# Type Prediction in RDF Knowledge Bases Using Hierarchical Multilabel Classification

André Melo
University of Mannheim
B6 26
68161 Mannheim
andre@informatik.uni-mannheim.de

Heiko Paulheim
University of Mannheim
B6 26
68161 Mannheim
heiko@informatik.uni-mannheim.de

Johanna Völker
University of Mannheim
B6 26
68161 Mannheim
johanna@informatik.uni-mannheim.de

## ABSTRACT

Large Semantic Web knowledge bases are often noisy, incorrect, and incomplete with respect to type information. Automatic type prediction can help reduce such incompleteness, and, as previous works show, statistical methods are well-suited for this kind of data. Since most Semantic Web knowledge bases come with an ontology defining a type hierarchy, in this paper, we rephrase the type prediction problem as a hierarchical multilabel classification problem. We propose *SLCN*, a modification of the local classifier per node approach, which performs feature selection, instance sampling, and class balancing for each local classifier. Our approach improves scalability, facilitating its application on large Semantic Web datasets with high-dimensional feature and label spaces. We compare the performance of our proposed method with a state-of-the-art type prediction approach and popular hierarchical multilabel classifiers, and report on experiments with large-scale RDF datasets.

## CCS Concepts

•**Computing methodologies** → **Semantic networks;** •**Theory of computation** → *Incomplete, inconsistent, and uncertain databases;* •**Information systems** → Resource Description Framework (RDF);

## Keywords

Knowledge Base, Type Prediction, Hierarchical Multilabel Classification

## 1. INTRODUCTION

Type information plays an important role in Semantic Web (SW) knowledge bases, with type assertion axioms being one of the atomic building blocks of knowledge bases. Many datasets suffer from type assertion incompleteness. For example, for DBpedia [2], the upper bounds for completeness of DBpedia 3.8 types are estimated to be at most 63.7%, with at least 2.7 million missing type statements, while YAGO types in DBpedia 3.8 are estimated to be at most 53.3% complete [29].

A possible way to automatically infer type information on the Semantic Web is the use of reasoning, e.g., standard RDFS reasoning via entailment rules. However, reasoning methods are sensitive to noisy data, and since open knowledge bases created by crowdsourcing and/or heuristics are often noisy, logic-based reasoning approaches are likely to multiply errors. Statistical approaches, on the other hand, are more robust to noise, and are therefore more suitable for the type prediction task [28].

Since most Semantic Web knowledge bases organize the possible types as hierarchies (defined in ontologies), we propose to model the type inference problem in noisy and incomplete knowledge bases as a *hierarchical multilabel classification* problem. It is *hierarchical* because we assume the types to be structured in a hierarchy, and it is a *multilabel* problem because instances are allowed to have more than one type. For example, in a knowledge base with the type hierarchy depicted in Figure 1, the instance `Arnold_Schwarzenegger` should be typed as `OfficeHolder`, `Actor`, and `BodyBuilder`, as well as their generalizations `Artist`, `Athlete`, and `Person`, which can be inferred from type hierarchy.

As SW knowledge bases, especially cross-domain ones, can have a large number of types, the high dimensionality of the label space may challenge a multilabel classification algorithm in many ways. First, the number of training examples annotated with each type, in particular those in the lower levels of the hierarchy and in the long tail of an uneven distribution, will be significantly smaller than the total number of examples. This is similar to the class imbalance problem in single-label classification [33]. Second, the computational cost of training a multilabel classifier may be strongly affected by the number of labels [39].

Due to the presence of ontologies and their type hierarchies on the Semantic Web, viewing type prediction as a hierarchical machine learning problem is the most natural translation of the type prediction problem to a machine learning problem. However, it has never been viewed like that – to the best of our knowledge, all machine learning based methods for type prediction in SW knowledge bases proposed so far flatten the problem to non-hierarchical classification [27]. One possible reason that hierarchical multilabel classification has not been applied in the field may be scalability issues when applying those methods to large-scale



*WIMS '16, June 13-15, 2016, Nîmes, France*



DOI: http://dx.doi.org/10.1145/2912845.2912861

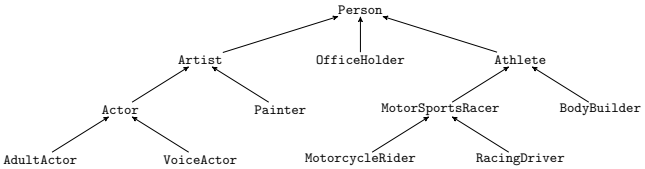



Figure 1: A subset of the DBpedia type hierarchy

SW knowledge bases.

In this paper, we propose *SLCN* which stands for *Scalable Local Classifier Per Node*, a modification of the *local classifier per node* approach, which improves the scalability by performing local sampling, feature selection, and class balancing. We show that the approach outperforms the current state of the art approaches for type prediction in SW knowledge bases, and does so in a more scalable way than existing algorithms for hierarchical multi-label classification.

The rest of this paper is structured as follows. First, we briefly introduce the foundations of hierarchical multilabel classification in section 2, followed by a problem statement in section 3. We outline the proposed approach in section 4, and report the outcome of experiments on various SW knowledge bases in section 5. We conclude with a review of related work in section 6, and conclusions and an outlook future work in section 7.

# 2. PRELIMINARIES

In this section, we lay out the foundations of hierarchical multilabel classification used in this paper.

## 2.1 Multilabel Classification Approaches

In the multi*label* classification problem, there are multiple classes and, contrary to the single-label multi*class* classification problem, instances are allowed to have more than one class. We define the set of classes as $C = \{c_1, ..., c_{|C|}\}$, and we represent the multilabel of an instance $x$ with a binary vector $y = (y_1, ..., y_{|C|}) \in \{0, 1\}^{|C|}$.

Some of the existing multilabel classification approaches are standard binary classification algorithms which have been adapted to the multilabel task, without requiring problem transformations. This includes, e.g., *AdaboostMH* [36], *MLkNN* [45] and *BPMLL* [44]. Other approaches, such as *Binary Relevance* (BR),*Classifier Chains* [33] (CC), *Label Powerset* (LP), and Random k-Labelsets (RAKeL) [40], transform the multilabel problem into a set of binary classification problems.

*Binary Relevance* (BR) is the simplest transformation approach, where a binary classifier is trained for each class assuming the classes are mutually independent. More complex transformation methods, such as *Classifier Chains* [33] (CC) and *Label Powerset* (LP), can model dependencies between the classes. There are also ensemble methods, such as Ensembles of Classifier Chains (ECC) [33] and Random k-Labelsets (RAKeL) [40], where several classifiers are trained on different subsamples and combined into a single model.

These approaches are agnostic with respect to a hierarchy relations among the labels, and hence, they do not necessarily guarantee the predicted classes to be consistent with the hierarchy.

## 2.2 Hierarchical Multilabel Classification Approaches

The hierarchical multilabel classification problem is similar to the multilabel classification problem, but the classes $C$ are structured in a hierarchy $G$. The labels of an instance should be consistent with $G$, i.e., if an instance belongs to a non-root class then it must also belong to its ancestors (i.e., $c_i \sqsubseteq c_j \wedge y_i = 1 \rightarrow y_j = 1$). The class hierarchy can be of two types: a tree, which allows nodes to have a single parent only, and a directed acyclic graph (DAG) which allows nodes to have multiple parents.

As pointed out by [37], most of the current literature focuses on working with trees as it is a simpler problem. There are mainly two types of hierarchical multilabel classification approaches: local and global classifiers. The main difference is that the former breaks down the classification problem into smaller and simpler problems exploiting the class hierarchy, while the latter consider the problem as a whole, learning a single more complex model. In the next subsections we present these approaches in more details.

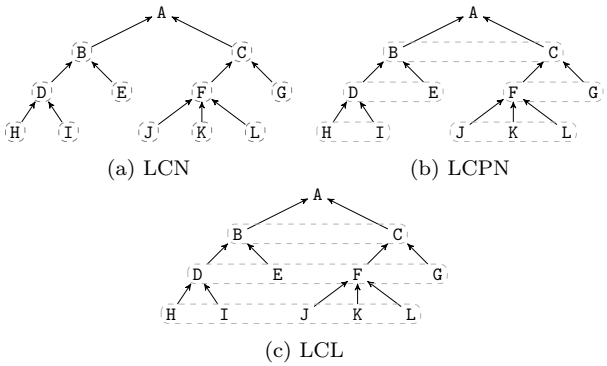
### 2.2.1 Local Classifier Approach

The hierarchy is taken into account by using a local information perspective to transform a multilabel classification problem into a set of simpler problems. According to [37], there are mainly three approaches of using local information: *local classifier per node*, *local classifier per parent node*, and *local classifier per level*. The local hierarchical classification algorithms share a similar top-down approach in their prediction phase, where the classifier first predicts its first-level (most generic) class of an instance, then it uses that predicted class to reduce the choices of classes to be predicted at the second level (the children of the classes predicted at the first level), and so on, recursively, until the most specific prediction is made.

*Local Classifier Per Node (LCN):* The local classifier per node approach consists of training one binary classifier for each node of the class hierarchy. Each local binary classifier predicts whether an instance belongs to the class associated with the node or not. There are two main ways to define the training set of the local binary classifiers, which are called *negative examples selection policies*. One is the *all* approach, which uses all instances to train all local classifiers, and *siblings*, which uses the instances belonging to a node's class and its siblings' classes to train the local classifiers. A comparison of different negative example selection approaches is made in [9] and [11]. The results indicate that both approaches have roughly similar performances, however, *siblings* is more scalable than *all*.

*Local Classifier Per Parent Node (LCPN):* In this approach, a local multilabel classifier is learned for every non-leaf node in the hierarchy. The labels are the direct child nodes and the training instances are those which belong to the parent node class. If each multilabel problem is transformed into a set of binary problems with the binary relevance method, this is equivalent to local classifier per node. Depending on the choice of the local multilabel classifier, it is possible to model dependencies between sibling nodes.

*Local Classifier Per Level (LCL):* This is the type of classifier approach least used so far on the literature. The local classifier per level approach consists of training one multilabel classifier for each level of the class hierarchy. That means it is prone to class-membership inconsistency and




(a) LCN (b) LCPN

(c) LCL

Figure 2: Hierarchical multilabel classification local classifier approaches

therefore requires a post-processing step to prevent it. In the literature this approach was only mentioned as a possible approach by [12], and used as a baseline comparison method in [6] and [7]. Moreover, there is no publicly available implementation of this kind of approach.

Figure 2 illustrates the difference between the three local classifier approaches. The dashed closed curves indicate the set labels of each local classifier. In the case of LCN (2a), each of the eleven local binary classifiers predicts whether an instance belongs to its correspondent class or not. For LCPN (2b), there are five local multilabel classifiers, whose labels are sibling nodes. For LCL (2c), there are three local multilabel classifiers, whose labels are the nodes of each level of the hierarchy.

### 2.2.2 Global Classifier Approach

In contrast to local classifier approaches, the global classifier approach (also known as *big bang approach*), learns one single classification model built from the training set, taking into account the class hierarchy as a whole during a single run of the classification algorithm. When used during the prediction phase, each instance is classified by the induced model, a process that can assign classes at potentially every level of the hierarchy to the instance. Global classifier approaches lack the kind of modularity for local training of the classifier that is a core characteristic of the local classifier approaches.

An example of global classifier approach is MLC4.5 [5], which is a decision tree algorithm adapted to handle multilabel data. A single decision tree is created for the classifier, where each leaf node contains a vector with the class distributions. This method guarantees consistency with the hierarchy, as the probability of a class in the class distribution cannot be smaller than that of its children. Therefore, for any probability threshold, the generated prediction will be consistent with the hierarchy.

## 2.3 Evaluation Measures

Silla Jr. et al. [37] recommend the use of hierarchical loss (*h-loss*), and the hierarchical precision ($hP$), recall ($hR$), and F-measure ($hP$) to evaluate hierarchical multilabel classifiers. In this paper, we also use the hamming loss (*hamm*), which is commonly used in (non-hierarchical) multilabel classification and serves as basis for the *h-loss*.

The $hP$, $hR$, and $hF$ [18] are the micro-averaged measures of precision, recall and F-measure per class. By using the micro average, each class is weighted according to the label frequencies. Equation 3 shows the definition of these measures, where $tp_i$, $fp_i$ and $fn_i$ denote respectively the number of true positives, false positives and false negatives of the class $c_i$. Similarly to their binary class versions, $hP$, $hR$ and $hF$ values range is in the interval $[0, 1]$.

$$hP = \frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fp_i} \tag{1}$$

$$hR = \frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fn_i} \tag{2}$$

$$hF_\beta = \frac{(\beta^2 + 1) \cdot hP \cdot hR}{\beta^2 \cdot hP + hR} \tag{3}$$

Equation 4 shows the Hamming loss (*hamm*) for one instance. We denote the true label vector of an instance as $y$, and the predicted vector as $\hat{y}$, with $y_i = 1$ if the instance is of class $c_i$, $y_i = 0$ otherwise. Hamming loss reports how many times on average, a class label is incorrectly predicted, i.e., the number of false positives and false negatives, normalized over total number of classes and total number of examples.

$$l_h(\hat{y}, y) = \sum_{i=1}^{|C|} 1_{\hat{y}_i \neq y_i} \tag{4}$$

$$hl_H(\hat{y}, y) = \sum_{i=1}^{|C|} 1_{\hat{y}_i \neq y_i} \max_{\{j | c_i \sqsubseteq c_j\}} 1_{\hat{y}_j = y_j} \tag{5}$$

Equation 5 shows the hierarchical loss (*h-loss*) [4] for one instance, which extends hamming loss to account for any existing underlying hierarchical structure of the labels. The idea of hierarchical loss is based on the notion that, whenever a classifier makes a mistake at any node in a given hierarchy, no further loss should be counted for any mistake in the subtree rooted at that particular node ignoring any subtree which is rooted at a wrong prediction node.

# 3. PROBLEM DEFINITION

The problem studied in this paper is the prediction of instance types on RDF data, where the types are organized in a hierarchy. Untyped instances and instances with incomplete set of types are a common problem in Semantic Web knowledge bases [27], therefore, we need methods which can automatically predict types of instances and, at the same time, are able to handle noisy data, which is a common problem in such datasets. In this paper, we assume that the type hierarchy is correct, and we restrict the hierarchical structure to trees for simplicity and because DAGS are not supported by multilabel classification libraries.

We model the type prediction task as a hierarchical multilabel classification problem, which we define according to the categorization proposed in [37]. The classification problem is defined as $< T, MPL, PD >$, which means that the type

of graph representing the class hierarchy is a tree ($T$), instances are allowed to have multiple paths of labels ($MPL$), and that instances are allowed to have partial depth ($PD$) labeling (i.e., non-mandatory leaf node prediction).

The partial depth labeling is important in our problem because in many cases the class hierarchy is incomplete, requiring an instance which cannot be typed with any leaf node to be assigned a more general type. In Fig. 1, `Arnold_Schwarzenegger` is neither an `AdultActor` nor a `VoiceActor`, i.e., none of the specializing classes of `Actor` is appropriate. Thus, the instance should be typed as an `Actor`, which is a non-leaf node. Supporting multipath labels is also relevant because many instances might have multiple labels which are not in the same path in the hierarchy. In the same example, `Arnold_Schwarzenegger` is labeled with `OfficeHolder`, `BodyBuilder`, `Actor`, and their generalizations, and thus has three paths in the hierarchy.

Although our problem definition does not support DAGs, they can be transformed into trees by selecting (e.g., at random, or by leveraging a priori distributions) a single parent for nodes with multiple parents. This simplifies the hierarchy, but leads to an information loss, which could result in a drop in the quality of the predictions.

The extraction of features for the classifier is also an important part of the problem addressed in this paper. Different datasets might have domain specific features highly valuable for the type prediction. The extraction of features from knowledge bases is a problem which deserves an exclusive study. Therefore in this paper we focus on general features which can be extracted from any RDF knowledge base.

# 4. APPROACH

The problem of type prediction in RDF data requires highly scalable approaches which can handle a high number of labels, features, and instances inherent to many Semantic Web datasets. In this paper, we propose a more scalable version of a local classifier per node approach which we call *SLCN*.

In our approach, we assume that the knowledge base has a type hierarchy which is materialized in the dataset, i.e., if an instance is assigned a given type, it must also be assigned all its superclasses. If the hierarchy is is not materialized, we perform simple reasoning to infer the assertions of all superclasses absent in the dataset by exploiting the `subClassOf` relations.

## 4.1 Algorithm

SLCN is based on the local classifier per node (LCN) with top-down prediction approach and *siblings* negative examples selection policy. This means that we train one binary classifier for every class $c_i \in C$, and each of those classifiers is trained on a local transformed dataset with a binary class label (belongs to the type: $y_c = 1$, or not: $y_c = 0$). The top-down prediction approach means that when predicting the types of a given instance, we first classify the instance for the types in the highest level. For all the types which the instance is predicted to belong to, the local classifiers of its subtypes predict if the instance belongs to any of its children, and so forth. Whenever the instance is predicted not to belong to a given type, then it is assumed that it does not belong to any of its subtypes either, therefore there is no need to run the local classifiers of the children nodes.

Assuming that a hierarchical multilabel classifier is perfect and correctly predicts all classes and we want to, for example, predict the types of the instance `Arnold_Schwarzenegger`. The classifier would first predict it is a `Person`. Then it would predict that it also belongs to its three subtypes `Artist`, `OfficeHolder` and `Athlete`. Following the `Artist` branch, it would then predict that it belongs to `Actor` and does not belong to `Painter`, and finally that it does not belong to either `AdultActor` or `VoiceActor`. Following the `Athlete` branch, the classifier would predict it belongs to `BodyBuilder`, and does not belong to `MotorsportRacer`. The local classifiers for the subtypes `MortorcycleRider` and `RacingDriver` would not need to make any prediction since the instance does not belong to their supertype `MotorsportRacer`, and therefore, in order to be consistent with the hierarchy, cannot belong to any of its children.

The top-down approach ensures that the outcome of the multilabel classifier is consistent with the type hierarchy. However, it can cause the *blocking problem* [38], which may occur during the top-down process of classifying a test example. The classifier at a certain level in the class hierarchy predicts that the example in question does not have the class associated with that classifier. In this case the classification of the example will be blocked, i.e., the example will not be passed to the descendants of that classifier.

As scalability is an important factor in the problem studied in this paper, we choose to use the *siblings* negative examples policy, which reduces the sizes of local training datasets for classes in the lower levels of the hierarchy. The local training sets are created including the instances belonging to the target class as positive examples and the instances belonging to its sibling classes as negative examples. For instance, the transformed dataset with *siblings* for the type `BodyBuilder` would contain as negative examples the instances belonging to `MotorsportRacer` and, because we allow partial-depth prediction, the instances which belong to `Athlete` but not of its children.

Typically, the number of labels in the lower levels of the hierarchy is higher, and the lower the level of the label node, the smaller the subset is. Assuming that the label hierarchy has a fanout $b$, and the instances have a single path only, the average transformed dataset size would be $|D| * (b * log_b(|C|))/|C|$ instead of $|D|$. The average size of the transformed datasets also increases with the number of different paths instances have. However, for simplicity and because the average number of different paths per instance is low in most real datasets, we ignore this factor when calculating the average size.

In the proposed approach, local feature selection, sampling, and class balancing are performed for every local classifier. The intuition is that in each binary subproblem, where for the instances of a given class we predict whether they belong to a subclass, not all the features might be relevant. Especially in cross-domain datasets, such as DBpedia, YAGO, and Wikidata, the set of features required to predict, for instance, if an `Athlete` is a `MotorsportRacer` is completely different from those required to predict if an `Infrastructure` is an `Airport`. This allows the local classifier to handle a smaller set of locally relevant features instead of a larger set with all the features.

Moreover, we choose to use the *filter* instead of the *wrapper* feature selection method, where we calculate the information gain of each feature and select the top-$k$ most rele-

vant features ranked by information gain. Since the idea of local feature selection is to reduce the training time of the local classifiers, it only makes sense to perform the feature selection if its complexity is lower than that of the classifier training. Hence, we decide to use a simple feature selection method, whose complexity grows linearly with the number of features.

For the local sampling, we set a maximum local training sample size $n$. The idea is that if a local classifier has a number of instances smaller than the maximum training sample size, no sampling is performed, so that the training set does not lose any valuable instances. On the other hand, local classifiers with a high number of instances, such as those for the classes in higher level of the hierarchy, will be trained on a smaller sample of size $n$, reducing the time required for training the local classifier. When sampling the data, potential class imbalance can be addressed individually for each class in its transformed dataset. For that, we define a bias to uniform class distribution $u \in [0, 1]$, where $u = 0$ means that the class distribution is left as it is, and $u = 1$ means that class weights are assigned values that result in an uniform class distribution. With that, the classifier settings can be defined by the triple $< k, n, u >$. In the experiments discussed in section 5.2, we evaluate the influence of each of these parameters on the performance of SLCN.

One limitation of SLCN is that it does not support disjointness between classes, since it assumes independence between sibling nodes. However, at the moment, most knowledge bases do not contain class disjointness axioms. Approaches which can model dependencies between classes, such as MLC4.5 and LCPN with ECC or LPW, should be able to handle such disjointness even if not explicitly defined in the ontology. When training the classifier, since we use the siblings negative examples selection policy, we implicitly use the closed world assumption in order to generate negative labels for the local classifiers. This can be a problem on datasets where the type assertions are highly incomplete [26].

## 4.2 Features

Every instance $x \in X$ in the classification dataset is a typed entity in the knowledge base. The set $C$ contains the instance types, which are the labels in our classification problem. In this paper, we propose the extraction of *binary* features, following [30], which are not specific to a dataset at hand, but applicable on any general SW knowledge base. $R$ is the set of outgoing relations, $R'$ is the set of ingoing relations, $Q$ is the set of qualified relations, i.e., pairs of outgoing relations and object types, and $Q'$ is the set of ingoing qualified relations.

The feature sets for the classification problem are extracted with the following SPARQL queries, where the keyword `a` is used as a shorthand notation for `rdf:type`:

$R$: select distinct ?p where {?x ?p ?z, ?x a ?c}
$R'$: select distinct ?p where {?z ?p ?x, ?x a ?c}
$Q$: select distinct ?p ?t where {?x ?p ?z, ?z a ?t, ?x a ?c}
$Q'$: select distinct ?p ?t where {?z ?p ?x, ?z a ?t, ?x a ?c}

In our experiments, we define the set of features used in a type prediction task as $F$, which may consist of any combination of the features sets $R$, $R'$, $Q$ or $Q'$. While in SLCN it is possible to include dataset specific features, such as DBpedia categories or text features extracted from Wikipedia abstracts, in this paper, we concentrate on general features which can be extracted from *any* SW knowledge base. It is worth mentioning that, in contrast to SDType [28] and other existing methods, which usually rely on a certain kind of features, the proposed hierarchical multilabel classification approaches can handle any kind of features which could be extracted from knowledge bases, and is thus more versatile.

In the future we plan to perform experiments with different kinds of features and propositionalization strategies [34], and evaluate how they affect the predictive performance. However, since in this paper we focus on the prediction methods and not the feature extraction, we restrict ourselves to the features described previously.

# 5. EXPERIMENTS

The experiments are divided into three main parts. In the first one, we evaluate the performance of different local classifiers for SLCN, and different parameter values for $< k, n, u >$. In the second part, we compare SLCN to SDType and different state-of-the-art multilabel classifiers, analyzing performance and scalability with respect to the number of instances, features, and labels. We do not compare the proposed approach to RDFS reasoning, because it has already been shown to outperform reasoning in the case of real-world SW knowledge bases [28]. Finally, in the last part, we make a comparison on different large-scale RDF datasets.

For our experiments, we use MULAN 1.5, which is an open-source Java library for learning from multilabel datasets based on WEKA [41]. It includes a variety of state-of-the-art multilabel classification algorithms, and offers multilabel feature selection and evaluation. Apart from SDType, we compare SLCN to the local approaches HMC, HOMER, and the global approach MLC4.5 [5]. HMC is an implementation of the LCPN approach. HOMER [39] is similar to HMC, but it uses balanced clustering to generate a hierarchy for flat labels, where the non-leaf nodes are meta-labels identifying label clusters. MLC4.5 and SDType were re-implemented in the MULAN framework. The performance of SDType is very sensitive to the chosen confidence threshold, and the optimal threshold may vary with the used dataset. Therefore, for our experiments, we added an extra step to the training phase of SDType in order to find the confidence threshold which maximizes the $hF$ measure. Apart from that, all methods were used with their standard settings in MULAN.

## 5.1 Datasets

In our experiments, we use four different datasets: DBpedia, DBpedia with YAGO types, NELL, and Wikidata[1]. Because MULAN can only handle trees, not arbitrary DAGs, we convert all DAG type hierarchies to trees by retaining only the subsumption relation of the most frequent parent node. Table 1 shows some statistics about the different datasets, including number of instances, percentage of instances with partial-depth (PD) and multipath (MPL) labels, number of labels ($|C|$), and size of the different feature sets ($|R|$, $|R'|$, $|Q|$, $|Q'|$). In the next paragraphs we briefly discuss relevant characteristic of each dataset used.

[1]The datasets used are available for download at http://dws.informatik.uni-mannheim.de/en/research/hmctp

| Dataset | Instances | MPL | PD | $\|C\|$ | $\|R\|$ | $\|R'\|$ | $\|Q\|$ | $\|Q'\|$ |
|---|---|---|---|---|---|---|---|---|
| DBpedia | 4 218 125 | 0.02% | 32.6% | 476 | 1390 | 659 | 30 423 | 10 427 |
| DBp(YAGO) | 2 886 305 | 81.4% | 86.2% | 454 | 1308 | 638 | 61 595 | 45 484 |
| NELL | 120 720 | 8.3% | 4.6% | 264 | 259 | 246 | 2357 | 2762 |
| Wikidata | 19 254 100 | 63.4% | 18.4% | 474 | 1324 | 474 | 53 175 | 119 207 |

Table 1: Statistics about the datasets used

| classifier | rt($ms$) | *h-loss* | *hamm* | *hP* | *hR* | *hF* |
|---|---|---|---|---|---|---|
| J48 | 111 711 | **2.51±0.20** | **0.01±0.00** | **0.51±0.04** | **0.50±0.02** | **0.50±0.01** |
| NaiveBayes | **56 692** | 2.85±0.11 | 0.01±0.00 | 0.45±0.01 | 0.42±0.02 | 0.43±0.01 |
| AdaboostM1 | 104 238 | 2.62±0.14 | 0.01±0.00 | 0.48±0.02 | 0.43±0.02 | 0.45±0.00 |
| LibSVM | 880 441 | **2.51±0.24** | **0.01±0.00** | **0.52±0.05** | 0.47±0.03 | 0.49±0.01 |

Table 2: Comparison of different local classifiers on SLCN

*DBpedia:* We use DBpedia 2014[2] with mapping-based properties. There are two main issues with existing DBpedia type assignments. The first is that DBpedia only contains single path labels, although it is clear that several instances, such as `Arnold_Schwarzenegger`, should belong to multiple paths. This happens because of its extraction framework, which maps infoboxes to types and assigns an instance the type of the first infobox of its Wikipedia page. That makes it an exception amongst other main RDF datasets which, as Table 2 illustrates, have a significant portion of its instances with multipath labels. The second problem is that the correct type can be trivially predicted from outgoing properties, as reported in [29], which happens because the DBpedia outgoing properties and types are generated in one step from the same original information. Therefore, in our experiments, we use only the feature sets $R'$ and $Q'$ for DBpedia. DBpedia 2014 has a class hierarchy which is not a tree only because of the class `Library`, which is a subclass of `EducationalInstitution` and `Building`. All the other classes have a single parent class. In order to convert it to a tree, we choose `EducationalInstitution` to be the only superclass, following the tree depicted by the DBpedia Ontology browser.[3] Since DBpedia types were originally materialized with the DAG hierarchy, after the transformation to a tree all, the 816 instances of `Library` (0.02% of the total) appear to have two paths in the tree.

*DBpedia with YAGO types:* Unlike DBpedia, YAGO[4] extracts its type hierarchy from Wikipedia categories. Because of that, it has a staggering 384 174 different types and a complex DAG type hierarchy. Moreover, YAGO has a very limited number of properties, which results into a small number of features. With the number of labels much greater than the number of features, the YAGO dataset as it is, is not well suited for the type prediction problem. Most of the DBpedia instances are linked to the YAGO types. Therefore it makes sense to combine both datasets by using DBpedia features and YAGO types as labels. With that, the problem of DBpedia's exclusively single-path labels, which are extracted together with the outgoing properties, can be ruled out. The problem of the high number of YAGO labels can be solved by simply choosing the top-$k$ most frequent types. In our experiments, arbitrarily select the 474 most frequent YAGO labels.

*Wikidata:* Similarly to what was done to the YAGO types, in Wikidata, which also has a large original $|C| = 29099$, we arbitrarily select the 454 most frequent types. In contrary to the other knowledge bases used in the experiments, Wikidata does not rely on information extraction methods to generate its RDF graph. Wikidata is part of the Wikimedia community and its pages contain structured data, which can be exported to RDF format [10]. Its type hierarchy is a DAG, and in order to transform it into a tree, for all types with multiple parents we keep the `subClassOf` relation with parent with greatest number of instances and delete the rest.

*NELL:* We use the NELL dataset (08m.690), which has originally 1 168 998 instances. The properties are very sparse, and 89.7% of the instances have just the property `haswikipediaurl` or none. Therefore, in our dataset we remove these instances and use only the other 10.3%.

## 5.2 SLCN Parameter Setting Experiments

We conduct a first experiment to evaluate the performance of different types of local classifiers on our approach. Four different popular binary classifiers available in WEKA are evaluated. Table 2 reports the results of the comparison, which was performed on a random sample of the DBpedia data with YAGO types containing 28 863 instances (1% of the total) and features $F = P \cup P'$. The results indicate that J48 (an implementation of the C4.5 decision tree algorithm) and LibSVM perform equally well in terms of prediction quality, with J48 being about eight times faster than SVM. Thus, we use J48 as a base classifier in the subsequent experiments.

In the experiments, we evaluate how the three parameters $k$, $n$ and $u$ (i.e., the number of features, the local training sample sizes, and the bias to uniform class distribution) affect the performance of SLCN. The evaluation is performed on the same sample described before, using J48 as local classifier and the default setting $k = 100$, $n = 500$. We then vary $k$ and $n$ measuring the runtime as well as $hP$, $hR$, $hF$, *h-loss* and *hamm*.

The plots in Figure 3 show $hF$ and *runtime* for different parameter values. It is notable that for both the number of features $k$ and maximum train set size $n$, the $hF$ curves flatten after a certain point, while the runtime curves continue to grow. The optimal values for $n$ and $k$ depend on characteristics of the data, and may vary from dataset to dataset.

As the local classification problems can be rather skewed, we have also performed experiments with different sampling biases towards a more uniform class distribution in the local sampling. Since SLCN is based on LCN with *siblings* negative example selection, the classes are not as imbalanced in the local training sets as they are in the whole dataset. Moreover, we select the most frequent classes from Wikidata and YAGO, which excludes the the smallest classes, and hence avoids the most skewed local classification problems. Therefore, the sampling bias to uniform class distribution does not significantly affect the performance of SLCN, i.e., we stick to stratified sampling in our experiments.

## 5.3 Scalability Experiments

In this section, we compare the scalability of the methods in terms of the number of instances, number of features, and

[2]http://wiki.dbpedia.org/Downloads2014
[3]http://mappings.dbpedia.org/server/ontology/classes/
[4]http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/

number of labels of a dataset. The experiments were conducted on the same sample of DBpedia with YAGO types described in the previous section. To vary the number of instances, we randomly sample instances as training set and progressively increase the sample size, for the number of features we select features with highest information gain first, and for the number of labels we select the most frequent labels first.

Figure 4 shows the runtime and hF of each method for different number of instances, number of features and number of labels. SDType is the most scalable of the compared methods, however, its $hF$ was significantly lower than all the other compared methods. The runtime of SLCN is close to that of SDType, improving the runtime in comparison to the other hierarchical multilabel classifiers, and improving $hF$ in comparison to SDType. MLC4.5 has the best overall $hF$, however, in terms of runtime, it does not scale as well as SDType and SLCN. It is particularly noteworthy that the runtime of SDType and SLCN is not much dependent on the number of instances, features, and labels, which is not the case for the competing approaches.

## 5.4 Large-Scale Experiments on SW Datasets

In this section, we perform large-scale experiments on whole RDF datasets. Table 3 shows the results of 5-fold cross validation on the RDF datasets presented earlier. Because of time limitation, we do not report the results for classifiers which require more than a week for training. HMC, HOMER and MLC4.5 were able to finish only on NELL, therefore for the other datasets in Table 3 we report the results only for SDType and SLCN. On the other hand, SLCN was able to finish for all datasets, showing the effectiveness of the proposed approach in improving scalability.

On the NELL dataset, the HMC, HOMER and MLC4.5 perform better than SDType and SLCN. However, the runtime of the first three methods are notably longer than the others. When comparing SLCN against SDType, the former performs consistently better with respect to all evaluation measures, but longer runtime. Note that the results of SDType differ from those reported in [28] because the latter includes `owl:Thing` and classes in other ontologies, such as FOAF and schema.org, in the evaluation, while we exclude them. On all the other datasets, which are signif-

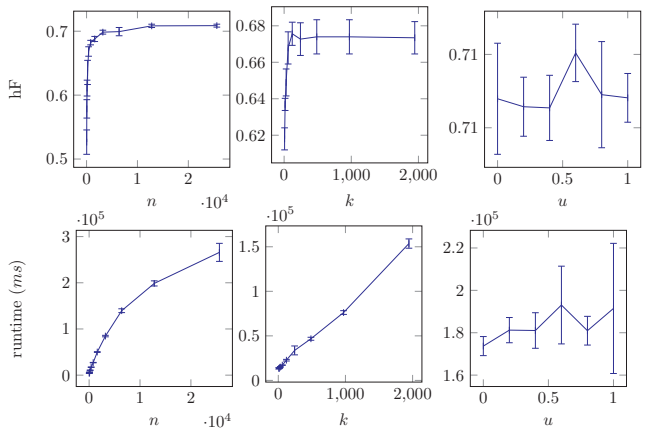



Figure 3: Evaluation of the impact of the parameters $n$ and $k$ on $hF$ and runtime.

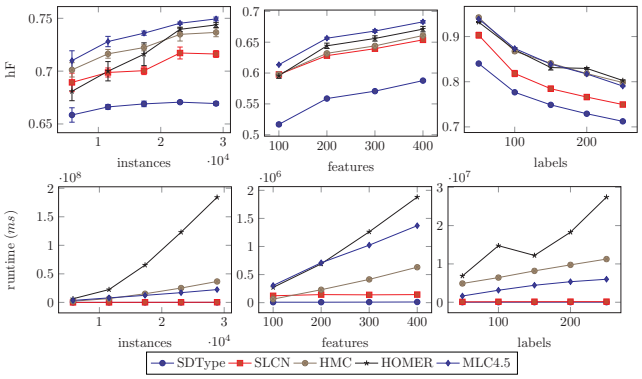



Figure 4: Scalability in terms of number of instances, features and labels

| Dataset | Method | $hF$ | $h$-loss | $hamm$ | rt($ms$) |
|---|---|---|---|---|---|
| DBpedia $F = R^l$ | SDType | 0.765±0.002 | 0.773±0.008 | 0.003±0.000 | 16 080 553 |
| | SLCN | **0.847±0.001** | **0.463±0.005** | **0.002±0.000** | **7 024 255** |
| DBpedia $F = R^l \cup Q^l$ | SDType | 0.770±0.000 | 0.750±0.001 | 0.003±0.000 | 54 511 659 |
| | SLCN | **0.846±0.001** | **0.461±0.005** | **0.002±0.000** | **10 154 987** |
| DBp(YAGO) $F = R \cup R^l$ | SDType | 0.666±0.000 | 2.672±0.002 | 0.016±0.000 | 6 744 282 |
| | SLCN | **0.703±0.007** | **2.097±0.089** | **0.013±0.001** | **7 635 499** |
| DBp(YAGO) $F = R \cup R^l \cup Q \cup Q^l$ | SDType | 0.671±0.000 | 2.648±0.002 | 0.016±0.000 | 213 904 335 |
| | SLCN | **0.702±0.006** | **2.106±0.090** | **0.013±0.001** | **48 374 257** |
| Wikidata $F = R \cup R^l$ | SDType | 0.753±0.000 | 0.575±0.000 | 0.002±0.000 | 208 957 224 |
| | SLCN | **0.812±0.011** | **0.375±0.009** | **0.001±0.000** | **44 807 901** |
| Wikidata $F = R \cup R^l \cup Q \cup Q^l$ | SDType | 0.776±0.000 | 0.519±0.000 | 0.002±0.000 | 272 206 437 |
| | SLCN | **0.868±0.003** | **0.271±0.006** | **0.001±0.000** | **64 413 619** |
| NELL $F = R \cup R^l$ | SDType | 0.544±0.001 | 2.443±0.005 | 0.022±0.000 | **61 074** |
| | SLCN | 0.603±0.008 | 1.297±0.045 | 0.011±0.000 | 495 571 |
| | HMC | **0.646±0.002** | **1.215±0.005** | **0.011±0.000** | 22 194 619 |
| | HOMER | **0.646±0.013** | 1.331±0.094 | **0.010±0.001** | 205 837 211 |
| | MLC4.5 | 0.643±0.002 | 1.329±0.004 | **0.011±0.000** | 28 298 930 |

Table 3: Evaluation of different classification methods on SW datasets

icantly larger than NELL (c.f. Table 1), SLCN is the best overall performer as HMC, HOMER and MLC4.5 were not able to finish in less than one week.

The use of qualified relation features ($Q$ and $Q'$) substantially increase the dimensionality of the feature space, as it can observed in Table 1, and therefore the runtime is also increased. SDType is able to improve its results when considering the greater set of features for DBpedia and DBpedia with YAGO types, but SLCN actually yield slightly worse results. This may be because of a possibly higher level of dependency between the features in $Q$ and $Q'$. Since the filter feature selection method does not take dependencies between features into account, the selected feature set could contain several redundant features.

# 6. RELATED WORK

The problems of inference on noisy data in the Semantic Web have been identified, e.g., in [16] and [32]. There have been solutions proposed for the specific problem of type inference in (general or particular) RDF datasets in the recent past, using strategies such as machine learning, statistical methods, and exploitation of external knowledge such as links to other data sources or textual information. One of the first approaches to type classification in relational data

is discussed in [22]. The authors train a machine learning model on instances that already have a type, and apply it to the untyped instances in an iterative manner.

Some works address slightly different inference problems. Instead of predicting instance types, [25] predict possible predicates for resources based on co-occurrence of properties. The approach discussed in [31] addresses the problem of mapping DBpedia entities to the category system of OpenCyc. They use DBpedia specific information – infoboxes, textual descriptions, Wikipedia categories and instance-level links to OpenCyc – and apply an a posteriori consistency check using Cyc's own consistency checking mechanism.

HYENA [42] is a multi-label classifier for named entity types based on hierarchical taxonomies derived from YAGO. Textual features extracted from the mentions of the named entity Wikipedia articles are used in by the classifier, which consists of the SCN approach with siblings negative examples selection. Thus, it can only be applied to Semantic Web knowledge bases that are linked to Wikipedia.

There are several works on type prediction which exploit specific aspects of DBpedia. In [1], an approach is introduced which first exploits cross-language links between DBpedia in different languages to increase coverage. Then, they use nearest neighbor classification based on different features, such as templates, categories, and bag of words of the corresponding Wikipedia article. The *Tipalo* system [14] leverages the natural language descriptions of DBpedia entities to infer types, exploiting the fact that most abstracts in Wikipedia follow similar patterns. Those descriptions are parsed and mapped to the WordNet and DOLCE ontologies in order to find appropriate types. The authors of [15] exploit types of resources derived from linked resources, where links between Wikipedia pages are used to find linked resources (which are potentially more than the resources actually linked in DBpedia). For each resource, they use the classes of related resources as features, and use $k$-nearest neighbors for predicting types based on those features. However, none of those approaches can be trivially applied to datasets other than DBpedia.

SDType [28] uses links between resources as indicators for types, namely the ingoing and outgoing properties of instances. The method requires the prior distribution of types, as well as, for every property, a conditional probability distribution of object and subject types. Every property is assigned a weight, where maximum weight is given to properties that appear with a single type only, while the minimum weight is given to properties which are equally present in all types. Based on that, when predicting the types of an instance, SDType computes a confidence value for every type possible. Those types whose confidence value satisfy an arbitrarily defined minimum confidence threshold are assigned to the instance's prediction.

SDType is a simple and highly scalable method, whose complexity grows linearly with the number of statements in the knowledge base. According to the algorithm categorization by Silla Jr. et al [37], SDType can be considered a global hierarchical multilabel classifier with multipath, non-mandatory leaf-nodes. SDType also generates predictions consistent with the type hierarchy. That is because the confidence of any non-root class will be always smaller or equal to that of its superclass.

The SLCN approach proposed in this paper relies on the idea of class specific features for local classifiers on multi-label classification, which has been also been exploited by LIFT [43]. However, in their setting, instead of performing local feature selection, the authors propose a method for generation of class specific features based on the distance of instances to the centroid of clusters computed for the positive and negative examples. Since this approach requires a clustering algorithm to be executed twice for each class, its application to large-scale knowledge graphs would lead to massive scalability issues.

Amongst the approaches discussed above and in [28], SDType is reportedly the best performing approach for the problem addressed in this paper [28, 29], also outperforming RDFS reasoning. Therefore, in our experiments, we have restricted ourselves to comparing hierarchical classification methods against SDType. The evaluations in the previous section have shown that SLCN, as proposed in this paper, clearly outperforms SDType (and thereby also many other approaches, including RDFS reasoning, which are themselves outperformed by SDType).

Statistical relation learning is an area which has a lot in common with type prediction. In fact, type prediction can be considered a special case of the link prediction problem where instances are linked with types. According to [23], statistical relational learning models can be divided into latent feature and graph feature models or a combination of both.

Graph feature models extract features from the directly observed edges in the knowledge graph. These include ILP based methods, such as ALEPH [21] and AMIE [13], similarity based methods, such as Katz Index [17], Local Random Walks [20], and Path Ranking Algorithm [19]. The main disadvantage of these methods is that they can use exclusively graph features, therefore leaving relevant text features and numerical properties unexploited. Our proposed approach, on the other hand, is able to use any kind of features.

Latent feature models derive the relationships between features from the interactions between their latent features. RESCAL [24], TransE [3] and multiway neural networks (mwNN) [8] are some of the state-of-the-art methods in link prediction. There are no reported results for type prediction using link prediction approaches in the literature, therefore we cannot directly compare the results presented in the respective papers with our method. In the future, we plan to evaluate these methods for the type prediction task and compare them with our proposed approach.

One interesting aspect of latent feature models is the general low-dimensional representations of entities, which could also be used as features in our proposed approach. Employing instance embeddings as features would probably reduce training time because of their low dimensionality, however, computing these embeddings is expensive. A more detailed study would be necessary in order to find out whether it the time spent computing is worth the gain in training time, and how it affects the predictive performance.

# 7. CONCLUSION AND FUTURE WORK

In this paper, we have modeled the type prediction problem in Semantic Web knowledge bases as a hierarchical multilabel classification problem. We propose SLCN, and compare it both to popular hierarchical multilabel classifiers and the state-of-the-art type prediction approach SDType (which is currently one of the strongest and best scalable algorithms for the task at hand) for SW knowledge bases.

The experiments indicate that the local feature selection and local sampling can significantly improve scalability without sacrificing performance, and they also show that SLCN can perform better than SDType, and scales better than the other multilabel classifiers evaluated in this paper.

In the future, as our approach assumes independence between sibling classes, we plan to consider a post processing to take disjointness axioms into account. Combining our approach with specific feature selection methods for Semantic Web datasets [35] would be a promising refinement. We also plan to evaluate the performance of our approach when exploiting dataset specific features, such as DBpedia categories and NLP features from abstracts. Furthermore, we want to adapt our approach to support arbitrary DAGs as type hierarchies and investigate the impact it has on the quality of the predictions and runtime. Finally, we plan to exploit the parallelism potential of the SLCN in order to further improve the scalability and develop a Spark implementation of the studied approaches in MULAN. We expect that a distributed implementation could allow us to perform type prediction on Linked Data.

## Acknowledgements

The work presented in this paper has been partly supported by the Ministry of Science, Research and the Arts Baden-Württemberg in the project SyKo$^2$W$^2$ (Synthesis of Completion and Correction of Knowledge Graphs on the Web).

## 8. REFERENCES

[1] A. P. Aprosio, C. Giuliano, and A. Lavelli. Automatic expansion of DBpedia exploiting Wikipedia cross-language information. In *10th Extended Semantic Web Conference (ESWC 2013)*, 2013.

[2] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics*, 7(3):154–165, 2009.

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc., 2013.

[4] N. Cesa-bianchi, L. Zaniboni, and M. Collins. Incremental algorithms for hierarchical classification. In *Journal of Machine Learning Research*, pages 31–54. MIT Press, 2004.

[5] A. Clare and R. D. King. Knowledge discovery in multi-label phenotype data. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD'01, pages 42–53, London, UK, 2001. Springer-Verlag.

[6] A. Clare and R. D. King. Predicting gene function in saccharomyces cerevisiae. *Bioinformatics*, 19:42–49, 2003.

[7] E. P. Costa, A. C. Lorena, A. C. P. L. F. Carvalho, A. A. Freitas, and N. Holden. Comparing several approaches for hierarchical classification of proteins with decision trees. In *Proceedings of the 2nd Brazilian Conference on Advances in Bioinformatics and Computational Biology*, BSB'07, pages 126–137, Berlin, Heidelberg, 2007. Springer-Verlag.

[8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 601–610, New York, NY, USA, 2014. ACM.

[9] R. Eisner, B. Poulin, D. Szafron, P. Lu, and R. Greiner. Improving protein function prediction using the hierarchical structure of the gene ontology. In *Proc. IEEE CIBCB*, 2005.

[10] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, and D. Vrandečić. Introducing Wikidata to the linked data web. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandečić, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, editors, *Proceedings of the 13th International Semantic Web Conference (ISWC'14)*, volume 8796 of *LNCS*, pages 50–65. Springer, 2014.

[11] T. Fagni and F. Sebastiani. On the selection of negative examples for hierarchical text categorization. In *In Proceedings of The 3rd Language Technology Conference*, pages 24–28, 2007.

[12] A. Freitas and A. C. de Carvalho. *A Tutorial on Hierarchical Classification with Applications in Bioinformatics.*, volume Research and Trends in Data Mining Technologies and Applications, chapter VII, pages 182–196. Idea Group, January 2007.

[13] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 413–422, New York, NY, USA, 2013. ACM.

[14] A. Gangemi, A. G. Nuzzolese, V. Presutti, F. Draicchio, A. Musetti, and P. Ciancarini. Automatic typing of DBpedia entities. In *11th International Semantic Web Conference (ISWC 2012)*, 2012.

[15] A. Giovanni, A. Gangemi, V. Presutti, and P. Ciancarini. Type inference through the analysis of wikipedia links. In *Linked Data on the Web (LDOW)*, 2012.

[16] Q. Ji, Z. Gao, and Z. Huang. Reasoning with noisy semantic data. In *The Semanic Web: Research and Applications (ESWC 2011), Part II*, pages 497–502, 2011.

[17] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[18] S. Kiritchenko, S. Matwin, and A. F. Famili. Functional annotation of genes using hierarchical text categorization. In *in Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05*, 2005.

[19] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Mach. Learn.*, 81(1):53–67, Oct. 2010.

[20] W. Liu and L. Lü. Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007, 2010.

[21] S. Muggleton. Inverse Entailment and Progol. *New*

*Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[22] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[23] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[24] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 809–816, New York, NY, USA, June 2011. ACM.

[25] E. Oren, S. Gerke, and S. Decker. Simple algorithms for predicate suggestions using similarity and co-occurrence. In *European Semantic Web Conference (ESWC 2007)*, pages 160–174. Springer, 2007.

[26] H. Paulheim. Exploiting linked open data as background knowledge in data mining. In *International Workshop on Data Mining on Linked Data (DMoLD*, 2013.

[27] H. Paulheim. Knowlegde Graph Refinement: A Survey of Approaches and Evaluation Methods. *Semantic Web Journal*, 2016. to appear.

[28] H. Paulheim and C. Bizer. Type inference on noisy rdf data. In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *International Semantic Web Conference (1)*, volume 8218 of *Lecture Notes in Computer Science*, pages 510–525. Springer, 2013.

[29] H. Paulheim and C. Bizer. Improving the quality of linked data using statistical distributions. *Int. J. Semant. Web Inf. Syst.*, 10(2):63–86, Apr. 2014.

[30] H. Paulheim and J. Fürnkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, page 31. ACM, 2012.

[31] A. Pohl. Classifying the wikipedia articles in the opencyc taxonomy. In *Web of Linked Entities Workshop (WoLE 2012)*, 2012.

[32] A. Polleres, A. Hogan, A. Harth, and S. Decker. Can we ever catch up with the web? *Semantic Web Journal*, 1(1,2):45–52, 2010.

[33] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD'09, pages 254–269, Berlin, Heidelberg, 2009. Springer-Verlag.

[34] P. Ristoski and H. Paulheim. A comparison of propositionalization strategies for creating features from linked open data. In *Linked Data for Knowledge Discovery*, 2014.

[35] P. Ristoski and H. Paulheim. Feature selection in hierarchical feature spaces. In *Discovery Science*, 2014.

[36] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[37] C. N. Silla, Jr. and A. A. Freitas. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.*, 22(1-2):31–72, Jan. 2011.

[38] A. Sun, E.-P. Lim, W. K. Ng, and J. Srivastava. Blocking reduction strategies in hierarchical text classification. *IEEE Trans. Knowl. Data Eng.*, 16(10):1305–1308, 2004.

[39] G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, 2008.

[40] G. Tsoumakas, I. Katakis, and I. Vlahavas. Random k-labelsets for multi-label classification. *IEEE Transactions on Knowledge and Data Engineering*, 99(1), 2010.

[41] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

[42] M. A. Yosef, S. Bauer, J. Hoffart, M. Spaniol, and G. Weikum. HYENA: hierarchical type classification for entity names. In *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Posters, 8-15 December 2012, Mumbai, India*, pages 1361–1370, 2012.

[43] M. Zhang and L. Wu. Lift: Multi-label learning with label-specific features. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(1):107–120, 2015.

[44] M. Zhang and Z. Zhou. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18:1338–1351, 2006.

[45] M.-L. Zhang and Z.-H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recogn.*, 40(7):2038–2048, 2007.