material and the Thomas Bernhard novel *Concrete*, etc. In particular the latter case (names of novels also denoting other things) is found quite frequently in the NELL dataset.

## Deployment in DBpedia

In our experiments on the DBpedia dataset, we found that filtering by a significance level of 0.05 and 0.01 yields comparable precision, but the number of statements that are validated is significantly different: at a significance level of 0.05, 563,657 statements are validated, at 0.01, it is only 144,043. Hence, we decided to apply filtering by significance at a level of 0.05, and used a

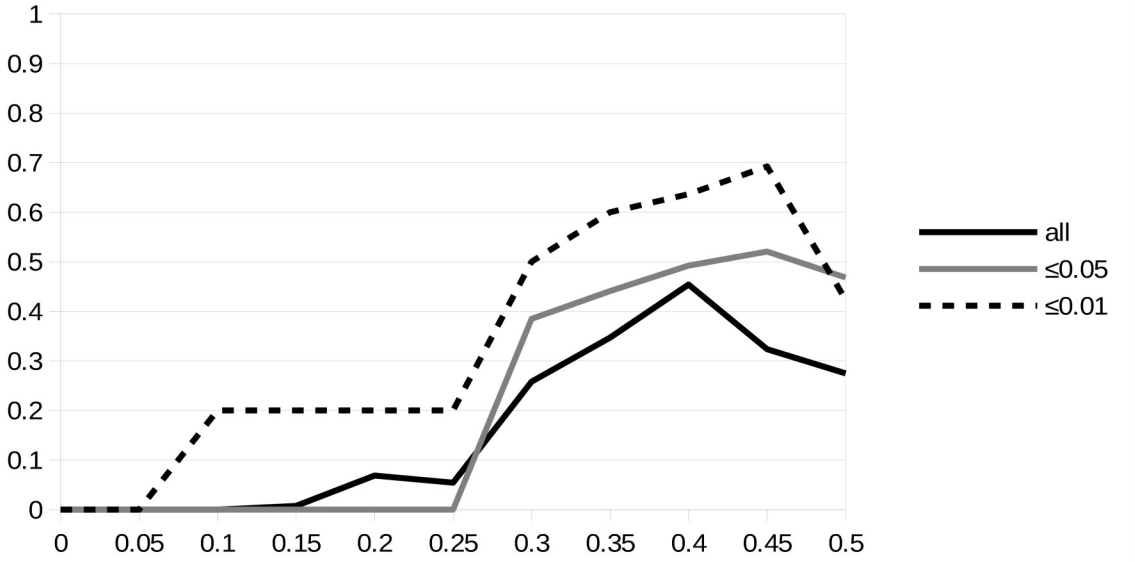


*Figure 8: Performance of SDValidate on the NELL dataset, using 10,000 random instances and ingoing relations only. The figure depicts the precision that is achieved using different thresholds. The three lines show the results for applying SDValidate to all statements in NELL, as well as to subsets obtained by filtering by significance level of 0.05 and 0.01.*

final threshold of 0.15, which was optimal for our sample w.r.t. maximizing precision, as discussed above.

When applying the algorithm to the DBpedia knowledge base, we aimed at minimizing the number of false negatives, i.e., the number of correct statements being removed from the knowledge base. To that end, we manually examined a sample of the output of SDValidate. Upon that examination, we found that only five different properties were responsible for 75% of the false negatives. These properties are: `dbpedia-owl:knownFor`, `dbpedia-owl:product`, `dbpedia-owl:nonFictionSubject`, `dbpedia-owl:programmeFormat`, and `skos-core:subject`. It is obvious that these are properties that are very unspecific about their object (for example, people, places, or events can be the non-fiction subject of a book, among many other classes of things). In other words, properties whose objects tend to take a large number of types are problematic, unlike properties whose objects usually only take a small number of types. To account for that difference and avoid such false negatives automatically, we added an additional filter using the Gini index of a property with respect to its types, i.e.,

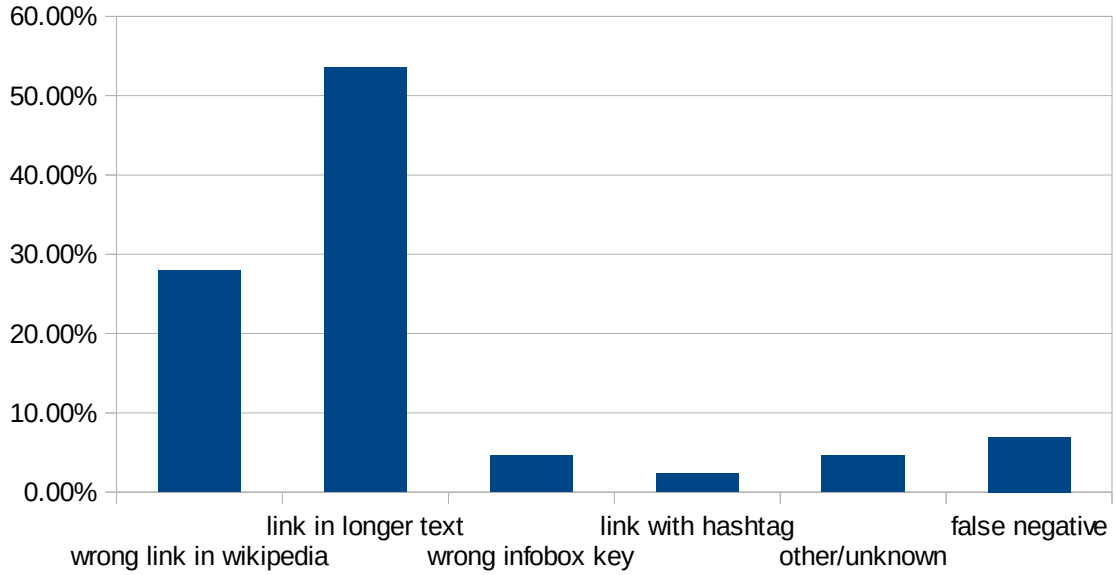


*Figure 9: Analysis of sources of errors found by SDValidate on DBpedia. The diagram shows the percentage of the different error sources.*

$$gini(p) := \sum_{typest} \left(P(t|p)\right)^2$$

Note that unlike the standard use of the Gini index, values larger than 1 are possible here, since the probabilities of types do not add up to 1 because an instance can have multiple types. Nevertheless, the index provides a good indicator about the "pureness" of a property.

From looking at the "suspicious" properties, we decided to eliminate all identified statements whose predicate has a gini index below 0.15. After that filtering step, 10,120 statements (i.e., roughly half of the statements identified by SDValidate) were removed from the mapping-based properties file[10], which means that approximately 0,1% of the statements are removed because it is likely that they are wrong.

## IMPLEMENTATION AND COMPLEXITY OF THE ALGORITHMS

Both SDType and SDValidate can be implemented in a relational database system (for our experiments, we have used MySQL). The algorithms start by importing two files containing the triples to validate, separated into type statements (*types*) and statements with non-literal objects which are not `rdf:type` statements (*relations*).

[10] For the DBpedia 3.9 release, we used a manual filtering instead of automatic filtering by Gini index, which in the end removed around 13,000 statements in total.

From these input tables, we subsequently compute the basic statistics, as well as the final tables containing the additional type statements (for SDType) and the identified wrong statements (for SDValidate). The following pseudocode illustrates the process:

```
// data preparation
input: table types (resource,type)
input: table relations (subject,predicate,object)

table type_count (type, count)
   ← pass over types, count occurrences of each type

table predicate_count(predicate, count)
   ← pass over relations, count occurrences of each predicate

table type_apriori_probability (type, probability)
    ← pass over type_count, divide each by number of resources

table resource_predicate_frequency (resource, predicate, frequency)
   ← pass over relations, count pairs of resource and predicate (both
subject and object)

table predicate_type_probability (predicate, type, probability)
   ← pass over types ⋈ resource_predicate_frequency ⋈
   predicate_count,
   count occurrences of type and relation, divided by occurences of the
   predicate

table predicate_weight (predicate, weight)
   ← pass over predicate_type_probability ⋈ type_apriori_probability,
      sum up squared difference of type probability given the predicate
      and apriori probability

// materialize types (SDType)
table inferred_types (resource, type, score)
   ← pass over resource_predicate_frequency
     ⋈ predicate_type_probability ⋈ predicate_weight,
    sum up count*probability*weight divided by sum(weight)
    for each resource

// find wrong statements (SDValidate)
table statements_to_check (subject, predicate, object)
   ← pass over relations ⋈ resource_predicate_frequency,
     collect all statements that have a predicate frequency below the
     significance level
```

```
table scored_statements (subject, predicate, object, score)
  ← pass over statements_to_check ⋈ types, statements_to_check
      ⋈ predicate_type_probability
     compute cosine similarity between actual and probable types for
      each resource
```

Given that T denotes the number of types, P denotes the number of properties, R denotes the number of resources, and S is the total number of statements, the sizes of the tables used in the algorithm have the following upper bounds:

| | |
|---|---|
| type_count | T |
| predicate_count | P |
| type_apriori_probability | T |
| resource_predicate_frequency | S |
| predicate_type_probability | PT |
| predicate_weight | P |
| inferred_types | RT |
| statements_to_check | S |
| scored_statements | S |
| types ⋈ resource_predicate_frequency ⋈ predicate_count | ST |
| type_predicate_probability ⋈ type_apriori_probability | PT |
| resource_predicate_frequency ⋈ predicate_type_probability ⋈ predicate_weight | SPT |
| relations ⋈ resource_predicate_frequency | $S^2$ |
| statements_to_check ⋈ types | ST |
| statements_to_check ⋈ predicate_type_probability | SPT |

Given these upper bounds, the largest tables to pass over for SDType have the maximum size SPT (since R<S), while SDValidate needs to pass over a table of size $S^2$. As there are no loops in the algorithm, these are also the upper bound for the algorithms' runtime and memory complexity.

Practically, the whole deployed process on the DBpedia dataset (including both SDType and SDValidate) ran in less than six hours on a 2.4 GHz linux machine with 2 GB Ram, with the data stored in a MySQL database, using appropriate index structures on the intermediate tables.

## RELATED WORK

We organize the review of related work in two subsections, which correspond to the tasks addressed by SDType and SDValidate: the first subsection compares approaches for predicting (missing) types, while the second subsection discusses approaches for detecting false statements. Since many of the related approaches have also been evaluated on the DBpedia dataset, a direct comparison is possible in those cases.

### Approaches for Type Prediction

Type prediction can be seen as a special case of inductive inference on the Semantic Web, where the goal is to infer all type statements for a given instance. The problems of inference on noisy data in the Semantic Web has been identified, e.g., by Polleres et al. (2010), and Ji et al. (2011). While general-purpose reasoning on noisy data is still actively researched, there have been solutions proposed for the specific problem of type inference for (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.

One of the first approaches to type classification in relational data has been discussed by Neville and Jensen (2000). 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 approach we examined in Paulheim (2012) assumes that for many instances, there are some, but not all types. Association rule mining is employed to find common co-occurence patterns between types, and apply them to the knowledge base. We have shown that this approach can lead to add approximately 3 additional types to an average instance at a precision of 85.6% on the DBpedia dataset with all types (including the YAGO types).

An approach close to the one presented in this paper is ProSWIP, which uses properties of resources to assign types. The authors evaluate their approach on a subset of the Billion Triples Challenge dataset, and report an F-measure of up to 0.39 for movies, 0.90 for music items, and 0.13 for books. However, since the evaluation set is different, the results cannot be directly compared.

Aprosio et al. (2013) introduce an approach 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 in DBpedia, the authors report a recall of 0.48, a precision of 0.91, and an F-measure of 0.63.

The *Tipalo* system (Gangemi et al., 2012) leverages the natural language descriptions of DBpedia resources to infer types, exploiting the fact that most abstracts in Wikipedia follow similar patterns (e.g., "Rammstein is a Neue Deutsche Härte band from Berlin, Germany"). Those descriptions are parsed and mapped to the WordNet and 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, evaluated against a manually created gold standard.

Giovanni et al. (2012) exploit types of resources derived from linked resources, where links between Wikipedia pages are used to find linked resources. As DBpedia only exploits links within Wikipedia infoboxes, this means that Giovanni et al. use more links than we do. 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 precision of 0.52, and hence an F-measure of 0.65, on DBpedia.

Sleeman and Finin (2013) try to predict the type of instances, given the attributes of that instance. They use a labeled training set for training an approach for type prediction, in contrast to SDType, which is directly working on the target data without the need for a labeled gold standard. They evaluate their approach on three example classes (Person, Place, and Organization) on Freebase, reporting an F-measure near 1.0 for places, while the F-measure for persons and organizations is around 0.6.

Pohl (2012) addresses a slightly different problem, i.e., the mapping DBpedia resources 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.

Oren et al. (2007) use a similar approach as ours, i.e., inspecting statistical distributions of properties and types, but on a slightly different problem setting: they try to predict possible properties for resources based on co-occurrence of properties. They report an F-measure of 0.85 at linear runtime complexity. However, the approach cannot be used for fully automatic quality enhancement, as it only predicts the existence of a property, but not the predicate's object.

The approaches discussed above, except for our approach published in Paulheim (2012), are using specific features of 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 (i.e., instances for which no Wikipedia page exists at all) only by using information from the ingoing properties.[11]

[11] *Red links* appear in Wikipedia if link is created in a Wiki page, which points to another Wiki page which does not (yet) exist, but which the author believes will be created in the future (upon creation, the red link will turn into a normal, functioning link – see http://en.wikipedia.org/wiki/Wikipedia:Red_link). If a red link is contained in a Wikipedia infobox, a DBpedia resource is created for that link, which has no corresponding page in Wikipedia.

## Approaches for Detecting Wrong Statements

The problem of automatically detecting errors in knowledge bases has been acknowledged to be hard. While ontology reasoning is capable of detecting conflicts in a knowledge base, this requires a rich ontology which allows for such detection, e.g., contains statements about the disjointness of classes. Since the DBpedia ontology does not have that level of expressiveness, it would have to be enriched first, e.g., by means of ontology learning (Völker and Niepert, 2011). Töpper et al. (2012) have evaluated the approach of first enriching the DBpedia ontology with additional domain and range restrictions, as well as class disjointness axioms, and then using the enhanced ontology for error detection. They report that they are able to identify around 60,000 inconsistent statements, however, they come to the conclusion that in most cases the identified statement itself is actually correct, whereas the ontology should be altered. Lehmann and Bühmann (2010) discuss a similar approach, but do not provide quantitative results.

Other approaches use external knowledge to validate statements, either from experts or from external data sources. Acosta et al. (2013) have discussed the use of crowd-sourcing, using platforms such as Amazon Mechanical Turk which pay users for micro-tasks, e.g., the validation of a statement. Furthermore, they used a custom platform which organized the validation of statements as a competion. Their evaluation concentrates on three error classes, i.e., wrong literal values, wrong literal datatypes, and wrong interlinks to other datasets. For the three tasks, they report a maximum precision (across both approaches) of 0.90, 0.83, and 0.94, respectively. Waitelonis et al. (2011) use games with a purpose to evaluate DBpedia and spot inconcistencies. They report that in 4,051 statements used in the game, 265 inconsistencies have been detected by users, 121 out of which were actually inconsistencies. This leads to a precision of only 0.46, which makes that approach only partially suitable for increasing data quality, at least without expert reviewing.

While the precision achieved by Acosta et al. (2013) is impressing, the problem with crowd-sourcing approaches is their scalability. The authors report that on Amazon Mechanical Turk, 1,073 statements could be validated within four days. This means that the whole DBpedia knowledge base would take more than 3,000 years to validate using that approach. Even the reduced number of statements evaluated by SDValidate after the pre-selection step, i.e., 563,657 statements, would still take almost six years.

Approaches based on statistical distributions have also been used on literal-valued, in particular on numeric properties. Outlier detection methods such as interquantile range and kernel density estimation can be used to find numeric values that do not fall into the typical range of a property (Chandula et al., 2009). In Wienand and Paulheim (2014), we use a fully automatic approach that combines clustering of resources by their type and various outlier detection methods, achieving a precision of 89%. In Paulheim (2014), we have shown that outlier detection can also be applied to identifying wrong *dataset interlinks*.

External knowledge is used, e.g., by *DeFacto* (Lehmann et al., 2012). The authors have build a pattern library of lexical forms for properties frequently used in DBpedia. Using those lexical patterns, DeFacto runs search engine requests for natural language representations of DBpedia statements. Their approach reaches a precision and an F-measure of 0.88. West et al. (2014) follow a similar approach, but automatically learn the search engine queries for each property.

When contrasting these approaches to SDValidate, we can observe that there are to the best of our knowledge no evaluated approaches that detect wrong statements (with resources, not literals in the object position) without using external knowledge or crowdsourcing. Furthermore, it is remarkable that the precision of our approach can keep up with crowd-sourcing approaches and DeFacto, both of which use external knowledge or social intelligence to solve the task.

## CONCLUSION AND FUTURE WORK

This article has discussed two algorithms for improving the quality of RDF knowledge bases, which rely on statistical distributions of properties and types: SDType assigns types to instances, and SDValidate assesses the correctness of statements. Both algorithms use purely the knowledge base itself, i.e., they neither rely on human interaction nor on external knowledge sources.

Evaluations on DBpedia and a Linked Data version of NELL have shown that both algorithms work with high accuracy and scale to large datasets. SDType has been shown to outperform all other existing approaches for type prediction in DBpedia, while there are only few approaches that are directly comparable to SDValidate. It is, however, noteworthy that the precision of SDValidate can keep up with approaches that use external knowledge from other sources or human intelligence to identify errors in DBpedia.

Both SDType and SDValidate have been applied for creating the DBpedia 3.9 release. With SDType, 3.4 million missing type statements have been added, and with SDValidate, 13,000 erroneous statements have been removed from the knowledge base.

So far, we have concentrated on the English DBpedia. While both approaches can be easily transferred to other languages as well, it would be particularly interesting to use information from multiple languages, both for computing more stable distributions, as well as for gathering additional evidence for predicted types and identified errors.

Our detailed analysis of SDValidate on the DBpedia dataset has shown that around one quarter of the identified wrong statements are due to wrong links in Wikipedia, e.g., linking to the island *Rhodes* instead of the *Rhodes piano*. Future extensions of our approach might even try to replace such statements with the most likely alternative, e.g., by searching for alternatives with a service such as DBpedia Lookup,[12] using the original object's label as input. In the above example, this search would yield both the island as well as the piano. Comparing the found objects' type vectors to the property's distribution, a replacement can be generated. With such an approach, we could not only *remove*, but also *repair* wrong statements. On the NELL dataset, we have observed that a main problem identified by SDValidate is homonymous resources wrongly fused into one resource. Here, it would be interesting to extend the approach in a way that it is capable of automatically detecting and splitting resources denoting homonymous resources.

Furthermore, SDValidate has pointed us at a number of cases where the DBpedia extraction framework has problems in extracting a correct object. While at the moment, we have identified those manually, clustering the identified wrong statements for automatically detecting errors in the DBpedia mapping framework would be an interesting research item for future work.

If statistical distributions are computed for two interlinked datasets, it would also be possible to use the mechanisms proposed in this paper for validating RDF links between datasets. For example, the analysis by Zaveri et al. (2013) has revealed that around 19% of the links from DB-

[12] http://lookup.dbpedia.org/

pedia to Freebase are incorrect. Computing distributions not only for DBpedia, but also for Freebase may help to reveal such interlinking errors.

## REFERENCES


Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann: Crowdsourcing Linked Data Quality Assessment. In: 12th International Semantic Web Conference, 2013.

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.

Isabelle Augenstein, Sebastian Padó, and Sebastian Rudolph: LODifier: Generating Linked Data from Unstructured Text. In: Extended Semantic Web Conference, 2012.

Christian Bizer and Richard Cyganiak: D2R Server – Publishing Relational Databases on the Semantic Web." Poster at the 5th International Semantic Web Conference, 2006.

Christian Bizer, Tom Heath, and Tim Berners-Lee: Linked Data – The Story So Far. In: International Journal on Semantic Web and Information Systems, 5(3):1–22, 2009a.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann: DBpedia - A crystallization point for the Web of Data. In: Web Semantics – Science Services and Agents on the World Wide Web, 7(3):154–165, 2009b.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell: Toward an Architecture for Never-Ending Language Learning. In: AAAI 2010.

Varun Chandola, Arindam Banerjee and Vipin Kumar: Anomaly Detection: A Survey. In: ACM Computing Surveys 41.3 (2009).

Arnab Dutta, Christian Meilicke, and Simone Paolo Ponzetto: A Probabilistic Approach for Integrating Heterogeneous Knowledge Sources. In: Extended Semantic Web Conference, 2014.

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.

Lise Getoor and Christopher P. Diehl. Link Mining: a Survey. In: ACM SIGKDD Explorations Newsletter, 7(2):3-12, 2005.

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.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich and Gerhard Weikum: YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. In: Artificial Intelligence, 194, 28-61, 2013.

Qiu Ji, Zhiqiang Gao, and Zhisheng Huang. Reasoning with noisy semantic data. In: The Semantic Web: Research and Applications (ESWC 2011).

Daniel Gerber and Axel-Cyrille Ngonga Ngomo: Extracting Multilingual Natural-Language Patterns for RDF Predicates. In: Knowledge Engineering and Knowledge Management, 2012.

Silviu Homoceanu, Philipp Wille, and Wolf-Tilo Balke: ProSWIP: Property-based Data Access for Semantic Web Interactive Programming. In:12th International Semantic Web Conference (ISWC 2013).

Jens Lehmann and Lorenz Bühmann: ORE - A Tool for Repairing and Enriching Knowledge Bases. In: Proceedings of the 9th International Semantic Web Conference (ISWC 2010).

Jens Lehmann, Daniel Gerber, Mohamed Morsey, Axel-Cyrille Ngonga Ngomo: DeFacto - Deep Fact Validation. In: International Semantic Web Conference (ISWC 2012).

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, Christian Bizer: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. In: Semantic Web Journal, 2014.

Cynthia Matuszek, John Cabral, Michael Witbrock, and John DeOliveira. An Introduction to the Syntax and Content of Cyc. In: AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and its Applications to Knowledge Representation and Question Answering, 2006.

Jennifer Neville and David Jensen. Iterative classification in relational data. In: AAAI-2000 Workshop on Learning Statistical Models from Relational Data.

Eyal Oren, Sebastian Gerke, and Stefan Decker. Simple algorithms for predicate suggestions using similarity and co-occurrence. In: European Semantic Web Conference (ESWC 2007).

Heiko Paulheim: Browsing linked open data with auto complete. In: Semantic Web Challenge, 2012.

Heiko Paulheim: Identifying Wrong Links between Datasets by Multi-dimensional Outlier Detection. In: International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM 2014).

Heiko Paulheim and Christian Bizer: Type Inference on Noisy RDF Data. In: International Semantic Web Conference (ISWC 2013).

Axel Polleres, Aidan Hogan, Andreas Harth, and Stefan Decker. Can we ever catch up with the web? In: Semantic Web Journal, 1(1,2):45-52, 2010.

Leo L. Pipino, Yang W. Lee, and Richard Y. Wang: Data Quality Assessment. In: Communications of the ACM, Vol. 45, No. 4, pp. 211-218, 2002.

Aleksander Pohl. Classifying the Wikipedia Articles in the OpenCyc Taxonomy. In Web of Linked Entities Workshop (WoLE 2012), 2012.

Cartic Ramakrishnan, Krys J. Kochut, and Amit P. Sheth: A Framework for Schema-driven Relationship Discovery from Unstructured Text. In: International Semantic Web Conference, 2006.

Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz. Pellet: A practical OWL-DL reasoner. In: Journal of Web Semantics, 5(2), 2007.

Jennifer Sleeman and Tim Finin: Type Prediction for Efficient Coreference Resolution in Heterogeneous Semantic Graphs. In: 7th IEEE International Conference on Semantic Computing, 2013.

Gerald Töpper, Magnus Knuth, and Harald Sack: DBpedia Ontology Enrichment for Inconsistency Detection. In: Proceedings of the 8th International Conference on Semantic Systems (I-SEMANTICS 2012).

Johanna Völker and Mathias Niepert. Statistical schema induction. In: The Semantic Web: Research and Applications (ESWC 2011).

Jörg Waitelonis, Nadine Ludwig, Magnus Knuth, and Harald Sack: WhoKnows? - Evaluating Linked Data Heuristics with a Quiz that Cleans Up DBpedia. In: International Journal of Interactive Technology and Smart Education (ITSE), 2011.

Richard Y. Wang, Diane M. Strong, and Lisa Marie Guarascio: Beyond accuracy: What data quality means to data consumers. In: Journal of Management Information Systems, Vol. 12 Number 4, pp. 5-33, 1996.

Gabriel Weaver, Barbara Strickland, and Gregory Crane. Quantifying the Accuracy of Relational Statements in Wikipedia: a Methodology. In: 6th Joint Conference on Digital Libraries (JCDL 2006).

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, Dekang Lin: Knowledge Base Completion via Search-Based Question Answering. In: International World Wide Web Conference (WWW), 2014.

Dominik Wienand and Heiko Paulheim: Detecting Incorrect Numerical Data in DBpedia. In: Extended Semantic Web Conference, 2014

Amrapali Zaveri, Dimitris Kontokostas, Mohamed A. Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann: User-driven Quality Evaluation of DBpedia. In: 9th International Conference on Semantic Systems (I-SEMANTICS 2013)

Antoine Zimmermann, Christophe Gravier, Julien Subercaze, and Quentin Cruzille: Nell2RDF: Read the Web, and turn it into RDF. In: 2nd International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data, 2013.