Preprint, final version available at http://dx.doi.org/10.1016/j.websem.2016.05.001

LHD 2.0: A Text Mining Approach to Typing Entities In Knowledge Graphs

Tomáš Kliegr[a,b,1], Ondřej Zamazal[a,1]

[a] *Department of Information and Knowledge Engineering, Faculty of Informatics and Statistics, University of Economics, Prague, nám. W Churchilla 4, 13067, Prague, Czech Republic*
[b] *Multimedia and Vision Research Group, Queen Mary, University of London, 327 Mile End Road, London E1 4NS, United Kingdom*

Abstract

The type of the entity being described is one of the key pieces of information in linked data knowledge graphs. In this article, we introduce a novel technique for type inference that extracts types from the free text description of the entity combining lexico-syntactic pattern analysis with supervised classification. For lexico-syntactic (Hearst) pattern-based extraction we use our previously published Linked Hypernyms Dataset Framework. Its output is mapped to the DBpedia Ontology with exact string matching complemented with a novel co-occurrence-based algorithm STI. This algorithm maps classes appearing in one knowledge graph to a different set of classes appearing in another knowledge graph provided that the two graphs contain common set of typed instances. The supervised results are obtained from a hierarchy of Support Vector Machines classifiers (hSVM) trained on the bag-of-words representation of short abstracts and categories of Wikipedia articles. The results of both approaches are probabilistically fused. For evaluation we created a gold-standard dataset covering over 2,000 DBpedia entities using a commercial crowdsourcing service. The hierarchical precision of our hSVM and STI approaches is comparable to SDType, the current state-of-the-art type inference algorithm, while the set of applicable instances is largely complementary to SDType as our algorithms do not require semantic properties in the knowledge graph to type an instance. The paper also provides a comprehensive evaluation of type assignment in DBpedia in terms of hierarchical precision, recall and exact match with the gold standard. Dataset generated by a version of the presented approach is included in DBpedia 2015.

Keywords: type inference, Support Vector Machines, entity classification, DBpedia

1. Introduction

One of the most important pieces of information in linked data knowledge graphs is the *type* of the entities described. The next generation linked open data enabled applications, such as entity classification systems, require complete, accurate and specific type information. However, many entities in the most commonly used semantic knowledge graphs miss a type. For example, DBpedia 3.9 is estimated to have at least 2.7 million missing types with the percentage of entities without any type being estimated at 20% [1]. Type inference has thus received increased attention in the recent years, with the approaches proposed taking either of the two principal paths: statistical processing of information that is already present in the knowledge graph, or extraction of additional types from the free text. In this article we introduce a novel technique for type inference which combines lexico-syntactic analysis of the free text and machine learning. This combined approach can complete types for about 70% of Wikipedia articles

Email addresses: tomas.kliegr@vse.cz (Tomáš Kliegr), ondrej.zamazal@vse.cz (Ondřej Zamazal)

[1] Both authors contributed equally.

The final version of this article is available at http://dx.doi.org/10.1016/j.websem.2016.05.001

without a type in DBpedia.

Our previously published Linked Hypernyms Dataset (LHD) framework [2] extracts types from the first sentence of Wikipedia articles using lexico-syntactic patterns. In this work we extend it with Statistical Type Inference (STI) which helps to map LHD results to the DBpedia Ontology used by the native DBpedia solution. STI algorithm is a generic co-occurrence-based algorithm for mapping classes appearing in one knowledge graph to a different set of classes appearing in another knowledge graph provided that the two knowledge graphs contain common set of instances. In our setup, our target knowledge graph is DBpedia, and the source knowledge graph is LHD.

There are many articles for which lexico-syntactic patterns fail to extract any type. To address this, we employ Support Vector Machines (SVMs) trained on the bag-of-words representation of short abstracts and categories of Wikipedia articles. This supervised machine learning approach gives us a second set of entity type assignments.

In order to exploit the complementary character of the co-occurrence based STI algorithm and the supervised SVM models, we implement an ontology-aware fusion approach based on the multiplicative scoring rule proposed for hierarchical SVM classification. The hSVM algorithm can also be used separately as a language independent way to assign types since it uses abstract or categories as input feature set and it does not require language-specific preprocessing.

We validate our work on DBpedia 2014 [3], one of the most widely used Wikipedia-based knowledge graphs, the algorithmic approach is applicable also to the YAGO knowledge base [4], as well as to other semantic resources which contain instances (entities) that are a) classified according to a taxonomy, and b) described with a free text definition.

The evaluation of our algorithms is performed on DBpedia using a gold standard dataset comprising more than 2,000 entities annotated with types from the DBpedia ontology using a crowdsourcing service.

The dataset generated with an earlier version of our approach is part of the DBpedia 2015-04 release as *Inferred Types LHD* dataset.

Parts of the work presented in this article have been published within the conference paper "Towards Linked Hypernyms Dataset 2.0: complementing DBpedia with hypernym discovery and statistical type inference (Kliegr and Zamazal, 2014)" [5]. This article extends the conference paper by introducing the hierarchical SVM approach and by performing extensive evaluation on the contributed gold standard dataset allowing the community to track progress in accuracy and coverage of entity typing and extraction tools. Also, the review of related work was substantially expanded.

The article is organized as follows. Section 2 gives an overview of related work, focusing on approaches for inference of entity types in DBpedia. Section 3 gives an overview of our approach. Section 4 describes how our LHD framework extracts types from the first sentence of Wikipedia articles and disambiguates them to DBpedia concepts. Section 5 presents the proposed algorithm for statistical type inference. Section 6 introduces the hierarchical support vector machines classifier. Section 7 describes the fusion algorithm. Section 8 presents the evaluation on the crowdsourced content and comparison with the state-of-the-art SDType algorithm and the DBpedia infobox-based extraction framework. The conclusions provide a summary of the results and an outlook for future work.

2. Related work

Completing missing types based on statistical processing of the information already present in the knowledge graph is in current research approached from several directions: a) RDFS reasoning, b) obtaining types through the analysis of the unstructured content with patterns, c) machine learning models trained on labeled data, d) unsupervised models that perform inference from statistical distributions of types, instances and the relations between them.

The four approaches listed above are covered in Subsection 2.1-2.4. Subsection 2.5 covers the comparison of our STI/hSVM with SDType, which is a state-of-the-art unsupervised algorithm actually used for type inference in DBpedia 3.9 and DBpedia 2014. Subsection 2.6 motivates our choice of hSVM as a suitable machine learning classifier. Since we perceive the crowdsourced gold standard as an important element of our contribution, Subsection 2.7 reviews methods and resources for evaluation of algorithms that assign types to DBpedia entities. Table 1 gives an overview of selected related algorithms in terms of the methods and input features used and provides a comparison with our solution described in this article. A recent broader

overview of approaches for knowledge graph refinement is present in [6].

2.1. RDFS Reasoning

The standard approach to the inference of new types in semantic web knowledge graphs is RDFS reasoning. There are two general requirements enabling RDFS reasoning. First, these graphs need to have *domain* and *range* for properties specified and, second, they need to contain the corresponding *RDF facts* employing the defined properties. However, since according to common ontology design best practices (e.g. in Noy et al. [11]), domain and range should be defined in a rather general way, the inferred types tend not to be very specific. Also, *type propagation* goes upward along the taxonomy as a result of interaction of the subsumption knowledge from the ontology with the RDF facts from a dataset. Hence, RDFS reasoning usually cannot infer a specific type (i.e. type low in the hierarchy).

Furthermore, it is well known that RDFS reasoning approach will not correctly work for problems where the knowledge graph contains false statements (which is the case for DBpedia), since the errors are amplified in the reasoning process. Additional discussion on unsuitability of reasoners for type inference in DBpedia has been presented by Paulheim and Bizer in [8].

2.2. Pattern-based analysis of unstructured content

Major semantic knowledge graphs DBpedia and YAGO are populated from the *semistructured data* in Wikipedia – infoboxes and article categories using extraction framework that primarily relies on hand-crafted patterns. Approaches that extract types from the *free text* of Wikipedia articles can be used to assign types to articles for which the semistructured data are either not available, or the extraction for some reason failed.

The analysis of the unstructured (free text) content also often involves hand-crafted patterns. Tipalo, presented by Gangemi et al. in [7], covers the complete process of generating types for DBpedia entities from the free text of Wikipedia articles using a set of heuristics based on graph patterns. The algorithm starts with identifying the first sentence in the abstract which contains the definition of the entity. In case a coreference is detected, a concatenation of two sentences from the article abstract is returned. The resulting natural language fragment is deep parsed for entity definitions.

Our STI component uses as input types that were extracted from the free text with lexico-syntactic patterns with the Linked Hypernyms Dataset extraction framework presented in [12]. This framework proceeds similarly with Tipalo in that it extracts the hypernym directly from the POS-tagged first sentence and then links it to a DBpedia entity.

The accuracy of LHD matches the results for Tipalo algorithm – as reported by its authors in [7] – for the type selection subtask (0.93 precision and 0.90 recall). A detailed comparison between LHD and Tipalo is presented in [2] as well as a more extensive literature review on pattern-based extraction.

A conceptual disadvantage of pattern-based approaches is that they require relatively complex natural language processing pipeline, which is costly to adapt for a particular language. In contrast, the hSVM approach that we introduce in this article has essentially no language-specific dependencies, apart from basic tokenization, which makes its portability to another language comparatively straightforward.

2.3. Supervised methods

One of the first supervised approaches was, according to Paulheim and Bizer [1], an iterative algorithm proposed in a relational data context by Neville and Jensen in [13]. The training instances are described by attributes derived from relations of the instance (object) to other instances (objects). Additionally, the high confidence inferred statements are inserted into the data and used in the subsequent inference process, which allows to define attributes that are dependent on the result of classification in earlier iterations.

In the experiments presented in the original paper the inferred property was the type (companies were classified by industry). The relations considered included *subsidiary*, *owner* and *percentage owned for given owner*. Example attributes included *the number of subsidiaries* and *whether the company is linked to more than one chemical company through its insider owners*. Interestingly, for a given instance the value of the latter attribute can change as the algorithm progresses through the iterations.

To the best of our knowledge, the first supervised type inference algorithm applied directly in the semantic web context to assign type was described by Sleeman and Finin in [14]. This approach uses information gain as a feature selection algorithm and

Table 1: Overview of related algorithms and components of our solution (simplified)

algorithm	method	input features
related algorithms		
Tipalo [7]	linguistic parsing	first two sentences of Wikipedia articles
SDtype [8]	co-occurrence analysis	ingoing properties in DBpedia
TRank [9]	supervised machine learning (best – decision tree)	schema and instance relations in DBpedia and YAGO
"Autocomplete" [10]	co-occurrence analysis	existing type assignments in DBpedia
components of our algorithmic solution		
LHD [2]	linguistic parsing	first sentence in Wikipedia articles
STI	co-occurrence analysis	type assignments in DBpedia and LHD
hSVM	supervised ml. (Support Vector Machines)	Wikipedia article abstract and categories

Support Vector Machines (SVM) for classification. The reported F-measure is between 24.9% to 92.9%.

In addition to other differences to our approach such as a different input feature set, the two algorithms presented above take the flattened approach to classification, as they do not consider the taxonomical structure of target labels: each target label is a separate class. In contrast, our hSVM algorithm takes the hierarchical approach to classification, which has been shown by Liu et al. in [15], to have a superior performance when large taxonomies are involved.

Another type of supervised approach is exemplified by the TRank system [9], which ranks possible entity types given an entity and context. The TRank authors evaluated several type-hierarchy and graph-based approaches that exploit both schema and instance relations. This work is not directly comparable to ours, because the aim of TRank is to select *type for given entity mention in a longer context* (sentence, paragraph, three paragraphs), while we aim to assign types for already *disambiguated articles* describing the entity. What is particularly relevant to our work is the evaluation methodology, as the collection of TRank algorithms was similarly to our work evaluated with crowdsourcing.

2.4. Unsupervised methods

Recently, several unsupervised machine learning algorithms for type inference emerged. Paulheim [10] describes the use of association rule mining to discover missing types for a specific entity. To improve scalability, a lazy association rule algorithm is used to learn only rules that are relevant for the types associated with the specific entity. The confidence value associated by the apriori algorithm with a rule is used as type confidence. If multiple rules predict the same type, their confidence scores are aggregated.

This algorithm bears some resemblance to the STI algorithm that we proposed in [5] (also covered in Section 5), since both algorithms exploit the occurrence of types. The association rule approach is more advanced in that if the entity has multiple types, all of them can potentially contribute to the type prediction.

The STI algorithm generates a universally applicable mapping from one type to a set of types, each associated with a confidence score. The final output of the algorithm is one type which is a compromise between specificity and reliability. The advantage of STI is thus speed, since the algorithm tries to infer the mapping for the relatively small number of types (such as `dbpedia:Playwright`), rather than individually processing all entities. Since the algorithm can also be applied to instances without any type previously assigned, STI can be expected to cover wider range of untyped entities than the association rule learning approach.

SDType, covered in detail in the next subsection, is a state-of-the-art algorithm for type inference proposed by Paulheim and Bizer [8], which as its authors assert provides superior results in terms of

F-measure compared to all the earlier approaches. The results of the SDType algorithm are also included in the official release of English DBpedia as the *Heuristics* dataset.

2.5. SDType algorithm

The SDType algorithm assigns types based on ingoing *properties* of the object. The properties are readily available in DBpedia as they have been extracted from the article infoboxes.

For each relation p (e.g. `dbo:location`)[2] the algorithm computes the conditional probability that a specific entity x is of certain type if x appears as a *subject* of the relation p. Likewise, a dual conditional probability is computed for x as the object of the same relation. Additionally, each relation p is assigned a weight, which reflects the discriminative power of the property.

SDType authors consider as untypeable e.g. lists or disambiguation articles. To limit the number of false statements that would be generated if these entities are reassigned with types, the initial step of SDType is to determine whether the entity is typeable using a machine learning classifier. The authors report that 5.5% of entities was found as not typeable. Our LHD generation process excludes entities listed in the DBpedia *disambiguations* dataset, which also corresponds to roughly 5.6% of entities for English DBpedia 2014.

Using the probability distributions associated with properties attached to an entity, the SDType algorithm outputs a confidence score for each entity-type pair. A predefined cutoff threshold balances the number of inferred types and their quality.

SDType assigns multiple types per entity. A higher confidence threshold assigns more types at lower precision. The self-reported precision at a confidence threshold producing on average 3.1 types is 0.99 (0.95 confidence at 4.8 types). Inspection of SDType results shows that while multiple types are assigned to a given entity, these are, in our observation, typically composed of a specific type and its supertypes. STI/hSVM assigns only the most specific type (cf. Example 1 and 2).

[2] `dbo` refers to the DBpedia ontology namespace `http://dbpedia.org/ontology/`

Example 1.
`dbpedia:Triple_Stamp_Records` is assigned types: `dbo:RecordLabel`, `dbo:Company`, `dbo:Organisation` and `owl:Thing` by SDType.[a] The STI/hSVM algorithm assigns a single type `dbo:RecordLabel`.

[a] DBpedia 3.9 `instance_types_heuristic_en.nt` file

Example 2.
`dbpedia:Terry_Sejnowski` is assigned types: `dbo:Person`, `dbo:Agent` and `owl:Thing` by SDType. The STI/hSVM algorithm assigns a single type `dbo:Scientist`.

It should be noted that SDType has the advantage that it can generate types also for entities which are derived from Wikipedia red links. This is impossible with both STI and hSVM algorithms, which require that the article contains a short abstract. However, if an article is not referenced from infobox of another article then it cannot be processed by SDType. For STI/hSVM this is not an obstacle.

It can thus be concluded that both SDType and STI/hSVM approaches are largely complementary both what concerns the algorithmic techniques used and the set of applicable untyped entities. Subsection 8.5 presents a comparison of SDType and STI/hSVM in terms of accuracy on a crowdsourced gold standard dataset.

2.6. Text categorization with SVM

In order to enhance type assignment provided by the STI algorithm, we introduce a supervised model trained on the bag-of-words representation of article content. In this way, we effectively cast the problem of assigning a type to an entity as a text categorization task. The entity-type assignments already present in DBpedia serve as the training data.

From the range of applicable machine learning algorithms, we opted for Support Vector Machines (SVMs) [16]. SVMs have been found to be more accurate than other standard machine-learning algorithms such as Naive Bayes, neural networks and the Rocchio classifier on the text categorization task as reported in [17]. Experimental results presented within our evaluation (in Subsection 8.7) confirm the superior performance of lin-

ear SVMs over other common classification algorithms in the flat text categorization task on our data. The SVM classifier is particularly suitable as it is scalable and has been previously successfully adapted to handle tasks involving large web taxonomies [17]. We adapt the *hierarchical SVM* approach (hSVM), where a separate classifier is built for all non-terminal leaves in the class hierarchy.

The complexity of flat SVMs is proportional to the number of target classes as reported in [15]. With SVMs in a hierarchical setup, there are several options. The *sequential Boolean rule* [17] or Pachinko-machine search [15] has typically a significant performance benefit for the testing phase, since for a given test instance an SVM model for class "c" is used only if its parent category classifies the test instance to class "c". Another approach is the *multiplicative scoring rule* [17], which applies all SVM models and then combines their resulting models by multiplying the probabilities obtained by classifiers on individual levels.

The computationally efficient sequential Boolean rule was found to perform equally well as the multiplicative scoring rule and better than a flat SVM as reported in [17]. The way of merging the outputs of classification models on the individual layers is a major design choice for hierarchical classification algorithms. Since computational complexity is not a major design constraint for our use case, we opted for multiplicative scoring rule as it provides structurally more convenient output for fusion with our other approach, STI.

2.7. Evaluation of type assignment

An important part of our contribution is the evaluation of the accuracy of the inferred types and the comparison with the average accuracy in the original knowledge graph. Two fundamental approaches to checking the accuracy of the inferred types were given by Gangemi et al. in [7]: *gold standard* and *type checking*.

In the gold standard approach, one needs to create a dataset assigning each entity identifier (DBpedia URI) with one or more type URIs. Typically, several annotators participate on the design of the dataset. The advantage of this approach is that the resulting dataset is reusable as long as the system which is evaluated is able to assign types to the same set of entities. The disadvantage is that this evaluation scheme is not straightforward to apply. Requiring exact match between the assigned type and the gold standard implies that if the assigned type is more general than the gold standard (e.g. footballer vs. midfielder) then the assignment is considered as incorrect.

In the type checking approach, human users evaluate the accuracy of the types. In the Tipalo evaluation a three-value scale was available: *yes*, *maybe*, *no*. Similar evaluation scheme was also employed for YAGO [4] and LHD [2].

The type checking evaluation scheme is not reusable and potentially difficult to reproduce. The evaluation, unless performed in an environment controlled by a third-party, may be difficult to repeat. It is common that the human evaluators are students or postdocs from the same department as are the authors of the algorithm that the evaluation is intended to support. The human evaluators may thus be under implicit pressure to judge more types as relevant than they would do under other circumstance. A second problem with this scheme is that it does not express how far the type assigned by the system is from the most specific type available in the reference ontology. For example, if the system assigns type "Person" to Diego Maradona it is counted as correct to the same degree as if the assignment is "Footballer".

In this article, we present a freely available gold standard dataset that can be used for evaluation of knowledge graphs that use types mappable to the DBpedia 2014 ontology. This gold standard dataset consists of over 2,000 entities with a type. The annotation process was performed using a third-party operated crowd-sourcing tool with a built-in interface for assignment of categories from a taxonomy. There was no direct contact between the authors and the annotators (three or four per entity-type assignment). The detection of under-performing annotators was handled automatically by the crowdsourcing tool. The design of the gold standard dataset was thus completely decoupled from the evaluation of the algorithm. To compare with, the gold standard used in the Tipalo tool was created for 100 entities and using annotation tool designed by the authors, the annotators were four senior researchers and six PhD students in the area of knowledge engineering.

Another broader evaluation setup that aimed at assessing the quality of data in DBpedia using crowdsourcing is presented by Zaveri et al. in [18]. This paper describes a methodology and a software tool for detecting errors in DBpedia. The authors identified 17 data quality problem types. The annotators evaluated in total 521 resources. While

this research pioneers the use of crowdsourcing for evaluating DBpedia triples, it does not specifically report on the *rdf:type* relation, which is the focus of this article.

A very recent survey that scopes evaluation of type assignment is presented in [6].

3. Overview of our approach

Our algorithmic solution to type inference consists of several components. The *Linked Hypernyms Dataset* [2] is used to extract types with lexico-syntactic patterns from the first sentence of Wikipedia articles. Part of the types are mapped to DBpedia ontology using reliable exact string matching. The remaining types are mapped using our co-occurrence based *Statistical Type Inference* algorithm. STI is a novel approach for mapping classes appearing in one knowledge graph to a different set of classes appearing in another knowledge graph provided that the two knowledge graphs contain common set of instances.

A parallel path to obtaining types for an entity is a *supervised machine learning approach* with Support Vector Machines (SVMs). Entities with already assigned types in DBpedia are used as a training set and the text of the abstract and the list of article categories are used as input features.

In order to fuse the outputs of all three models (STI, SVMs on abstract, SVMs on categories), we perform early fusion by aggregating (averaging) the individual probability distributions using the *linear opinion pool* [19, Chapter 9]. After that we combine the (already aggregated) distributions for individual classes in the class hierarchy. For this, we use either the *Multiplicative Scoring Rule* (MSR) designed for combining results of SVM models in a hierarchical setting, or a variant of the algorithm called *Additive Scoring Rule* (ASR).

Our approach consists of the following succession of steps:

1. Extracting types from free text with lexico-syntactic patterns using the LHD framework, resulting types are DBpedia resource.
2. Mapping types to DBpedia ontology with exact string matching (LHD Core).
3. Mapping remaining types with Statistical Type Inference (STI), the result for each input type (in the DBpedia resource namespace) is a probability distribution over DBpedia Ontology classes.

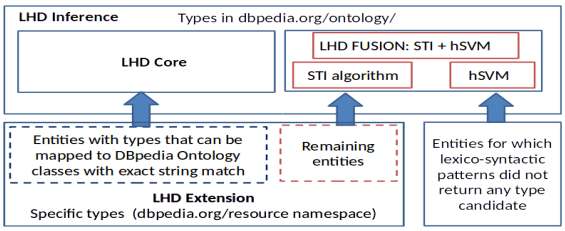

Figure 1: Partitions of the Linked Hypernyms Dataset

4. Training SVM models for a subset of classes in the DBpedia ontology.
5. Applying SVM models to obtain prediction for given entity, the output for a given entity is a probability distribution over a subset of DBpedia Ontology classes.
6. The probability distributions output by the SVM models and STI are aggregated using linear opinion pool.
7. The aggregated probability distribution is processed with respect to the DBpedia ontology in order to make reliable choice of a specific type.
8. The results of LHD Core (step 2) and SVM and STI models are combined to create the final dataset.

It should be emphasized that most of the steps above correspond to individual components, which can also be used independently. Steps 1-2 are performed by the Linked Hypernyms Dataset Framework described in Section 4. Step 3, the STI algorithm, is covered by Section 5. Steps 4-5 training and applying SVM models are covered in Section 6. Finally, steps 6-7 model fusion and final type selection are described in Section 7.

4. Linked Hypernyms Dataset

The Linked Hypernyms Dataset (LHD), introduced by Kliegr in [2], associates DBpedia entities (corresponding to Wikipedia articles) with a type which is obtained by parsing the first sentences of the respective Wikipedia article. The type is initially a plain text string, which is further disambiguated to a DBpedia entity creating a "linked hypernym". Figure 1 shows that the dataset is partitioned into several subsets.

The *Extension* dataset contains types in the `dbpedia.org/resource` namespace. This provides the highest precision types, but also the least semantic interoperability.

The types of about 50% of entities (for English, less for other languages) can be mapped to a DBpedia ontology type using a simple string matching algorithm, constituting the *Core* dataset. An overview of LHD Core in terms of size and accuracy is given in Table 2.

The entities with types extracted by the LHD framework but not mapped to the DBpedia ontology are used as input for the STI algorithm introduced in this paper. The remaining entities, for which the lexico-syntactic pattern extraction did not succeed, can be processed only with the hSVM approach, also introduced in this paper.

The *Inference* (Inferred types) dataset is published as a merge of all our approaches.

The remainder of this section briefly describes the individual steps of the LHD extraction framework: hypernym discovery, linking and the string matching approach leading to the *Core* dataset.

4.1. Hypernym Discovery

The Wikipedia manual of style [20] asserts that the page title should be the subject of the first sentence, and that it should tell the nonspecialist reader what, or who, the subject is. If the first sentence complies with these and other stated requirements, its structure can take only a limited number of forms, allowing a small number of hand-crafted patterns to cover most variations.[3]

Also, according to the Wikipedia Manual of Style "emphasis given to material should reflect its relative importance to the subject". Our decision to give preference to the first hypernym is based on the assumption that editors typically implement this clause by ordering hypernyms (e.g. occupations of a person) in the first sentence according to importance, starting with the most important one.

Our extraction framework exploits this regularity in the first sentence of Wikipedia articles. The framework is implemented on top of GATE.[4] The core of the system is a JAPE transducer (a GATE component) which applies lexico-syntactic patterns encoded as grammar in the JAPE language on the first sentence of Wikipedia articles.

The extraction grammars require that the input text is tokenized and assigned part-of-speech (POS) tags. For English, the framework relies on the ANNIE POS Tagger, available in GATE, for German and Dutch on TreeTagger.[5] Extraction grammars were hand-crafted using a development set of 600 manually annotated articles per language. The process of designing the grammars is described in detail in [2].

Example 3.
An example input for this phase is the first sentence of Wikipedia article on Václav Havel: *Havel was a Czech playwright, essayist, poet, dissident and politician.* The output is the word "playwright", the first hypernym in the sentence. The current version of the grammar outputs the head noun as the hypernym, not the complete noun chunk. Favouring head noun improves reliability as argued in [2].

The output of the hypernym discovery phase is provided as a separate dataset providing plain text, not disambiguated hypernyms. The accuracy for this dataset (denoted as "plain") is reported in Table 2.

4.2. Linking Hypernyms to DBpedia Instances

Once the hypernym is extracted from the article, it is disambiguated to a DBpedia identifier. The disambiguation algorithm relies on the Wikipedia Search API to resolve the string to a Wikipedia article.

Example 4.
Picking up on the Václav Havel example, the word "playwright" is used as a query, which returns the Wikipedia article `http://en.wikipedia.org/wiki/Playwright`. This is then translated to the DBpedia URI `http://dbpedia.org/resource/Playwright`.

Even if this disambiguation approach is simple, it is effective as confirmed both by our evaluation (Table 2) and by the recent results of the NIST TAC 2013 *English Entity Linking Evaluation* task, where it performed at median F1 measure (overall) [22].

[3] In [21] we studied whether article popularity could have an effect on the adherence to the Wikipedia manual of style, and in turn to the extractability of hypernyms from the first sentence. There was some evidence as to that may be the case, but due to the small size of the sample the results were inconclusive.

[4] `http://gate.ac.uk`

[5] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

Table 2: LHD Statistics. The `dbo` column indicates the portion of entities in LHD with type from the DBpedia ontology namespace, the rest is in the `dbpedia` namespace. The size is in thousands for the 3.9 dataset release and the accuracy was computed on the 3.8 release as reported by Kliegr in [2].

language	linked (total)	linked `dbo`	Acc linked	Acc plain
German	893k	199k	0.773	0.948
English	3,013k	1,136k	0.857	0.951
Dutch	834k	305k	0.884	0.933

4.3. Alignment with the DBpedia Ontology

While formally the output of the linking phase is already a Linked Open Data (LOD) identifier, the fact that the type is in the `http://dbpedia.org/resource/` namespace (further referenced by prefix `dbpedia`) is not ideal. Concepts from this namespace are typically entities, while this term is used as a type within LHD (cf. Example 5).

Example 5.
Entity Václav Havel has type `http://dbpedia.org/resource/Playwright` in LHD Extension. This entity is not present in LHD Core, because there is no Playwright class in the used DBpedia Ontology version. STI assigns this entity with additional type `http://dbpedia.org/ontology/Writer`.

DBpedia already contains a predefined set of types within the DBpedia ontology namespace `http://dbpedia.org/ontology/` (further abbreviated as `dbo`) such as `dbo:Person` or `dbo:Work`. The focus of the alignment phase is to map the original type, which is in the `dbpedia` namespace, to the `dbo` namespace.

The mappings are generated using a string matching algorithm, which requires total match in concept name (`dbpedia:Person` → `dbo:Person`). For these *exact match* mappings, only the `dbo:` type is output by the generation process.

This simple approach provides a mapping to the DBpedia ontology for a large number of entities across all three supported languages. However, in relative terms, this is less than 50% for each language as shown in Table 2, the types for almost all the remaining entities are mapped with the STI algorithm covered in the next section.

A more detailed description of the LHD framework as well as additional size and evaluation metrics are presented in [2].

5. Statistical Type Inference (STI)

The STI algorithm is a generic co-occurrence-based algorithm for mapping classes appearing in one knowledge graph to a different set of classes appearing in another knowledge graph provided that the two knowledge graphs contain common set of instances.

The algorithm thus works with two knowledge graphs, a primary knowledge graph $\mathcal{KG}$ associated with an ontology $\mathcal{O}_{\mathcal{KG}}$, and a knowledge graph $\mathcal{KG}_{map}$ that holds entity-type assignments that we desire to map to classes in $\mathcal{O}_{\mathcal{KG}}$. Both knowledge graphs hold entity-type assignments.

STI is based on a simple co-occurrence principle. First, for a specific input type $type_{map} \in \mathcal{KG}_{map}$ it finds the distribution of types that are assigned in $\mathcal{KG}$ to the same entities as $type_{map}$ is in $\mathcal{KG}_{map}$. The problem addressed is that the most frequently co-occurring types are very generic and thus it is necessary to identify out of the pool of the co-occurring types (classes from $\mathcal{O}_{\mathcal{KG}}$) those providing the best compromise between specificity and correctness.

The approach comprises two successive algorithms. The *Candidate generation* algorithm generates a set of candidate $\mathcal{O}_{\mathcal{KG}}$ types for $type_{map}$. The *Candidate pruning and selection* algorithm then performs removal of types for which a more specific one exists while maintaining reasonable trade off with correctness. From the types surviving the pruning, the type with the highest number of supporting entities is selected. A detailed description of the two algorithms follows.

Candidate generation (Algorithm 1) first identifies the set E that contains entities which have as

a type in $\mathcal{KG}_{map}$ the type $type_{map}$ that we desire to map to ontology $\mathcal{O}_{\mathcal{KG}}$. Algorithm output is the list of distinct $\mathcal{O}_{\mathcal{KG}}$ types which the entities in E have along with the number of occurrences of each type stored as *supp*. Example 6 illustrates this algorithm.

Example 6.
For the entity Václav Havel, the set E contains 1842 entities with `dbpedia:Playwright` as a type in LHD Extension ($\mathcal{KG}_{map}$) for DBpedia ($\mathcal{KG}$). Skipping entities without any type in DBpedia or with a type not in the DBpedia Ontology namespace, the list of the types associated with these 1842 entities (each type is followed by entity count): Comedian:1, MemberOfParliament:1, Royalty:1, BritishRoyalty:1, MilitaryPerson:1, Presenter:1, Politician:2, OfficeHolder:7, MusicalArtist:5, Writer:266, Artist:277, Agent:521, Person:521.

The output of the Candidate generation algorithm can already be used for probabilistic type prediction for a given entity. This process is exemplified in Algorithm 3 (contained in Section 7), which outputs the conditional probabilities for the specified parent class in the target ontology.

The selection process (Algorithm 2) is two stage. In the pruning step, the algorithm iterates through the candidates removing those which are, as indicated by the numbers of supporting entities, only a supertype of a more specific type on the list of Candidates C. Higher number of supporting entities implies reliability, however, the specific types tend not to have the highest values.

Candidate *type* is removed if there is its subtype *type′* in the list of Candidates C, which has more than *TRADEOFF* * *type.supp* supporting entities. Finally, the type with the highest support is selected from the pruned set of types. The process is illustrated in Example 7.

The effect of the setting of the *TRADEOFF* constant on the specificity and accuracy of the resulting types is investigated in Subs. 8.8.

Example 7.
Candidate pruning removes Royalty, Agent, Artist and Person and Politician from the list of candidates. Royalty is removed in favour of its subclass BritishRoyalty, which has the same number of supporting entities (one). The following three types Agent, Person and Artist are removed in favour of their subclass Writer. While Writer has less supporting entities than Artist or Person or Agent, the drop in support is within tolerance of the *TRADEOFF* constant set to 0.2. Similarly, Politician is removed in favour of its subclass MemberOfParliament.
The result of pruning is: Comedian, MemberOfParliament, BritishRoyalty, MilitaryPerson, Presenter, MusicalArtist, OfficeHolder, Writer. Finally, the algorithm selects $type_{opt}$=Writer as the type with the highest number of supporting instances in the pruned set.

The standalone output of the STI algorithm for given type is one mapping, such as `dbpedia:Playwright` → `dbo:Writer`.

Algorithm 1 Candidate Generation

Require: $type_{map}$ a class which we desire to map, $\mathcal{O}_{\mathcal{KG}}$ a target ontology containing types to which the mapping should be performed, $\mathcal{KG}$ knowledge graph containing instances of classes from $\mathcal{O}_{\mathcal{KG}}$, $\mathcal{KG}_{map}$ knowledge graph containing instances of class $type_{map}$.
Ensure: C – set of candidate mappings $\{\langle type\rangle\}$, where *type* is class from $\mathcal{O}_{\mathcal{KG}}$ associated with probability

$C := \emptyset$
$E :=$ set of instances of $type_{map}$ in $\mathcal{KG}_{map}$
for *entity*$\in E$ **do**
  types := set of classes *entity* has in $\mathcal{KG}$
  for *type*$\in$ *types* **do**
    if *type* is not a $\mathcal{O}_{\mathcal{KG}}$ class **then**
      continue
    end if
    if *type* $\notin C$ **then**
      add *type* to C
      C[type].supp := 1
    else
      // holds the number of entities assigned with *type* in $\mathcal{KG}$ and simultaneously with $type_{map}$ in $\mathcal{KG}_{map}$
      C[type].supp += 1
    end if
  end for
end for
return C

Algorithm 2 Candidate Pruning and Selection

Require: $C = \{\langle type\rangle\}$ set of Candidates from Alg. 1, each associated with support, T – TRADEOFF threshold
Ensure: $type^{opt}$ – class from $\mathcal{O}_{\mathcal{KG}}$

totalSupp := $\sum_{type} C[type].supp$
discardMade := true
while *discardMade* **do**
  discardMade := false
  for *type* $\in C$ **do**
    if $\exists type' \in C$: *type' subclass of type*, *type* $\neq$ *type'*, $type'.supp > T * type.supp$ **then**
      remove *type* from C
      discardMade := true
      break
    end if
  end for
end while
return $type^{opt}$: type with the highest *supp* from C

6. Support Vector Machines Classifiers

Since the set of target classes forms a hierarchy and we would like to experiment with fusing outputs of multiple models, we needed an algorithm that can output probability distributions, which can be easily aggregated in a hierarchical setup. SVMs meet this requirement, additionally this approach has a previous strong record in the hierarchical text categorization domain.

Our setup involves a knowledge graph $\mathcal{KG}$ containing entities, each associated with zero or more types. The types form an ontology (taxonomy) $\mathcal{O}_{\mathcal{KG}}$. The purpose of the classifier is to assign the most specific correct type from the ontology to those entities in the knowledge graph $\mathcal{KG}$ that have a missing type. In order to train the classifier, existing entity-type assignments in $\mathcal{KG}$ are used as the training data. The entities are represented using a bag-of-words model created from the textual properties associated with the entities in $\mathcal{KG}$. If an entity does not have the required textual property it is exempt from the processing.

As the classification algorithm, we use SVM with *linear kernel*, the choice of which is justified in Subs. 8.7. We also let the SVM implementation output probability distribution for all target classes, which is required by the fusion process.

Further, we describe our setup in a greater detail using DBpedia as the knowledge graph $\mathcal{KG}$. In DBpedia there are multiple textual properties associated with most entities. To build the classifier, we selected two of them: short abstracts and article categories.

We should ideally have an SVM classifier for each non-leaf class in the DBpedia ontology. However, since multiple classes in the DBpedia ontology have only a few instances, better results are obtained if a dedicated *classification ontology* $\mathcal{O}_{cl}$ is derived from the DBpedia ontology.

For each non-leaf type in the classification ontology, we create two classifiers: *abstract* classifier, which uses the text of the short abstract, and the *cat* classifier, which uses article categories (treated as text).

Once the classifiers have been trained, the classification models are applied to assign types to entities using the standard *Multiplicative Scoring Rule* approach or our *Additive Scoring Rule* approach. The latter has the advantage that it outputs more specific types.

This section is organized as follows. The bag-of-words feature set used by our classifier is described in Subsection 6.1. Subsection 6.2 covers the classification ontology. The final type selection from the prediction of the individual SVM models is performed after the STI results have been merged in. This is described in Section 7.

6.1. Feature Set

The dataset consists of instances that correspond to entities (articles) in Wikipedia. Each entity is represented with the bag-of-words vector space model, which is created from the *short abstract* and article *categories* as retrieved from DBpedia.

Short abstracts represent entity in a more concise way than full abstracts (e.g. *John Forrest* entity is described by 208 words and 1317 characters in the case of its full abstract and by 72 words and 447 characters in the case of short abstract). In our experience, short abstracts provide comparable results to full abstracts with lower computational demands.

Categories naturally reflect a type of a given entity to a certain extent. Interestingly, they are not necessarily shorter than short abstract. It should be emphasized that we treat the article category data as text.

During the pre-processing step, short abstracts and categories are lowercased and tokenized into separate words. Further, stop words along with numbers are removed and term frequencies are computed for each pre-processed token per given entity. In the case of categories we further applied noun stemming.

6.2. Classification Ontology $\mathcal{O}_{cl}$

Since a supervised model is applied, it is necessary to restrict the classification to types in the knowledge graph for which sufficient amount of training data (i.e. instances) is available.

In order to achieve this the DBpedia ontology is reduced to DBpedia types having at least 100 instances while preserving asserted hierarchical relationships. Second, DBpedia types having only one to four direct subclasses are removed. This implies that these removed DBpedia types are replaced by their DBpedia subtypes. All DBpedia types are subsumed by the most general class Thing in the classification ontology. The thresholds of 100 and 4 respectively were chosen based on small-scale experimentation of the data, additional performance

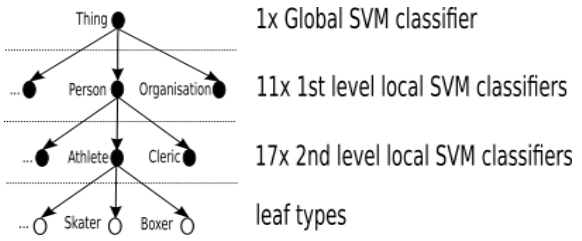

Figure 2: Structure of hierarchical SVM classifier (29 classifiers and 276 classes in total).

improvement can be gained when these are result of proper parameter tuning.

It should be noted that we obtained slightly improved results when the automatically built classification ontology is further manually edited. We explored this possibility in one of our development prototype. Our conclusion is that the small improvement in accuracy does not offset the costs associated with this manual intervention into the classification process each time the DBpedia ontology is changed. From these experiments we include in the following at least several figures. While the particular numbers are slightly different from the automatic version, these figures can be used to illustrate the role of the classification ontology in our workflow.

Since the maximum depth of the manually edited ontology was set to three, we have three layers of SVM classifiers (see Figure 2). There is one global SVM classifier, 11 first level local SVM classifiers and 17 second level local SVM classifiers:

- The *Global* SVM classifier covers top level types from the DBpedia Ontology (i.e. subtypes of the most general class *Thing*).
- *First level local SVM classifiers* enable classification into subtypes of (some) top level types.
- *Second level local SVM classifiers* enable classification into subtypes of (some) types assigned by the *first level local SVM classifiers*.

Table 3 contains details about the global SVM classifier and the first level local SVM classifiers, Table 4 covers second level SVM classifiers: *types* refers to the number of types the classifier distinguishes, *entities* refers to the number of entities on which the classifier was trained, *attributes* refers to the number of attributes (in the bag-of-words setting) the classifier works with and finally *accuracy* states how accurate the SVM classifier was in a ten-fold cross-validation setting.

Table 3: Global SVM classifier and first level local SVM classifiers. Each classifier has two variants (*abstract* and *cat*). The first number corresponds to a classifier based on short abstract (*abstract*) and the second one to a classifier based on categories (*cat*).

Classifier	Types	Entities	Attributes	Accuracy
Global	29/28	2900/2745	20419/4562	86%/89%
Work	13/13	1300/1295	11153/2654	86%/91%
Species	5/5	500/496	3402/623	92%/90%
Place	12/12	1200/1187	9136/2253	83%/89%
Transportation	7/7	700/700	5639/1137	95%/96%
Event	6/6	600/599	5472/1078	90%/93%
Device	3/3	300/298	3627/449	98%/98%
Organisation	12/12	1200/1194	9392/1925	91%/92%
Person	30/30	3000/3000	18980/6122	81%/81%
AnatomicalSt.	8/8	800/799	3878/191	93%/96%
CelestialBody	4/4	400/400	1969/318	97%/87%
SportsSeason	4/4	400/400	2530/590	91%/98%

Table 4: Second level local SVM classifiers. The first number corresponds to a classifier based on short abstract (*abstract*) and the second to a classifier based on categories (*cat*). *Arch.* means *ArchitecturalStructure*, *Educat.* means *EducationalInstitution* and *Popul.* means *PopulatedPlace*.

Classifier	Types	Instances	Attributes	Accuracy
WrittenWork	6/6	600/599	6287/1289	91%/95%
MusicalWork	4/4	400/400	3810/1134	82%/93%
Animal	9/9	900/896	6099/1006	87%/77%
Plant	7/7	700/692	4318/612	93%/92%
Arch.	22/22	2176/2174	13946/2991	89%/91%
SportsEvent	8/8	800/798	4253/858	96%/96%
Athlete	34/34	2550/2549	12674/4031	98%/96%
Broadcaster	3/3	300/300	2736/605	87%/83%
Company	4/4	400/400	3881/518	98%/99%
Educat.	3/3	300/300	2806/778	96%/96%
SportsLeague	5/5	500/491	2940/419	98%/98%
SportsTeam	6/6	600/595	4094/759	99%/97%
Artist	7/7	700/700	6690/1577	90%/86%
Cleric	4/4	400/400	3562/1224	95%/95%
Politician	7/7	700/700	5081/2089	76%/80%
NaturalPlace	8/8	775/773	5809/1252	90%/93%
Popul.	6/6	600/587	4972/1234	85%/86%

7. Hierarchical Combination of Classifiers

The final step in our solution for type inference is merging the results of STI and SVM models and selecting the type that poses a compromise between specificity and reliability from the assigned ones.

The results of STI and the two SVM models (abstract and categories) are merged using *linear opinion pool*: the probability distributions output by

the individual models are simply averaged. This is performed by Algorithm 4. The merging process also takes into account the situation that a prediction from a particular classifier may be missing for given class.

The SVM classifiers provide the function *parent.prob_classify(e)* that for given entity e outputs the conditional probabilities for a particular parent concept (an individual SVM classifier). Algorithm 3 presents how a structurally compatible output can be generated from the STI output. This short algorithm addresses two principal points:

- Making a prediction for a specific entity as STI provides mapping for classes not entities.
- Use of different target ontology as STI Candidate Generation algorithm uses the full target ontology $\mathcal{O}_{\mathcal{KG}}$, while the SVM are trained on its subset $\mathcal{O}_{cl}$. This is achieved by simply skipping the classes on STI Candidate generation output, which are not included in $\mathcal{O}_{cl}$.

Example 8 illustrates the classification with Algorithm 3.

Example 8. Consider the use of STI-prob on the classification entity e = Václav Havel with respect to the *Artist* parent class.
Method Artist.prob_classify(e) first looks up i in $\mathcal{KG}_{map}$ obtaining $type_{map}$= `dbpedia:Playwright`. Next, it executes *candidate generation*($type_{map}$) obtaining the set of candidate classes from $\mathcal{O}_{\mathcal{KG}}$ along with support values (ref. to Example 6 featured in Section 5). Finally, these support values are converted to the following probabilities for subclasses of Artist in $\mathcal{O}_{cl}$: Comedian 0.4%, MusicalArtist 0.4%, Writer 99.2% (only classes with non-zero probability are listed).

The result of Algorithm 4 is a set of conditional probabilities assigned to classes in the classification ontology. Next we apply *multiplicative scoring rule* approach (Algorithm 5) proposed in [17] for hierarchical classification of web content with SVMs. This algorithm takes on the input computed set of conditional probabilities from Algorithm 4 and propagates their values downward the taxonomy, removing classes with joint probability lower than a preset threshold. While this approach is very simple, we feature our implementation in Algorithms 5 and 6 for reference purposes.

One modification to Algorithm 5 we experimented with was averaging the probabilities rather than computing the joint probability by multiplying them. This modification aims at more reliable selection of the final type, while maintaining reasonable specificity of the selected type. With the MSR approach, the types associated with highest probability are the ones on the most general level of the ontology. Assignment of these types would not be very useful. With averaging as the aggregation operator the maximum can be on any level. We call this modification the *additive scoring rule* (ASR). We tried adapting the pruning for ASR since the confidence associated with subtype can be higher than of its supertype in the ASR approach, however, we found the current version in Algorithm 6 to work better.

We introduce two strategies for selecting one type per entity from the multiple types that can survive the pruning step in the following subsection.

Algorithm 3 Classify instance with STI-prob *parent.prob_classify(e)*

Require: e entity to classify, *parent* in function name is a concept $\in \mathcal{O}_{cl}$ with respect to which the classification should be performed, the source knowledge base $\mathcal{KG}_{map}$
Ensure: *prob* prob. distribution over children of $parent \in \mathcal{O}_{cl}$
1: $type_{map}$:= type of e in $\mathcal{KG}_{map}$
2: C := *candidate generation*($type_{map}$) // see Alg.1
3: **for** *type* in children of *parent* in $\mathcal{O}_{cl}$ **do**
4: $prob[type] = \frac{C[type].supp}{\sum_{s \in siblings(type, \mathcal{O}_{cl})} C[s].supp}$
5: **end for**
6: **return** *prob*

Algorithm 4 Linear opinion pool for hierarchy

Require: e – entity to be classified, $\mathcal{O}_{cl}$ Classification Ontology, cl – grid of $|M|$ x $|N|$ probabilistic classifiers, where N is the set of non-terminal types in $\mathcal{O}_{cl}$ and M the set of modalities, classifier for some combination of $m \in M$ and $n \in N$ may not exist, weight w_m for each modality, $\sum_{m \in M} w_m = 1$
Ensure: *prob* – array of conditional probabilities associating every class $c \in \mathcal{O}_{cl}$ except root (Thing) with a conditional probability of c given its parent p in $\mathcal{O}_{cl}$
1: //there is at least one classifier for each non-terminal class
2: $prob[*] := 0$
3: **for** non-terminal class $p \in \mathcal{O}_{cl}$ **do**
4: // we have up to 3 modalities: SVM categories, abstract and STI. If a classifier in any modality is missing, the weight vector needs to be adjusted by a factor of ws
5: $ws := 0$
6: **for** $m \in$ M **do**
7: **if** classifier $cl[m, p]$ exists **then**
8: $ws = ws + w_m$
9: **end if**
10: **end for**
11: **for** $m \in$ M, c $\in$ target classes of $cl[m, p]$ **do**
12: $prob[c] := prob[c] + \frac{w_m}{ws} * cl[m, p].prob\_classify(e)[c]$
13: **end for**
14: **end for**
15: **return** *prob*

7.1. Final Type Selection

By default MSR approach returns set of types. In order to provide a classification result, the algorithm selects a final type from *Candidates* according to probabilities associated with each type.

We use two approaches to determine the final type from the output of Algorithm 5 (MSR or ASR):

- α strategy selects the type with maximum joint probability from non-top[6] *leaf types*. This approach is used in conjunction with the default MSR version of the algorithm.
- β strategy selects the type with maximum joint probability from *all types*. This approach is used in conjunction with the ASR version of the algorithm.

8. Evaluation

Due to the unavailability of a suitable evaluation resource, we decided to build a *gold standard* dataset that associates a DBpedia entity (a Wikipedia article) with a manually curated list of types from the DBpedia Ontology. Such dataset allows not only to report on performance of our approach, but also to provide a comparison with other algorithms in an objective way.

The annotation setup for the three gold standard datasets GS1, GS2 and GS3 is described in Subsection 8.1. Evaluation metrics are described in Subsection 8.2. Subsection 8.3 describes the setup of our algorithms and Subsection 8.4 presents their results. Subsection 8.5 provides a comparison with the SDType algorithm. Subsection 8.6 evaluates the quality of types in DBpedia and assesses the suitability of or approach for completing types for entities without any type in DBpedia.

Subsection 8.7 justifies the choice of linear SVMs as the base learner comparing performance with other common classification algorithms. Subsection 8.8 evaluates the effect of varying the *TRADE-OFF* parameter of the STI algorithm.

[6] That is leaf types with parent Thing. We obtained better results when these were excluded.

Algorithm 5 Multiplicative Scoring Rule – Computing joint probability

Require: $prob$ – conditional probability for $c \in \mathcal{O}_{cl} \setminus \{Thing\}$ given its parent p in $\mathcal{O}_{cl}$
Ensure: $jprob$ – joint probability for $c \in \mathcal{O}_{cl} \setminus \{Thing\}$
1: $jprob := prob[class]$
2: // proceeds breadth-first from root to leaf
3: **for** $type \in$ non-leaf classes from $\mathcal{O}_{cl} \setminus \{Thing\}$ **do**
4: **for** $subtype \in children(\mathcal{O}_{cl}, type)$ **do**
5: $jprob[subtype] := jprob[subtype] \times jprob[type]$
6: **end for**
7: **end for**
8: **return** $jprob$

Algorithm 6 Multiplicative Scoring Rule – Pruning

Require: $jprob$ – joint probability for $c \in \mathcal{O}_{cl} \setminus \{Thing\}$, threshold T
Ensure: $jprob$ – joint probability with some types removed
1: // proceeds breadth-first from root to leaf
2: **for** $type \in$ classes from $\mathcal{O}_{cl} \setminus \{Thing\}$ **do**
3: **if** $jprob[type] \leq$ T **then**
4: **for** $subtype \in descendants(\mathcal{O}_{cl}, type)$ **do**
5: remove $subtype$ from $jprob$ and from $\mathcal{O}_{cl}$
6: **end for**
7: remove $type$ from $jprob$
8: **end if**
9: **end for**
10: **return** $jprob$

8.1. Building the Gold Standard

Since the task of assigning a final type to the entity described in English Wikipedia article does not necessarily need an expert we rely on collecting judgments from paid volunteer contributors via a *crowdsourcing* service.

We decided to perform crowdsourcing as opposed to expert annotation based on experimental evidence presented in a seminal article of Snow et al. [23] that evaluates the quality of crowdsourced annotations on five different natural language processing tasks. For all five task types the paper reports high agreement between Amazon Mechanical Turk non-expert annotations and expert labelers.

8.1.1. Task setup

For the crowd sourcing service we opted for *CrowdFlower*[7] as Amazon Mechanical Turk is not available for Europe. The annotation instructions asked the CrowdFlower workers to assign *the most specific category* (categories) from the presented *taxonomy of categories* for each Wikipedia article describing certain entity from the given list. The taxonomy used corresponded to the DBpedia 2014 ontology, which contains almost 700 DBpedia types.[8] The annotators were aided in the task

[7] `http://www.crowdflower.com/`

[8] Since CrowdFlower only allows one super-category for each category in a taxonomy, we did one correction: *Library* is originally subsumed by both *EducationalInstitution* and *Building* in DBpedia Ontology, for the taxonomy we only kept subsumption to *EducationalInstitution*.

Table 5: Overview of evaluation dataset. Column *entities* denotes the number of entities in the annotation task (*all*), number of entities where annotators agreed on 'not found' category (*nf*), number of entities where annotators agreed on 'disambiguation page' category (*dp*), number of entities where annotators did not agree based on majority vote (*nma*), number of entities with ground truth (*gt*) and the number of the "hard" entities – those with groundtruth for which there is no type in DBpedia (gt_h). Interannotator agreement is reported in terms of Krippendorff's alpha. Column *workers* reports the number of unique annotators. LHD Fusion 3.9 denotes the set of entities in DBpedia 3.9 for which a hypernym was extracted but not mapped with exact string matching to DBpedia Ontology, cf. Fig. 1).

dataset	entities						Kr. α	workers	sample source
	all	nf	dp	nma	gt	gt_h			
GS 1	1219	140	5	53	1021	373	0.529	64	LHD Fusion 3.9
GS 2	176	2	2	12	160	NA	0.514	16	Intersection of SDType 3.9 and LHD Fusion 3.9
GS 3	1165	22	47	63	1033	331	0.503	48	Randomly drawn articles from Wikipedia

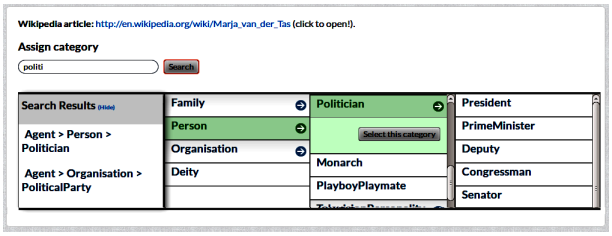

Figure 3: Interface of the CrowdFlower taxonomy annotation tool. The annotators can navigate through the taxonomy either by clicking on a concept, which shows its subtypes, or by fulltext search, which shows all concepts with substring match in the concept name along with the full path.

of locating the right class among the 700 candidates by the taxonomy annotation tool offered by the CrowdFlower platform, which enables the annotators to quickly browse through the taxonomy using fulltext queries. Figure 3 shows a screenshot of the tool.

It should be noted that it was up to the annotators to choose which part of Wikipedia articles they will read and identify types from,however, many of them might have opted only for reading the start of the article. This could have slightly favoured our SVM algorithm trained on short abstracts, and the evaluation of the LHD Core, which is based on the lexico-syntactic analysis of the article's first sentence.

The CrowdFlower platform has a wide range of setting for controlling the quality of the work done by its workers. Our setup was as follows:

- Only workers residing in the following countries were eligible: Australia, Canada, Denmark, Germany, Ireland, Netherlands, Sweden, United Kingdom and United States. The workers were Level 1 Contributors, which are described by the CrowdFlower service as accounting for 60% of monthly judgments and maintaining a high level of accuracy across a basket of jobs.
- Amount of 0.02 USD was paid for each annotated entity to a worker.
- The workers were given a quiz before starting a task with minimum of four test questions (entities to annotate). Only workers with accuracy of 30% or higher could continue in the task.
- To maintain high accuracy, additional test questions were asked as the workers were completing their job.
- A speed trap was put in place that eliminated workers who took less than 10 seconds to complete a task.

Concerning the appropriateness of the remuneration, [24] gives half-a-penny per question as the rule of thumb for payment on crowd sourcing services, which our remuneration exceeded. To further ensure that the pay is appropriate, we checked the satisfaction scores reported in the final questionnaire by the workers. On a 1–5 Likert scale (1 worst, 5 is best), the workers rated their remuneration on average between 3.1 to 4.0. None of the jobs had pay rating in the red band.[9]

Each entity was typically annotated by three to four workers. The CrowdFlower platform ensured that the annotations from workers who failed the test questions were replaced by untainted annotations.

[9]The crowdflower platform assigns three color codes to the final scores (red, orange and green) to help interpreting the questionnaire results.

Our setup can be somewhat compared the crowdsourcing evaluation performed in [9]. There the number of workers annotating each entity was similar to ours (three). Also, similarly to our setup, the workers were supposed to select only one best type. One major difference is that in [9] the workers were presented preselected types (with the option to enter a new type), while in our system they had to select the type from a larger fixed list of types. Another difference is that in [9] no majority type was selected for given entity. Instead, all types were used with a relevance score corresponding to the number of workers selecting the respective type.

8.1.2. Interannotator agreement

For measuring interannotator agreement we have opted for Krippendorff's alpha [25] (as implemented in [26]), since this measure supports multiple annotators and is applicable to incomplete data. The values of Krippendorff's alpha as reported in Table 5 are in the 0.4 to 0.6 range which is considered as moderate agreement for kappa-like coefficients ([27] cited according to [28]). While some sources would consider already value below 0.8 as unacceptable for any serious purpose [25, Chapter 11, page 242], it should be noted that our annotation task with hundreds of distinct concepts to choose from was exceptionally difficult. Also, when computing the α we used binary distance function (i.e. the similarity of two distinct yet semantically close annotations was not considered). Annotations assigning more than one concept were ignored for the purpose of computing the α value.

8.1.3. Gold standard datasets

The gold standard for given entity consists of all types that were assigned by at least two annotators to the entity. As a consequence, not all entities included in the annotation task are contained in the gold standard (cf. Table 5). The process of establishing the gold standard is illustrated by Example 9.

Example 9.
Wikipedia article describing *August Nybergh* entity was annotated in the following way by four annotators:

- {Agent > Person > Politician > Senator}, {Agent > Person > Politician > MemberOfParliament}
- {PersonFunction > PoliticalFunction}
- {Agent > Person > Politician > Senator}, {Agent > Person > Politician}
- {Agent > Person > Politician}

The first and the third annotator assigned two different most specific types. The final most specific type, having frequency at least two, is the *Senator* type. The Politician type was not added to the gold standard as it is a superclass of Senator.

Any redundant superclasses were removed as also illustrated by the example. The annotators could assign more than one most specific type to the entity. Multiple final types were assigned for less than 1% of entities in our initial annotation task, thus we ignored multiple types in our evaluation, selecting one type randomly in such cases for the gold standard. Besides categories corresponding to types in the DBpedia Ontology, annotators could select 'not found' category if they could not find the article or 'disambiguation page' category in case the article was a disambiguation page in their opinion. Entities with these categories are omitted from the gold standard. In order to foster reusability of the dataset as the evaluation ontology we used the most up-to-date released version of the DBpedia Ontology (2014) at the time.

The gold standard resulting from the annotation process is composed of three datasets depending on the subset of DBpedia/Wikipedia from which the entities to be annotated were drawn. Table 5 shows an overview of the three gold standard datasets, totaling 2214 entities with groundtruth.

8.2. Evaluation Metrics

We use four evaluation measures: exact precision, hierarchical precision, hierarchical recall and hierarchical F-measure. The first measure corresponds to precision which does not take into account the type hierarchy:

$$P_{exact} = \frac{\sum_i |P_i \cap T_i|}{\sum_i |P_i|}, \quad (1)$$

where P_i is the set of the most specific types predicted for test example i, T_i is the set of the true most specific type of test example i.[10]

The other three measures consider the type hierarchy. Hierarchical precision (hP), hierarchical recall (hR) and hierarchical F-measure (hF) are defined according to [29] as follows:

$$hP = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{P}_i|}, \quad (2)$$

$$hR = \frac{\sum_i |\hat{P}_i \cap \hat{T}_i|}{\sum_i |\hat{T}_i|}, \quad (3)$$

$$hF = \frac{2 * hP * hR}{hP + hR}, \quad (4)$$

where $\hat{P}_i$ is the set of the most specific type(s) predicted for test example i and all its (their) ancestor types and $\hat{T}_i$ is the set of the true most specific type(s) of test example i and all its (their) ancestor types.

8.3. Evaluated Setups

Our evaluation involves the following setups of our algorithms:

- *LHD Core*: lexico-syntactic patterns, extracted types were successfully mapped to DBpedia Ontology with exact string matching (LHD Core, approach published in [2]).
- STI$_{prune}$: lexico-syntactic patterns, type mapping was performed by the standalone STI with pruning (exact string matching failed).
- $hSVM_{cat}$: hierarchy of SVM models trained on article categories with the final types selected with Multiplicative Scoring Rule (MSR).
- $hSVM_{abstract}$: hierarchy of SVM models trained on article abstracts with the final types selected with MSR.
- $hSVM_{text}$: $hSVM_{cat}$ and $hSVM_{abstract}$ merged with linear opinion pool using equal weights.
- $hSVM_{text}STI$: all three models ($hSVM_{cat}$, $hSVM_{abstract}$, STI without pruning) were merged with linear opinion pool, the final types were selected with MSR.
- $hSVM^{add}_{text}STI$: all three model results were merged with linear opinion pool, the final types were selected with ASR.
- *Core+*STI$_{prune}$: merge of results of LHD Core and STI_{prune}.
- STI$_{prune}$ + $hSVM_{text}$: merge of results of STI_{prune} and $hSVM_{text}$ where results of STI are prioritized (if an entity has types assigned both in STI and $hSVM_{text}$, only results from STI are used).
- *Core+* $hSVM_{text}STI$: merge of results of LHD Core and $hSVM_{text}STI$ where results of LHD Core are prioritized.
- *Core+*STI$_{prune}$*+*$hSVM_{text}$: merge of results of LHD Core, STI_{prune} and $hSVM_{text}$ where results of LHD Core and STI are prioritized.

The results of LHD Core and STI were generated by the LHD framework [12] and are available as part of the DBpedia 2014 release. The tradeoff threshold constant of STI was set to 0.6, which is a value that maximizes F-measure on GS 1 (refer to Subs. 8.8). Note that this threshold is used only in the standalone STI runs. Based on parameter tuning, the STI weight for linear opinion pool was set to 0.33.

All SVM models were also generated on DBpedia 2014. Threshold for MSR or ASR algorithms for combining SVM models was selected according to the maximum hF-measure based on evaluation on a different dataset. That is, for GS1 dataset we used the best hF-measure computed on GS3 and vice versa. The optimization step was 0.01.

For reference purposes, our evaluation also involves the following:

- *SDType*: SDType results for DBpedia 3.9 obtained from the DBpedia website.[11]

[10] We measure P_{exact} only for algorithms that assign at most one type (P_i and T_i always contain at most one element).

[11] The reason why we use 3.9 and not 2014 results is that the GS2 dataset designed for comparison of SDType results with our approach was generated on version 3.9. Since SDtype result for version 2014 does not contain many of these entities, the evaluation sample would be too small.

- *DBpedia 2014.* Entity type assignments in the DBpedia ontology namespace that are part of the English DBpedia 2014 release.

The evaluations are performed in addition to GS1, GS2, and GS3 also on GS3 subset GS3h that contains the "hard" entities – those with no type assigned in DBpedia 2014.

8.4. STI, hSVM and their combinations

Table 6: Evaluation on gold standard GS1 (1021 entities) and GS2 (160 entities).

Classifier	P_{exact}	hP	hR	hF
STI_{prune}	.446	.780	.589	.671
$hSVM_{abstract}$	NA	.622	.550	.584
$hSVM_{cat}$	NA	.587	.644	.614
$hSVM_{text}$	NA	.713	.668	.690
$hSVM_{abstract}\alpha$	.261	.622	.597	.609
$hSVM_{cat}\alpha$	.267	.715	.611	.659
$hSVM_{text}\alpha$	.310	.719	.675	.696
$hSVM_{text}STI\alpha$	.347	.735	.730	.732
$STI{+}hSVM_{text}\alpha$	.400	.763	.734	.748
$hSVM^{add}_{text}\beta$	.365	.719	.706	.712
$hSVM^{add}_{text}STI\beta$	.294	.817	.652	.726
DBpedia (2014)	.548	.890	.665	.761
	GS2			
SDType (3.9)	.338	.809	.641	.715

We evaluated separately the STI and hSVM classifier and their combination using Multiplicative Scoring Rule (MSR) and its ASR variant. The results are presented in Table 6.

With respect to our individual approaches, STI outperforms all runs of the hSVM classifier including its combination with STI in the P_{exact} measure, while hSVM has better results with regard to the hierarchical measures. The good STI result might be to certain extent influenced by existing type assignment in DBpedia, since the STI classifier exploits the co-occurrence information with types already in DBpedia. Also GS1 dataset was used to tune the TRADEOFF threshold affecting the results of STI_{prune}. An unbiased evaluation on GS3h shows that indeed the hierarchical precision of STI drops below hierarchical SVM on this dataset.

With respect to the hSVM classifier, the improvement in all metrics for $hSVM_{text}$, which uses both abstract and categories as input features, suggests that these sets of features are not redundant. What we have not evaluated is if a hSVM model built upon a merge of both feature sets would not provide even better results than building two models and merging them. Individually, the classifiers built upon the *categories* feature set perform slightly better than the ones built upon *abstracts*, but this difference is not statistically significant as the 95% Wilson confidence intervals for binomial probabilities for exact match overlap.[12]

The comparison between the baseline MSR approach $hSVM_{text}STI\alpha$ and our additive variant $hSVM^{add}_{text}STI\beta$ shows that the additive version provides an improvement in hierarchical precision, but this is offset by even higher drop in the remaining metrics.

Selecting one final type with either α or β strategies is better in terms of all metrics than the vanilla MSR approach $hSVM_{text}$, which uses all types with joint probability exceeding the threshold.[13] Since selecting one type per entity is preferred (DBpedia infobox-based framework and STI also assign one type) we therefore select $hSVM_{text} + STI\alpha$ as the final approach. This corresponds to merge of the results of STI and hSVM algorithms rather than their fusion with linear opinion pool.

Overall, the hSVM approach can be used to assign type to entities unmatched by the lexico-syntactic patterns, but it does not improve – at least with the current version of the linear opinion pool fusion approach – the existing type assignments generated by the STI algorithm.

8.5. SDType

This section compares our approach to the state-of-the-art algorithm SDType described in Section 2.5.

We evaluated SDType on gold standard dataset *GS*2, which covers untyped instances in DBpedia 3.9 that were assigned a type with SDType. The

[12] Paper [30] suggests that when interval overlap is used for significance testing, 95% confidence interval will give very conservative results.

[13] A noteworthy comparison is that the α and β strategies, which select one final type, have higher recall than vanilla MSR, which selects all types above the threshold. The reason is that the threshold weights were trained separately for all three approaches.

evaluation statistics are provided in the bottom of Table 6. Results on GS1 show that on this sample SDType is very reliable in selecting types with hierarchical precision very close to that of DBpedia. Hierarchical recall and F-measure have little meaning on GS1 for SDType since a criterion for selecting GS1 entities was the presence of a type assigned with SDType.

Our second evaluation was performed on GS3h containing randomly drawn articles from English Wikipedia that are untyped in DBpedia 2014. The hierarchical F-measure and the number of covered entities show that SDType assigned a type only to a very small number of instances compared to all other approaches. When SDType did assign the type, the hierarchical precision was on par with hSVM. Inspection of P_{exact} on GS2 and GS3h evaluation shows that the specificity of types assigned by SDType is relatively low.

Overall, SDType completes a high number of untyped instances, but these are often instances without any Wikipedia page that were possibly created in DBpedia from Wikipedia "red links". In contrast, our algorithms require at least the abstract of categories to be present. Overall, this shows that SDType and our approach are highly complementary.

8.6. DBpedia

The entities in the gold standard GS3 were randomly selected from all the Wikipedia articles. The evaluation using GS3 thus provides the most objective evaluation of all approaches for type assignment.

For DBpedia type assignment to given entity we consider only the most specific DBpedia Ontology types, which is in-line with how our gold standard is constructed. First, we obtained all DBpedia Ontology types for given entity and next we selected the most specific types.[14]

Overall, DBpedia has the best hierarchical precision. However, the results, presented in Table 7, perhaps surprisingly show that the lexico-syntactic patterns (LHD Core) provide exact types with higher precision than DBpedia (22% relative improvement in P_{exact}). We hypothesize that this is caused by some infoboxes being mapped in the DBpedia extraction framework to higher-level types than is the most specific available type in the DBpedia ontology. This interpretation is supported by DBpedia having marginally higher hierarchical precision than LHD Core. Another possible reason contributing to LHD Core having higher exact precision than DBpedia is that it was easiest for annotators to base their type assignment on the first sentence of the article from which the LHD patterns extract the type.

The results on GS3 show that all our approaches combined achieve higher hierarchical F-measure and assign types to more entities than the DBpedia infobox-based DBpedia extraction framework. The GS3 dataset contains 331 entities untyped in DBpedia (out of which 50 do not exist in DBpedia 2014 at all).[15] Out of these entities composing the GS3h dataset, our combined approach is able to assign types to 197 entities (which is 70% of untyped instances existing in DBpedia).

There are two main reasons why our most universal hSVM approach was unable to type the remaining 30% of untyped instances: part of these instances did not have any abstract and categories in DBpedia and for some instances the type assignment was computed, but was not considered reliable enough given the precomputed threshold in Algorithm 6.

8.7. Comparison with other classifiers

In order to further ground (beyond the related work discussed in Subs. 2.6) the selection of SVMs with linear kernel as our base model, we performed a benchmark on all 58 datasets, which were used to train the individual SVM classifiers. Ten percent of each dataset was used for testing, the rest for training (stratified selection). The feature set was pruned by removing features with less than 0.1 standard deviation in each dataset.

No parameter tuning for any of the classifiers was performed, the default values from the RapidMiner 5 implementation[16] of the respective classifier was used:

- **Ripper [31]**: information gain criterion used, sample ratio=0.9, pureness=0.9, minimal prune benefit=0.25.
- **SVM linear kernel**: C=0.0, $\epsilon = 0.001$, shrinking applied.

[14] Out of 1021 DBpedia entities there was not any case with more than one specific type from DBpedia ontology namespace.

[15] Based on the *titles file*

[16] `http://rapidminer.sourceforge.net`

Table 7: Evaluation on gold standard GS3 (1033 entities) and GS3h (331 entities), 50 entities from GS3 and GS3h are not present in DBpedia 2014.

	GS3 (randomly drawn articles)					GS3h (untyped instances)				
Classifier	entities	P_{exact}	hP	hR	hF	entities	P_{exact}	hP	hR	hF
DBpedia	715	.537	.902	.611	.729					
SDType						19	.105	.644	.033	.063
Core	402	.654	.864	.371	.519	26	.654	.713	.065	.119
STI_{prune}	379	.449	.754	.274	.403	106	.255	.461	.162	.240
$hSVM_{text}\alpha$	750	.307	.747	.597	.663	131	.130	.635	.293	.400
$hSVM_{text}STI\alpha$	765	.327	.757	.621	.682					
Core + STI_{prune}	781	.554	.814	.645	.720					
Core + $hSVM_{text}STI\alpha$	864	.439	.786	.720	.752	169	.169	.534	.289	.375
Core + STI_{prune} + $hSVM_{text}\alpha$	896	.465	.800	.724	.760	197	.205	.565	.379	.454

- **SVM RBF kernel**: C=0.0, $\epsilon = 0.001$, $\gamma = 0.0$, shrinking applied.
- **SVM polynomial kernel**: degree 3, $\epsilon = 0.001$, C=0.0, $\gamma = 0.0$, shrinking applied.
- **Logistic regression**: dot kernel used, convergence $\epsilon = 0.001$, C=1.0, value scaling applied.

The results depicted in Table 8 show that SVMs with linear kernels provide the best accuracy and at the same time have one of the smallest run times (aggregate for training and testing phase) on a core i5 2.6 GHz laptop with 16 GB of available memory running Open JDK 1.7. Our results are consistent with linear kernel being chosen for hierarchical classification of web content in Dumais and Chen [17] and by Liu et al. [15].

8.8. STI: Tuning the tradeoff parameter

We performed parameter tuning of the STI algorithm's *tradeoff* constant on GS1 dataset and DBpedia 3.9. We executed the algorithm with *tradeoff* set to values ranging from 0.02 to 0.99 with step 0.01.

Figure 4 shows that increasing value of this parameter improves hierarchical precision, which follows from more high level types surviving pruning. For the same reason, hierarchical recall drops as less types survive pruning. As a result, the hierarchical F-measure remains stable until around 0.8, with maximum having at tradeoff=0.6. Focusing on exact match, the best interval for the tradeoff parameter value lies between 0.2 and 0.6.

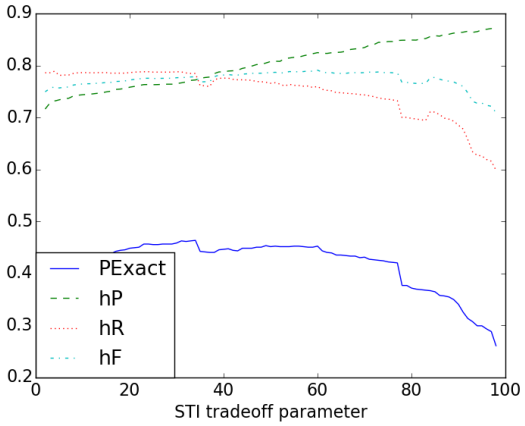

Figure 4: Effect of tradeoff threshold

Based on this examination, we suggest to set the value of the tradeoff parameter to 0.6.

9. Conclusion and Future Work

This article introduced a novel technique for inferring entity types in semantic knowledge graphs. The free text describing the entities is analyzed using algorithms from the two major directions of computational linguistics: lexico-syntactic analysis and statistical natural language processing.

The types extracted with lexico-syntactic patterns are processed with an unsupervised Statisti-

Table 8: Comparison of linear SVMs with other common classifiers

metric	Naive B.	SVM (linear)	SVM (RBF)	SVM (poly)	Ripper	Log Reg
macro avg accuracy	0.76	**0.90**	0.88	0.85	0.80	0.86
run time	less 1 minute	5 minutes	6 minutes	12 minutes	4 hours	5 minutes

cal Type Inference (STI) algorithm, which analyzes their co-occurrence with types already assigned in the knowledge graph. Further, we adapted the hierarchical Support Vector Machines (hSVMs) classifier, which we found particularly suitable due to the fact that our problem consists of a high number of taxonomically ordered classes.

During the course of the research, we were unable to find any resource that could be used for the evaluation of our algorithms providing an unbiased comparison with the accuracy of the DBpedia extraction framework. In response to this, we designed a new dataset using the commercial CrowdFlower crowdsourcing platform, which consists of more than 2.000 Wikipedia articles (DBpedia entities) that are assigned a type from the DBpedia 2014 Ontology. This dataset was made freely available along with the annotation guidelines under a Creative Commons license.

We evaluated the STI and hSVM algorithms and their fusion on the crowdsourced content and provide a comparison with DBpedia and its heuristics dataset, generated by the state-of-the-art SDType algorithm.

According to this evaluation we concluded that (1) the quality of types assigned with lexico-syntactic patterns from first sentence of Wikipedia articles is comparable to the quality of types inferred from information boxes by the DBpedia extraction framework, (2) the text categorization approach (hierarchical SVM) applied to the type inference problem has the highest recall of all but also the lowest precision (3) our approach has precision comparable to the state-of-the-art SDType algorithm while generating types for a largely different set of instances.

Notably, the hSVM approach requires as input only a free-text representation of the Wikipedia articles. Even the labeled data required to train the classifier for a particular language (i.e. DBpedia ontology types for at least some instances for each target class) can be obtained from Wikipedia's interlanguage links. The hSVM approach thus could serve as a starting point for populating type assignments in Wikipedia-based knowledge graphs for "smaller" languages or those with less development resources available.

As a future work, accuracy improvements could be gained by utilizing more sophisticated feature representation of the textual modality. A more involved enhancements would be replacement of the linear opinion pool with some meta machine learning approach such as *stacking*.

Resources for this article including the Inference dataset and the gold standard datasets are located at `http://ner.vse.cz/datasets/linkedhypernyms/`.

Acknowledgements

The authors wish to thank the three anonymous reviewers for their very helpful comments. Tomáš Kliegr thanks André Melo and Heiko Paulheim for insightful comments that led to our adoption of the standard set of hierarchical evaluation metrics and the German DAAD grant agency for making this interaction possible. The development of the STI and hSVM algorithms was supported by the European Union's 7th Framework Programme via the LinkedTV project (FP7-287911). Ondřej Zamazal has been additionally supported by the CSF grant no. 14-14076P, "COSOL – Categorization of Ontologies in Support of Ontology Life Cycle". This research is also partially supported by long term institutional support of research activities by Faculty of Informatics and Statistics, University of Economics, Prague.

References

[1] H. Paulheim, C. Bizer, Improving the quality of linked data using statistical distributions, International Journal on Semantic Web and Information Systems (IJSWIS) 10 (2014) 63–86.

[2] T. Kliegr, Linked hypernyms: Enriching {DBpedia} with targeted hypernym discovery, Web Semantics: Science, Services and Agents on the World Wide Web 31 (2015) 59 – 69.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia-a crystallization point for the web of data, Web Semantics: Science, Services and Agents on the World Wide Web 7 (2009) 154–165.

[4] J. Hoffart, F. M. Suchanek, K. Berberich, G. Weikum, YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia, Artificial Intelligence 194 (2013) 28–61.

[5] T. Kliegr, O. Zamazal, Towards linked hypernyms dataset 2.0: complementing dbpedia with hypernym discovery, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, May 26-31, 2014., pp. 3517–3523.

[6] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic Web (2016) 1–20. Preprint.

[7] A. Gangemi, A. G. Nuzzolese, V. Presutti, F. Draicchio, A. Musetti, P. Ciancarini, Automatic typing of DBpedia entities, in: P. Cudre-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, E. Blomqvist (Eds.), The Semantic Web - ISWC 2012, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 65–81.

[8] H. Paulheim, C. Bizer, Type inference on noisy RDF data, in: The Semantic Web–ISWC 2013, Springer, 2013, pp. 510–525.

[9] A. Tonon, M. Catasta, G. Demartini, P. Cudré-Mauroux, K. Aberer, TRank: Ranking Entity Types Using the Web of Data, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 640–656.

[10] H. Paulheim, Browsing Linked Open Data with auto complete, in: Proceedings of the Semantic Web Challenge co-located with ISWC2012, Springer, Boston, US, 2012.

[11] N. F. Noy, D. L. McGuinness, et al., Ontology development 101: A guide to creating your first ontology, 2001. Technical report.

[12] T. Kliegr, V. Zeman, M. Dojchinovski, Linked hypernyms dataset - generation framework and use cases, in: The 3rd Workshop on Linked Data in Linguistics: Multilingual Knowledge Resources and Natural Language Processing, co-located with LREC 2014, LDL-2014.

[13] J. Neville, D. Jensen, Iterative classification in relational data, in: Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data, pp. 13–20.

[14] J. Sleeman, T. Finin, Type prediction for efficient coreference resolution in heterogeneous semantic graphs, in: Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on, IEEE, pp. 78–85.

[15] T.-Y. Liu, Y. Yang, H. Wan, H.-J. Zeng, Z. Chen, W.-Y. Ma, Support vector machines classification with a very large-scale taxonomy, SIGKDD Explor. Newsl. 7 (2005) 36–43.

[16] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (1995) 273–297.

[17] S. Dumais, H. Chen, Hierarchical classification of web content, in: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '00, ACM, New York, NY, USA, 2000, pp. 256–263.

[18] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, J. Lehmann, User-driven quality evaluation of DBpedia, in: Proceedings of the 9th International Conference on Semantic Systems, I-SEMANTICS '13, ACM, New York, NY, USA, 2013, pp. 97–104.

[19] R. A. Howard, The foundations of decision analysis, in: W. Edwards, R. F. M. Jr., D. von Winterfeldt (Eds.), Advances in Decision Analysis, Cambridge University Press, 2007, pp. 32–56. Cambridge Books Online.

[20] Wikipedia, Wikipedia:manual of style/lead section, 2006. [Online; accessed 24-March-2016].

[21] T. Kliegr, K. Chandramouli, J. Nemrava, V. Svátek, E. Izquierdo, Wikipedia as the premiere source for targeted hypernym discovery, in: Proceedings of the Wiki's, Blogs and Bookmarking tools - Mining the Web 2.0 Workshop at ECML'08.

[22] M. Dojchinovski, T. Kliegr, I. Lašek, O. Zamazal, Wikipedia search as effective entity linking algorithm, in: Text Analysis Conference (TAC) 2013 Proceedings, NIST, 2013.

[23] R. Snow, B. O'Connor, D. Jurafsky, A. Y. Ng, Cheap and fast—but is it good?: Evaluating non-expert annotations for natural language tasks, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pp. 254–263.

[24] T. Schnoebelen, V. Kuperman, Using Amazon mechanical turk for linguistic research, Psihologija 43 (2010) 441–464.

[25] K. Krippendorff, Content analysis: An introduction to its methodology, Sage, second edition, 2004.

[26] M. Gamer, J. Lemon, I. F. P. Singhf, irr: Various Coefficients of Interrater Reliability and Agreement, 2012. R package version 0.84.

[27] G. G. K. J. Richard Landis, The measurement of observer agreement for categorical data, Biometrics 33 (1977) 159–174.

[28] R. Artstein, M. Poesio, Inter-coder agreement for computational linguistics, Comput. Linguist. 34 (2008) 555–596.

[29] C. N. Silla Jr, A. A. Freitas, A survey of hierarchical classification across different application domains, Data Mining and Knowledge Discovery 22 (2011) 31–72.

[30] M. E. Payton, M. H. Greenstone, N. Schenker, Overlapping confidence intervals or standard error intervals: what do they mean in terms of statistical significance?, Journal of Insect Science 3 (2003) 34.

[31] W. W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, pp. 115–123.