International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

Type Prediction in Noisy RDF Knowledge Bases using Hierarchical Multilabel Classification with Graph and Latent Features

Andre Melo

University of Mannheim
B6 26 C1.06, 68168 Mannheim, Germany
andre@informatik.uni-mannheim.de

Johanna Völker

University of Mannheim
B6 26 C1.09, 68168 Mannheim, Germany
johanna@informatik.uni-mannheim.de

Heiko Paulheim

University of Mannheim
B6 26 C1.09, 68168 Mannheim, Germany
heiko@informatik.uni-mannheim.de

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Semantic Web knowledge bases, in particular large cross-domain data, are often noisy, incorrect, and incomplete with respect to type information. This incompleteness can be reduced, as previous work shows, with automatic type prediction methods. Most knowledge bases contain an ontology defining a type hierarchy, and, in general, entities are allowed to have multiple types (classes of an instance assigned with the `rdf:type` relation). In this paper, we exploit these characteristics and formulate the type prediction problem as hierarchical multi classification, where the labels are types. We evaluate different sets of features, including entity embeddings, which can be extracted from the knowledge graph exclusively. 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 with the objective of improving scalability. Furthermore, we explore different variants of creating features for the classifier, including both graph and latent features. We compare the performance of our proposed method with the state-of-the-art type prediction approach and popular hierarchical multilabel classifiers, and report on experiments with large-scale cross-domain RDF datasets.

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, defined by the `rdf:type` relation, being one of the atomic building blocks of knowledge bases. Many datasets suffer from type assertion incompleteness. For example, for DBpedia[1], 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[2]. For example, `Arnold_Schwarzenegger` in DBpedia is assigned only the type `OfficeHolder` and none of the many other suitable types, such as `Actor` and `BodyBuilder`.

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, since they do not rely on the quality of the T-box axioms and are less influenced by single wrong A-box axioms. Therefore, they are considered to be more suitable for the type prediction task[3].

One example that illustrates this problem is that in DBpedia the Album entity `Abbey_Road` from the Beatles, is confused with the Musical Studio entity `Abbey_Road_Studios` in some triples. The assertion `No_Reply_(song) recordedIn Abbey_Road` would lead to the inference that `Abbey_Road` is not only an Album but also a Populated Place, which is the range of the property `recordedIn`. In [3], we have shown that RDFS reasoning is prone to propagate noise, whereas heuristic inference is suitable to limit the influence of noise. Therefore, in this paper, we pursue the use of heuristic inference methods.

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 *multilabel* 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[4]. Second, the computational cost of training a multilabel classifier may be strongly affected by the number of labels[5].

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[6]. 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 SW knowledge bases.

In this paper, we propose *SLCN* (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.

This paper is an extended version of a conference paper presented at the 6th International Conference on Web Intelligence, Mining and Semantics (WIMS 2016)[7]. It extends the conference paper by exploring the influence of local and global feature selection in the classification on both the result quality and the performance, and by comparing the predictive performance of graph-based and latent features.

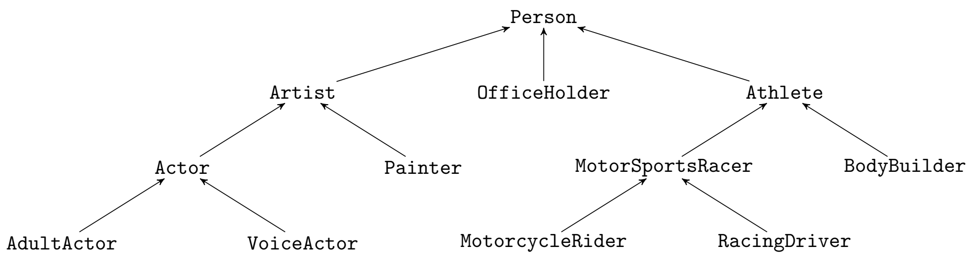


Fig. 1: A subset of the DBpedia type hierarchy

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 belong to 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*[8], *MLkNN*[9] and *BPMLL*[10]. Other approaches, such as *Binary Relevance* (BR),*Classifier Chains*[4] (CC), *Label Powerset* (LP), and Random k-Labelsets (RAKeL)[11], 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*[4] (CC) and *Label Powerset* (LP), can model dependencies between the classes. There are also ensemble methods, such as Ensembles of Classifier Chains (ECC)[4] and Random k-Labelsets (RAKeL)[11], 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 Silla et al.[12], 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 considers 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 subproblems. This is a kind of transformation approach, since for every subproblem works the local classifier is trained on a different transformed dataset. According to[12], there are mainly three approaches of using local information: *local classifier per node*, *local*

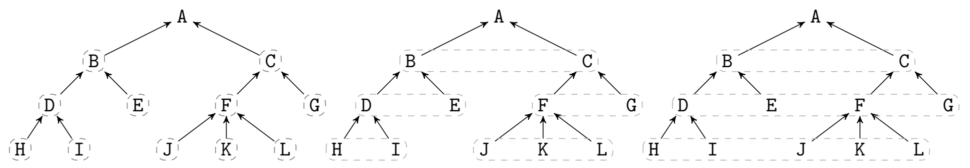


Fig. 2: Hierarchical multilabel classification local classifier approaches

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[13] and[14]. 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 therefore requires a post-processing step to prevent it. In the literature this approach was only mentioned as a possible approach by[15], and used as a baseline comparison method in[16] and[17]. 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[18], 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. [12] 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[19] 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. Equations 1, 2 and 3 show 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 $[0, 1]$.

$$hP = \frac{\sum\limits_{i=1}^{|C|} tp_i}{\sum\limits_{i=1}^{|C|} (tp_i + fp_i)} \tag{1}$$

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

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

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*)[20] 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

In a knowledge graph, not every instance may come with proper type information. Untyped instances and instances with incomplete set of types are a common problem in Semantic Web knowledge bases[6], therefore, we need methods which can automatically predict types of instances. Thus, the task of type inference is to assign types to untyped instances, as well as adding types to instances with incomplete type information. To that end, available information about the instance, such as its relation to other instances, is used to train an inference model.

At the same time, not all information in the RDF data set may be correct; there can be various types of errors[21], including wrong relations between instances[2,22], wrong interlinks between datasets[23], and wrong literal values[24,25], among others. Thus, when training a classifier for predicting missing types, those errors will shine up as *noise* in the respective training set, and require a noise-tolerant learning approach. 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.

In particular, types in RDF knowledge bases come are organized in hierarchies. Hence, we model the type prediction task as a hierarchical multilabel classification problem, which we define according to the categorization proposed in[12]. 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 in theory 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 we focus on graph and latent features which can be extracted from any RDF knowledge graph.

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 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.

When considering a non-mandatory leaf-node prediction problem, the class-prediction top-down approach has to use a stopping criterion that allows an example to be classified just up to a non-leaf class node. We follow the approach proposed by Wu et al. [26], where 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*[27], 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. Sun et at. [27] propose methods for addressing this this problem (e.g., such as threshold reduction, restricted voting, and extended multiplicative thresholds). However, the results show that, although the proposed methods can reduce the blocking problem, they also cause a degradation in precision. Therefore, in our approach we decide not to use any of these approaches.

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 its sibling class `MotorsportRacer` and, because we allow partial-depth prediction, the instances which belong to its superclass `Athlete` but none 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 (c.f. average number of paths per instance (ANP) in Table 1), 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 features relevant to all classes.

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 relevant features ranked by information gain. Since the idea of local feature selection is to reduce the training time of the local classifiers, it makes more 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.

The benefit of local feature selection cannot be achieved using global multilabel classification methods, since a single model is learned on a single set of selected features, which has to be relevant for all the classes. Selecting features globally might lead to preferring features relevant to the most frequent classes, and potentially leaving out features which are relevant to less frequent classes. In Section 5.3 we conduct experiments which show that SLCN performs better with local feature selection than with global feature selection.

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.

Algorithm 1 shows the pseudo code of SLCN indicating the main characteristics of the approach. The sampling and feature selection (SAMPLE_INSTANCES and SELECT_FEATURES) is performed for every class $c \in C$ and incorporated into the training of SLCN. In the algorithm X is the features matrix, where instances are represented by rows, and columns are features, Y is the labels matrix, also with instances represented as rows, a $|C|$ columns represent the types, h is a map containing all the nodes of the type hierarchy, $local\_feats$ is map containing the sets of selected features for each class, and $local\_clfs$ a map of the local binary classifiers of each class.

The function PREDICT_RECURSION illustrate how the top-down prediction approach stopping criterion works, which we implement by keeping a set of indices i to the instances to which the local classifiers should be applied.

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[28].

4.2. *Features*

It is not possible to directly apply hierarchical multilabel classification methods on knowledge bases. In order to be able to work with classifiers, we need to represent every typed instance $x \in X$ in the knowledge base as a feature vector, and the types of the given instance as labels. In this paper we focus on general features which can be extracted from the relations between entities represented in a knowledge graph. Therefore, we do not investigate features extracted from external sources and text. According to Nickel et al.[29], two kinds of features can be obtained from knowledge graphs: latent features and graph features. The former consists of features which cannot be directly observed in the graph, often these are lower-dimensional representations (also known as embeddings) of entities. The latter consists of features that can be directly observed from the edges in the graph.

Latent features are dense lower dimensional representations of entities in an

Algorithm 1 SLCN pseudocode

```
1: $local\_clfs \leftarrow \emptyset$
2: $local\_feats \leftarrow \emptyset$
3:
4: function TRAIN($X, Y, h$)
5:     for $c \in C$ do
6:         $n \leftarrow$ number of instances in $X$
7:         if HAS_PARENT(h[c]) then
8:             $i \leftarrow \{i | Y[i, h[c].parent] = 1\}$        ▷ Select instances that belong to parent
9:         else
10:            $i \leftarrow \{1, ..., n\}$
11:        end if
12:        $X_{local} \leftarrow X[i]$
13:        $Y_{local} \leftarrow Y[i, c]$
14:        $X_{local}, Y_{local} \leftarrow$ SAMPLE_INSTANCES($X_{local}, Y_{local}, n$)
15:        $local\_feats[c] \leftarrow$ SELECT_FEATURES($X_{local}, Y_{local}, k$)
16:        $X_{local} \leftarrow X_{local}[:, local\_feats[c]]$
17:        $local\_clfs[c] \leftarrow$ TRAIN_LOCAL_CLASSIFIER($X_{local}, Y_{local}$)
18:    end for
19: end function
20:
21: function PREDICT_RECURSION($X, Y, h, n, i$)
22:    $X_{local} \leftarrow X[:, local\_feats[n]]$
23:    $Y[i, n] \leftarrow$ PREDICT_LOCAL_CLASSIFIER($local\_clfs[n], X_{local}$)
24:    i $\leftarrow \{i | Y[:, n] = 1\}$
25:    for $c \in h[n].children$ do
26:        Y $\leftarrow$ PREDICT_RECURSION($X, Y, h, c, i$)
27:    end for
28:    return Y
29: end function
30:
31: function PREDICT($X, h$)
32:    $n \leftarrow$ number of instances in $X$
33:    $Y \leftarrow$ ZEROS($n, |C|$)        ▷ Initialize the matrix $Y$ with zeros (empty predictions)
34:    for $r \in$GET_ROOTS($h$) do
35:        $Y \leftarrow$ PREDICT_RECURSION($X, Y, h, r, \{1, ..., n\}$)
36:    end for
37:    return $Y$
38: end function
```

embedding space. The existent latent multirelational learning models, often used for the link prediction problem, generate this kind of entity representation. Since in these models every entity on the knowledge graph is represented by embeddings, we can use these embeddings as features and types as labels and train a multilabel classifier for type prediction. Assuming that in the latent feature representations, entities of same type are located close to each other in the embeddings space, these features should in principle be useful for the type prediction task.

Some of the state-of-the-art latent feature models include TransE[30], TransR[31], RESCAL[32], multiway neural networks (mwNN)[33] and Holographic Embeddings (HolE)[34]. In this paper we choose to use Holographic Embeddings (HolE)[34], which learns compositional vector space representations of entire knowledge graphs. The proposed method is related to holographic models of associative memory in that it employs circular correlation to create compositional representations. HolE is efficient to compute, easy to train, and highly scalable, but at the same time it is highly expressive and can model complex relations. This achieved by using circular correlation as the compositional operator (c.f. Equation 6), which is illustrated by the 3-dimensional example for c in Equation 7. It is shown that holographic embeddings are able to outperform state-of-the-art methods, such as TransE and RESCAL, for link prediction in knowledge graphs and relational learning benchmark datasets.

Circular correlation, also known as compressed tensor product, enables relations to be modeled as d-dimensional vectors instead of $d \times d$ matrices like in RESCAL. With that the amount of memory for representing relations is reduced from $O(d^2)$ to $O(d)$ and the runtime for learning the model is reduced from $O(d^2)$ to $O(d \log d)$[34].

$$c = a \star b = \sum_{0}^{d-1} a_i b_{(k+1) mod d} \tag{6}$$

$$\begin{aligned} c_0 &= a_0 b_0 + a_1 b_1 + a_2 b_2 \\ c_1 &= a_0 b_2 + a_1 b_0 + a_2 b_1 \\ c_2 &= a_0 b_1 + a_1 b_2 + a_2 b_0 \end{aligned} \tag{7}$$

An important characteristic of the circular correlation as an operator to model relations is that it is noncommutative, i.e., $a \star b \neq b \star a$, which enables it to model asymmetric relations. In HolE, the probability of a triple (s, p, o) is modeled as shown in Equation 8, where e_s and e_o are the embedding representations of the subject s and object o, and r_p is a vector that represents the relation p, and σ is the logistic function, that converts the triple score to a probability.

$$\sigma(r_p^\top (e_s \star e_o)) \tag{8}$$

In this paper, we will call the set of entity embedding features E, where E has d dimensions. In general, d is small (in this paper we use $d \in 5, 10, 25, 50, 150$), but the feature vectors are dense, i.e. they have few zeros.

Graph features can be directly extracted from the triples in the graph, and therefore are simpler to be obtained. In this paper, we propose the extraction of *binary* features, following[35], which are not specific to a dataset at hand, but applicable on any general SW knowledge base. R_{out} is the set of outgoing relations, R_{in} is the set of ingoing relations, Q_{out} is the set of qualified relations, i.e., pairs of outgoing relations and object types, and Q_{in} is the set of ingoing qualified relations, i.e., pairs of ingoing relations and subject types. For convenience, we define the set of ingoing and outgoing relations as $R = R_{\text{in}} \cup R_{\text{out}}$ and the set of all qualified relations as $Q = Q_{\text{in}} \cup Q_{\text{out}}$.

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`:

$$\begin{array}{lcl} R_{\text{out}} & : & \text{select distinct ?p where \{?x ?p ?z, ?x a ?c\}} \\ R_{\text{in}} & : & \text{select distinct ?p where \{?z ?p ?x, ?x a ?c\}} \\ Q_{\text{out}} & : & \text{select distinct ?p ?t where \{?x ?p ?z, ?z a ?t, ?x a ?c\}} \\ Q_{\text{in}} & : & \text{select distinct ?p ?t where \{?z ?p ?x, ?z a ?t, ?x a ?c\}} \end{array}$$

In theory, on a knowledge base containing relations $p_i \in P$, and $c_i \in C$ types, R_{out} and R_{in} can have up to $|P|$ dimensions, and Q_{out} and Q_{in} up to $|P| \times |C|$ dimensions. In practice, however, R_{in} is often smaller than R_{out} because, for example, data properties, which have literals as objects, appear always as outgoing relations of entities and never as ingoing. Also, because of domains and range restrictions of relations, many combinations of outgoing relation and object type, as well as ingoing relation and subject type never occur, therefore the sizes of Q_{out} and Q_{in} often are significantly smaller than $|P| \times |C|$. It is important to note that, in contrast to the dense entity embeddings feature set E, the feature vectors Q and R are in practice highly sparse with few non-zero entries. These characteristics can be observed in Table 1, which give the size of these feature sets on the datasets used in this paper.

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_{out}, R_{in}, Q_{out}, Q_{in} or E. 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[3] 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[36], 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.

In summary, the contributions of this paper are the formulation of the type prediction as a hierarchical multilabel classification problem and the proposal of SLCN, a scalable hierarchical multilabel classifier based on the local classifier per node approach which exploits local feature selection and sampling in order to improve scalability and facilitate the use of such approach on large cross-domain LOD datasets. Moreover, we investigate how the use of different feature sets affect the performance of SLCN and other methods. We consider embedding features obtained with HolE, as well as ingoing and outgoing relations and qualified relations.

5. Experiments

The experiments are divided into five main parts. First, we evaluate the performance of different local classifiers for SLCN, and different parameter values for $< k, n, u >$. The second part compares local and global feature selection, showing that local feature selection performs better. Therefore, in the following, local feature selection is used throughout all the experiments. Third, we compare the influence of graph and latent features, showing that the latter only lead to a marginal performance improvement. In the fourth 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[3]. 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[37]. 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[18]. HMC is an implementation of the LCPN approach. HOMER[5] 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 SD-Type 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 large scale cross-domain datasets: DB-pedia, DBpedia with YAGO types, NELL, and Wikidata[a]. We also use AIFB and Mutagenesis datasets, which are two smaller and simpler domain-specific datasets. They are especially needed for the latent features experiments, since the computation of entity embeddings for the large scale datasets can be expensive. 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 multi-path (MPL) labels, average number of paths per instance (ANP),number of labels ($|C|$), and size of the different feature sets ($|R_{\text{out}}|$, $|R_{\text{in}}|$, $|Q_{\text{out}}|$, $|Q_{\text{in}}|$). In the next

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

paragraphs we briefly discuss relevant characteristic of each dataset used.

Dataset	Instances	MPL	ANP	PD	$\lvert C\rvert$	$\lvert R_{\text{out}}\rvert$	$\lvert R_{\text{in}}\rvert$	$\lvert Q_{\text{out}}\rvert$	$\lvert Q_{\text{in}}\rvert$
DBpedia	4 218 125	0.02%	1.02	32.6%	476	1390	659	30 423	10 427
DBp(YAGO)	2 886 305	81.4%	3.16	86.2%	454	1308	638	61 595	45 484
NELL	29 317	38.9%	1.10	32.2%	264	248	248	2721	3056
Wikidata	19 254 100	63.4%	1.64	18.4%	474	1324	474	53 175	119 207
AIFB	27 100	48.3%	1.48	10.8%	58	81	81	287	538
Mutagenesis	14 157	0%	1.00	0%	87	4	4	118	14

Table 1: Statistics about the datasets used

DBpedia: We use DBpedia 2014[b] 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 to 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[2], 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_{in} and Q_{in} 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.[c] 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[d] 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 relations ($|R_{\text{out}}| = 37$) with most relations on the people and location domain), which results into a small number of features which are biased to specific classes. 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

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

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[38]. Its type hierarchy is a DAG, and in order to transform it into a tree, for all types with multiple parents we keep the `rdfs:subClassOf` relation with parent with greatest number of instances and delete the rest.

NELL: We use the NELL dataset (version 08m.690), which has originally 1 168 998 instances. However, the NELL's graph is highly sparse with only around 2% of instances being both typed and having at least one ingoing or outgoing relation. Therefore for the datasets used in our type prediction experiments we remove the other 98% of instances and use only the remaining 29317.

AIFB: The AIFB dataset[e] describes the AIFB research institute in terms of its staff, research group, and publications. The data is an export of the AIFB website and contains around 270 thousand triples. The type hierarchy is originally a wide and shallow tree with average fanout 14.25 and average depth 2.04.

Mutagenesis: The MUTAG dataset is distributed as an example dataset for the DL-Learner toolkit[f]. It contains information about 340 complex molecules that are potentially carcinogenic, which is given by the isMutagenic property. The molecules can be classified as "mutagenic" or "not mutagenic", and the main entity types atoms bonds and compounds which define the molecules. The type hierarchy is also originally a tree with average fanout 7.17 and average depth 2.49.

5.2. *SLCN Base Classifier and Parameter Settings*

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 $F = R$. 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.

[e]`http://www.aifb.kit.edu/web/Web_Science_und_Wissensmanagement/Portal`
[f]`http://dl-learner.org`

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

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

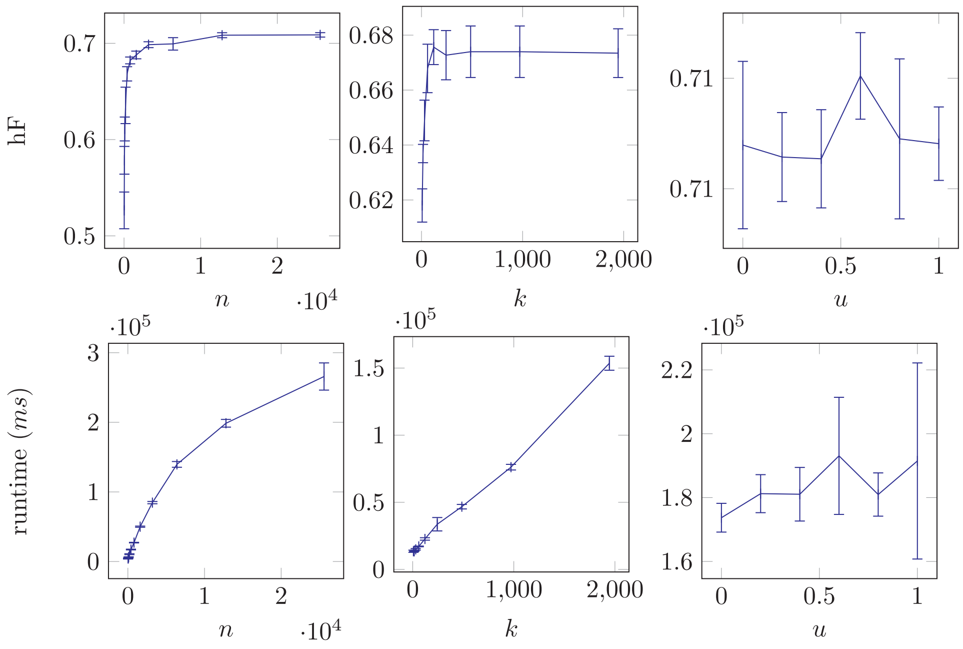


Fig. 3: Evaluation of the impact of the parameters n and k on hF and runtime.

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 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. *Local vs. Global Feature Selection*

Transformation-based multilabel classifiers learn different models for each transformed dataset. In the case of SLCN, there's one transformed dataset and one learned model for every type. This characteristic allows the feature selection to be performed either globally or locally.

In the global feature selection, the selection is performed on the original dataset, which means all the transformed datasets share the same set of selected features, and the features selected are those most relevant to all the classes. In the local feature selection, the selection is performed on each transformed dataset, which allows the sets of selected features to be specialized for each transformed dataset class.

In this experiment we compare the local and global feature selection approaches on the LCN approach with siblings negative examples selection policy, which is the base of our proposed approach SLCN. As described in Section 4, we used the filter method for the feature selection, which selects the top-k most relevant features without considering dependencies between features, but is highly scalable. In our experiments we use information gain as relevance measure.

Figure 4 shows a comparison of hF and runtime between the local and global strategies with SLCN. The local approach shown in red performs consistently better than the global approach on all datasets in terms of hF. Moreover, the runtime is also lower than the global approach. The feature selection has to be performed only once in the global approach, while on the local approach it has to be performed $|C|$ times. However, on the global approach the relevance measures require the computation of $|C|$-dimensional distributions over all the classes, while on the local approach it is computed for a single class. The local approach is faster because in the feature selection is performed on the transformed datasets, and, because of the *siblings* negative examples selection policy, the relevance of the features is calculated on datasets which on average are smaller.

The drop in runtime at the last data point for the experiments on AIFB and Mutagenesis happens because the time we report includes the training time and the feature selection. Since in the last data point the number of features is the same as the original number of features in the dataset, no feature selection is required. For smaller datasets, the time required for performing the feature selection is not paid off by the reduction in training time.

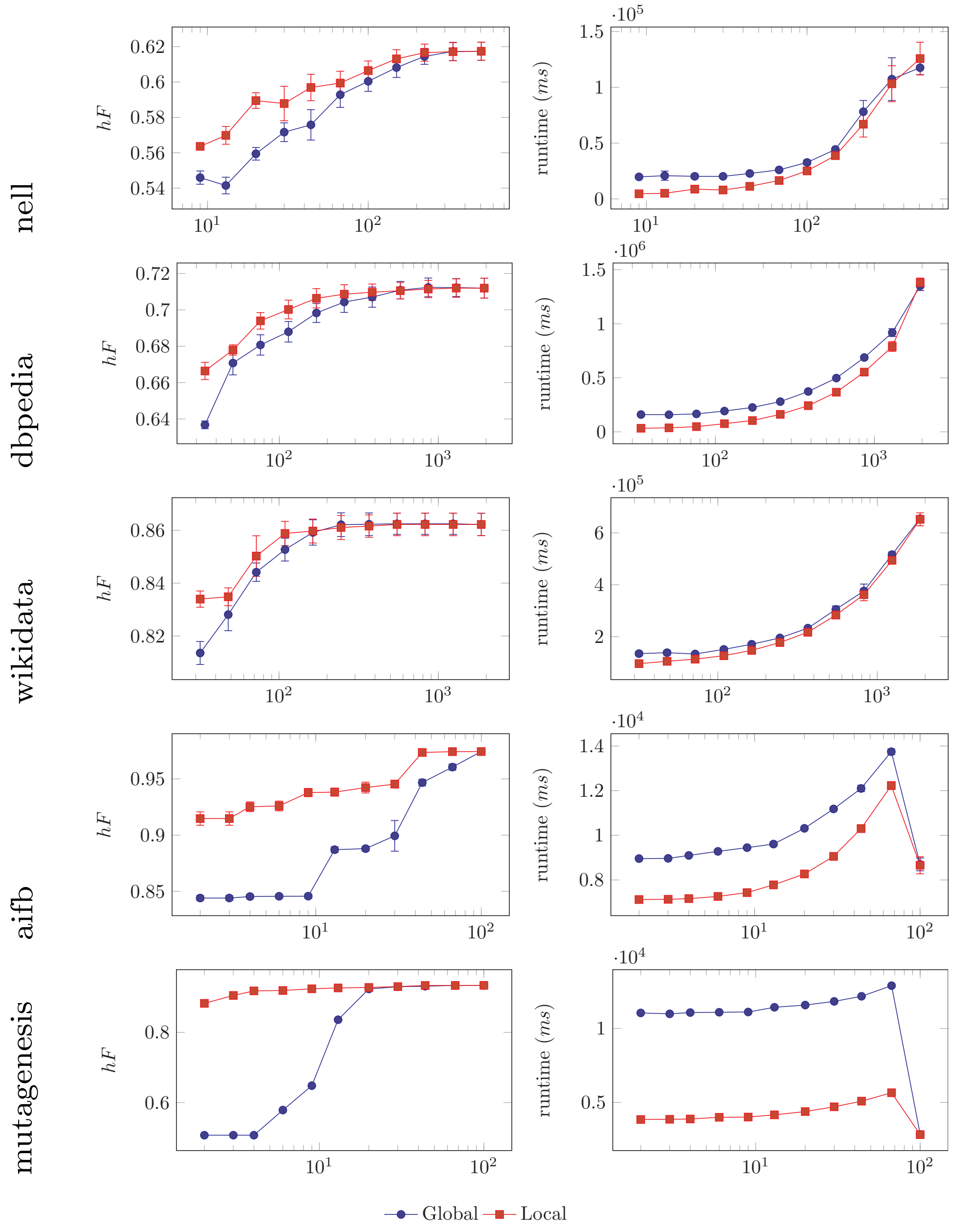
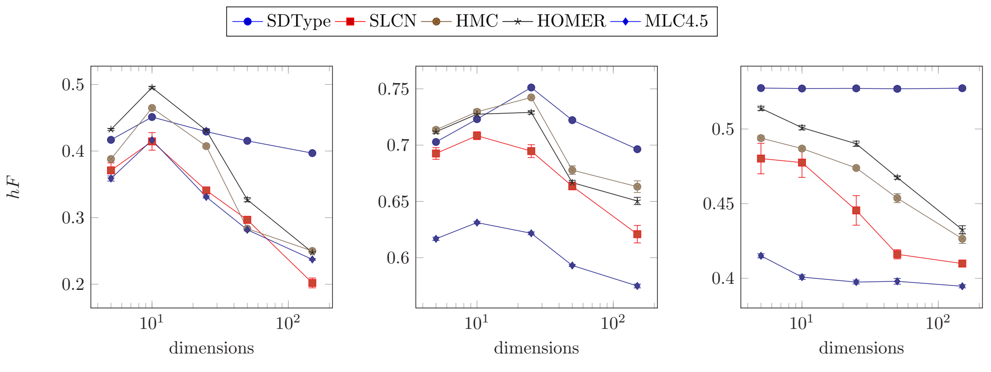


Fig. 4: Global vs local feature selection comparison with SLCN.

5.4. *Graph Features vs. Latent Features*

In this section, we perform a comparison between latent features and graph features for type prediction. In these experiments we use HolE features, which were learned



(a) Mutagenesis (b) AIFB (c) NELL

Fig. 5: Type prediction results for different number of HolE embeddings dimensions

on the whole datasets excluding the `rdf:type` relations. HolE provides state-of-the-art performance with an expressive, yet simple and scalable model, which enables us to learn the model on larger datasets[34]. We use the HolE implementation from the python library Scikit-KGE[g]. We learn holographic embeddings with the parameter settings recommended by the authors, and we vary the number of dimensions in order to find the optimal value on the smaller datasets AIFB and Mutagenesis.

Firstly, for the smaller datasets AIFB and Mutagenesis as well as NELL we make learn HolE embeddings with different number of dimensions $d \in 5, 10, 25, 50, 150$, and evaluate how good the learned embeddings are as features for the type prediction task. Figure 5 shows how the hF is affected by the dimensionality of the embeddings, for the Mutagenesis dataset 10 was the optimal value, for AIFB it was 25, and for NELL it was 5. Since both datasets are fairly simple, we can observe that for the higher number of dimensions, the model overfits. This, however, should not be a problem for the other datasets, which are much more complex and contain more relations.

We evaluate the performance of hierarchical multilabel classifiers for type prediction with five different feature sets: qualified ingoing and outgoing relations (Q), ingoing and outgoing relations (R), HolE embeddings only (E), combination of R and HolE embeddings ($R \cup E$), and combination of Q and HolE embeddings ($Q \cup E$).

The objective is to evaluate the relevance of entity embeddings to the type prediction task, and how they compare to other traditional graph features, as well as to examine if any improvement can be obtained by combining these two kinds of features. Since SDType cannot handle the real valued numerical features from E, for this type prediction method we discretize the numerical attributes into 25 bins using the equal frequencies approach, resulting in $25d$ binary features.

The results indicate that the best set of features is the qualified relations Q, which over all classifiers significantly improves the performance over the set of in-

[g]`https://github.com/mnick/scikit-kge`

Dataset	Method	Q	R	E	$R \cup E$	$Q \cup E$
AIFB	HMC	**0.987±0.001**	0.950±0.001	0.742±0.001	0.944±0.002	**0.987±0.001**
	MLC4.5	**0.988±0.001**	0.950±0.001	0.636±0.003	0.915±0.002	0.977±0.002
	SLCN	**0.976±0.004**	0.938±0.003	0.721±0.006	0.922±0.002	**0.976±0.004**
	HOMER	**0.987±0.001**	0.949±0.000	0.729±0.002	0.937±0.002	**0.987±0.001**
	SDType	**0.803±0.004**	0.737±0.004	0.751±0.001	0.771±0.001	0.784±0.001
Mutagenesis	HMC	**0.934±0.002**	0.761±0.003	0.465±0.004	0.761±0.003	**0.934±0.002**
	MLC4.5	**0.905±0.003**	0.680±0.002	0.417±0.006	0.600±0.003	0.890±0.003
	SLCN	**0.926±0.005**	0.762±0.008	0.415±0.032	0.758±0.008	**0.926±0.005**
	HOMER	**0.934±0.002**	0.778±0.004	0.495±0.004	0.778±0.004	**0.934±0.002**
	SDType	**0.737±0.002**	0.726±0.006	0.451±0.003	0.461±0.003	0.527±0.002
NELL	HMC	0.870±0.002	**0.917±0.004**	0.487±0.002	**0.919±0.002**	0.870±0.002
	MLC4.5	0.912±0.002	**0.955±0.001**	0.401±0.004	0.868±0.002	0.778±0.004
	SLCN	0.761±0.004	**0.896±0.004**	0.479±0.007	0.884±0.003	0.737±0.008
	HOMER	0.868±0.016	**0.925±0.004**	0.501±0.003	**0.925±0.001**	0.867±0.016
	SDType	0.892±0.003	**0.903±0.003**	0.527±0.002	0.559±0.004	0.609±0.003

Table 3: Comparison of hF for type prediction on different feature sets

going and outgoing relations R. The exception for that is NELL, where Q performs worse than R. This happens because the knowledge graph is highly incomplete, where several entities have no types or no ingoing or outgoing properties. For the classification dataset we select only the entities which have at least one type and at least one ingoing or outgoing relation, however, when extracting the features Q some of the objects of outgoing and subjects of ingoing relations are not typed entities, therefore we cannot use them as features in Q.

The latent features E, when used alone perform significantly worse than all other feature sets, indicating that this kind of features is not very relevant for the type prediction task. We also combine the embeddings E with R, in order to evaluate if the embeddings can add relevant information and improve performance. However, the experiments indicate that in some cases it does not significantly affect the hF measure, while in others it actually acts as noisy, reducing the quality of the predictions.

Although the HolE embeddings have been shown to be useful in the link prediction problem, in the type prediction problem they do not seem to be of significant relevance when comparing to the graph features evaluated in this paper. This can be attributed to the fact that these entity embeddings are learned with the objective of modeling links between entities, and not to separate them by types.

On the other hand, with R it is possible to exploit distributions of object and subject types for each relation, and based on these links predict the type, and with Q a more detailed version with conditional distributions of object types given subject type as well as subject types given object type for each relation. These kinds of features provide more sophisticated information about the domain and ranges of relations than those normally expressed by RDF Schema. The conditional distributions of object and subject types are obtained from the A-box and do not rely on the quality of domain and range axioms provided by the T-box, which is a crucial problem of reasoning methods.

5.5. *Scalability Experiments*

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

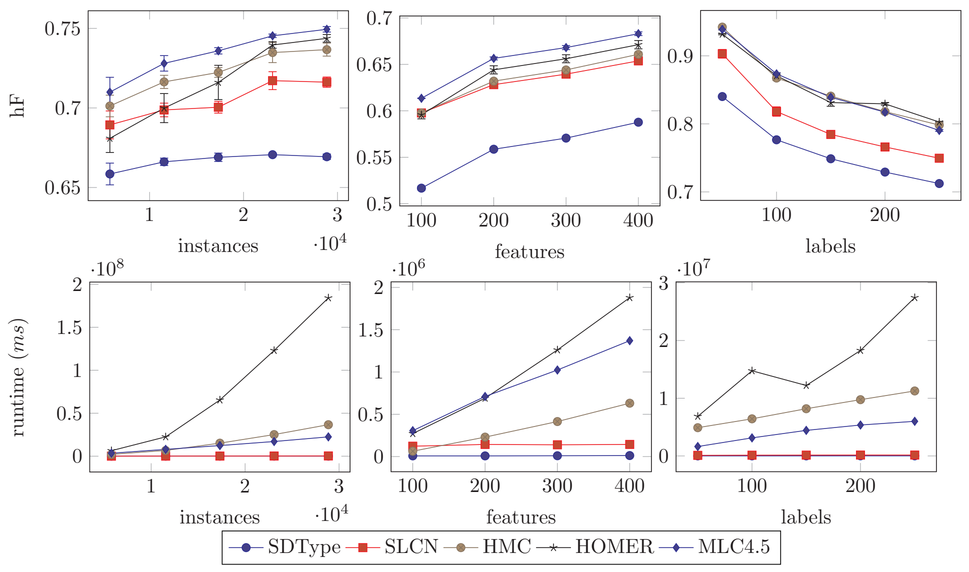


Fig. 6: Scalability in terms of number of instances, features and labels

Figure 6 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 significantly more scalable than the other approaches in terms of number of instances, features, and labels.

Although in the plots SDType and SLCN seems not to change its runtime with the number of instances, features and labels, their runtime is of course also affected. However, in comparison with the other methods, the increase in runtime is much smaller and cannot be visualized in the plot. The runtime of SLCN would grow similarly to its local classifier for number features smaller than k and instances smaller than n. For larger values, the number of instances and features of the local datasets the would remain constant, and the increase in runtime would be determined by the instance sampling and feature selection methods used.

5.6. *Large-Scale Experiments on SW Datasets*

In this section, we perform large-scale experiments on whole RDF datasets. Table 4 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 4 we report the results only for SDType and SLCN, which were able to finish for all datasets, showing the effectiveness of the proposed approach in improving scalability.

Dataset	Method	*hF*	*h-loss*	*hamm*	rt(ms)
DBpedia $F = R_{in}$	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 $F = R_{in} \cup Q_{in}$	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) $F = R_{out} \cup R_{in}$	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) $F = R_{out} \cup R_{in} \cup Q_{out} \cup Q_{in}$	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 $F = R_{out} \cup R_{in}$	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 $F = R_{out} \cup R_{in} \cup Q_{out} \cup Q_{in}$	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 $F = R_{out} \cup R_{in}$	SDType	0.903±0.003	0.484±0.013	0.004±0.000	**1 917 946**
	SLCN	0.896±0.004	0.585±0.015	0.005±0.000	2 547 871
	HMC	0.917±0.004	0.501±0.013	0.004±0.000	6 336 452
	HOMER	0.925±0.004	0.480±0.014	0.003±0.000	10 957 195
	MLC4.5	**0.955±0.001**	**0.331±0.009**	**0.002±0.000**	16 991 440
NELL $F = Q_{out} \cup Q_{in}$	SDType	0.892±0.003	**0.445±0.009**	0.005±0.000	**9 289 373**
	SLCN	0.761±0.004	1.081±0.018	0.010±0.000	18 252 489
	HMC	0.870±0.002	0.686±0.077	0.006±0.000	102 815 535
	HOMER	0.868±0.016	0.687±0.117	0.006±0.001	156 444 313
	MLC4.5	**0.912±0.002**	0.484±0.004	**0.004±0.000**	166 477 106

Table 4: Evaluation of different classification methods on large cross-domain SW datasets

Dataset	Method	*hF*	*h-loss*	*hamm*	rt(*ms*)
AIFB $F = R$	SDType	0.737±0.004	0.884±0.006	0.016±0.000	**80 892**
	SLCN	0.938±0.003	0.230±0.006	0.005±0.000	88 997
	HMC	**0.950±0.001**	**0.198±0.004**	**0.004±0.000**	442 235
	HOMER	**0.949±0.000**	0.198±0.003	**0.004±0.000**	387 434
	MLC4.5	0.950±0.001	0.199±0.006	**0.004±0.000**	389 573
AIFB $F = Q$	SDType	0.803±0.004	0.690±0.007	0.013±0.000	**207 402**
	SLCN	0.976±0.004	0.068±0.008	0.002±0.000	399 530
	HMC	**0.987±0.001**	**0.036±0.003**	**0.001±0.000**	1 609 020
	HOMER	**0.987±0.001**	0.036±0.003	**0.001±0.000**	3 739 919
	MLC4.5	0.988±0.001	0.034±0.003	**0.001±0.000**	1 484 945
Mutagenesis $F = R$	SDType	0.726±0.006	1.277±0.032	0.016±0.001	**56 511**
	SLCN	0.762±0.008	0.865±0.006	0.012±0.000	1679
	HMC	**0.761±0.003**	**0.854±0.006**	**0.012±0.000**	3505
	HOMER	**0.778±0.004**	0.881±0.012	**0.012±0.000**	29 643
	MLC4.5	0.673±0.002	1.327±0.008	**0.018±0.000**	55 289
Mutagenesis $F = Q$	SDType	0.737±0.002	0.936±0.004	0.014±0.000	**71 213**
	SLCN	0.926±0.005	0.284±0.012	0.004±0.000	10 304
	HMC	**0.934±0.002**	**0.254±0.003**	**0.004±0.000**	29 171
	HOMER	**0.934±0.002**	0.256±0.003	**0.004±0.000**	109 943
	MLC4.5	0.899±0.004	0.373±0.014	**0.006±0.000**	67 920

Table 5: Evaluation of different classification methods on smaller SW datasets

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[3] 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 significantly 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_{out} and Q_{in}) substantially increases 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_{out} and Q_{in}. Since the filter feature selection method does not take dependencies between features into account, the selected feature set could contain several redundant features.

The results in Table 5 can illustrate the importance of the high scalability of SLCN. Training SLCN on AIFB with $F = Q$ is faster than training HOMER or HMC on $F = R$, and the prediction quality is significantly higher. On Mutagenesis, the training time is a bit slower, but again, the prediction quality is significantly higher. That shows that, when time and computing resources are limited, using SLCN allows you to work on a higher number of features in comparison to less scalable methods, and ultimately achieve better results.

6. Related Work

The problems of inference on noisy data in the Semantic Web have been identified, e.g., in[39] and[40]. 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[41]. 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, [42] predict possible predicates for resources based on co-occurrence of properties. The approach discussed in[43] 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[44] 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 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. Aprosio et al.[45] introduced an approach which first exploits cross-language links between DBpedia in different languages to increase coverage. 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[46] 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. Giovanni et al.[47] 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[3] 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 satisfies 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. [12], 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 multilabel classification, which has been also been exploited by LIFT[48]. 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[3], SDType is reportedly the best performing approach for the problem addressed in this paper[3,2], 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[29], 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[49] and AMIE[50], similarity based methods, such as Katz Index[51], Local Random Walks[52], and Path Ranking Algorithm[53]. 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. These models can be divided into translation models, which include TransE[30], TransR[31], tensor factorization models, such as RESCAL[32], and other models such as multiway neural networks (mwNN)[33] and

Holographic Embeddings (HolE)[34].

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. However, as embeddings become increasingly popular, initiatives such as Resource2Vec[h] or RDF2Vec[54], which publish embeddings for konwlede graphs such as DBpedia, YAGO, Wikidata, WordNet, and Freebase, enable the use of such embeddings on various applications. It is conceivable that in a near future, the main knowledge bases will have available for download the embeddings representations of its entities in different latent multirelational models.

Further efforts have been made in order to mitigate some of the latent multirelational models weaknesses. TRESCAL[55] proposes a tensor decomposition approach for knowledge base embedding based on RESCAL which is especially suitable for relation extraction. By leveraging relational domain knowledge about entity type information, their algorithm improves scalability and is better able to discover new relations missing from the database. Krompass et al. [56] applies type constraints of relations, which are often available in the knowledge base ontologies, in order to improve the performance of TransE, RESCAL and mwNN in the link prediction task. The authors report improvements of up to 77% in AUPRC and AUROC. [?] address the problem of translation-based models (such as TransR and TransE), which do not represent transitive and symmetric relations precisely. They introduce a role-specific projection which maps an entity to distinct vectors according to its role in a triple. That is, a head entity is projected onto an embedding space by a head projection operator, and a tail entity is projected by a tail projection operator

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. However, we believe that tackling the type prediction problem as link prediction is not the best approach. As out latent features experiment showed, the entity embeddings learned with HolE are significantly worse features than qualified relations, and simple ingoing and outgoing relations.

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 the quality of the prediction, and they

[h]`http://resource2vec.aksw.org/`

also show that SLCN can perform better than SDType, and scales better than the other multilabel classifiers evaluated in this paper. Given a smaller datasets and enough computing power available, state-of-the-art hierarchical multilabel classifiers, such as HOMER, HMC and MLC4.5, are the best choice. However, on larger datasets with high number of features, and where training time is an important factor, SLCN is the best option.

We also evaluated the use of entity embeddings as features for type prediction and the results indicate that they are not of great relevance to the problem. Simple graph-features such as ingoing and outgoing relations and qualified relations yield significantly better results, and combining entity embeddings with them does not provide any significant improvement.

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[57] 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).

References

1. 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) (2009) 154–165.
2. H. Paulheim and C. Bizer, Improving the quality of linked data using statistical distributions, *Int. J. Semant. Web Inf. Syst.* **10** (April 2014) 63–86.
3. H. Paulheim and C. Bizer, Type inference on noisy rdf data., in *International Semantic Web Conference (1)*, eds. H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty and K. Janowicz *Lecture Notes in Computer Science* **8218**, (Springer, 2013), pp. 510–525.
4. 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*, (Springer-Verlag, Berlin, Heidelberg, 2009), pp. 254–269.

5. 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.
6. H. Paulheim, Knowlegde Graph Refinement: A Survey of Approaches and Evaluation Methods, *Semantic Web Journal* (2016) to appear.
7. A. Melo, H. Paulheim and J. Völker, Type prediction in rdf knowledge bases using hierarchical multilabel classification, in *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics (WIMS)*2016, pp. 14:1–14:10.
8. R. E. Schapire and Y. Singer, Boostexter: A boosting-based system for text categorization, *Machine Learning* **39**(2/3) (2000) 135–168.
9. M.-L. Zhang and Z.-H. Zhou, ML-KNN: A lazy learning approach to multi-label learning, *Pattern Recogn.* **40**(7) (2007) 2038–2048.
10. 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** (2006) 1338–1351.
11. G. Tsoumakas, I. Katakis and I. Vlahavas, Random k-labelsets for multi-label classification, *IEEE Transactions on Knowledge and Data Engineering* **99**(1) (2010).
12. C. N. Silla, Jr. and A. A. Freitas, A survey of hierarchical classification across different application domains, *Data Min. Knowl. Discov.* **22** (January 2011) 31–72.
13. 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.
14. T. Fagni and F. Sebastiani, On the selection of negative examples for hierarchical text categorization, in *In Proceedings of The 3rd Language Technology Conference*2007, pp. 24–28.
15. A. Freitas and A. C. de Carvalho, *A Tutorial on Hierarchical Classification with Applications in Bioinformatics.* (Idea Group, January 2007), ch. VII, pp. 182–196.
16. A. Clare and R. D. King, Predicting gene function in saccharomyces cerevisiae, *Bioinformatics* **19** (2003) 42–49.
17. 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*, (Springer-Verlag, Berlin, Heidelberg, 2007), pp. 126–137.
18. 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*, (Springer-Verlag, London, UK, 2001), pp. 42–53.
19. 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.
20. N. Cesa-bianchi, L. Zaniboni and M. Collins, Incremental algorithms for hierarchical classification, in *Journal of Machine Learning Research* (MIT Press, 2004), pp. 31–54.
21. A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer and J. Lehmann, User-driven quality evaluation of dbpedia, in *Proceedings of the 9th International Conference on Semantic Systems* ACM2013, pp. 97–104.
22. H. Paulheim and A. Gangemi, Serving dbpedia with dolce–more than just adding a cherry on top, in *International Semantic Web Conference* Springer2015, pp. 180–196.
23. H. Paulheim, Identifying wrong links between datasets by multi-dimensional outlier detection., in *WoDOOM*2014, pp. 27–38.
24. D. Wienand and H. Paulheim, Detecting incorrect numerical data in dbpedia, in

European Semantic Web Conference Springer2014, pp. 504–518.
25. D. Fleischhacker, H. Paulheim, V. Bryl, J. Völker and C. Bizer, Detecting errors in numerical linked data using cross-checked outlier detection, in *International Semantic Web Conference* Springer2014, pp. 357–372.
26. F. Wu, J. Zhang and V. Honavar, *Learning Classifiers Using Hierarchically Structured Class Taxonomies*, in *Abstraction, Reformulation and Approximation: 6th International Symposium, SARA 2005, Airth Castle, Scotland, UK, July 26-29, 2005. Proceedings*, eds. J.-D. Zucker and L. Saitta. (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005), Berlin, Heidelberg, pp. 313–320.
27. 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) (2004) 1305–1308.
28. H. Paulheim, Exploiting linked open data as background knowledge in data mining., in *International Workshop on Data Mining on Linked Data (DMoLD*2013.
29. M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich, A review of relational machine learning for knowledge graphs, *Proceedings of the IEEE* **104**(1) (2016) 11–33.
30. A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, Translating embeddings for modeling multi-relational data, in *Advances in Neural Information Processing Systems 26*, eds. C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger (Curran Associates, Inc., 2013) pp. 2787–2795.
31. Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in *In Proceedings of AAAI15*2015.
32. M. Nickel, V. Tresp and H.-P. Kriegel, A three-way model for collective learning on multi-relational data, in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, eds. L. Getoor and T. Scheffer *ICML '11*, (ACM, New York, NY, USA, June 2011), pp. 809–816.
33. 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*, (ACM, New York, NY, USA, 2014), pp. 601–610.
34. M. Nickel, L. Rosasco and T. A. Poggio, Holographic embeddings of knowledge graphs, *CoRR* **abs/1510.04935** (2015).
35. 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* ACM2012, p. 31.
36. P. Ristoski and H. Paulheim, A comparison of propositionalization strategies for creating features from linked open data, in *Linked Data for Knowledge Discovery*2014.
37. G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek and I. Vlahavas, Mulan: A java library for multi-label learning, *Journal of Machine Learning Research* **12** (2011) 2411–2414.
38. F. Erxleben, M. Günther, M. Krötzsch, J. Mendez and D. Vrandečić, Introducing Wikidata to the linked data web, in *Proceedings of the 13th International Semantic Web Conference (ISWC'14)*, eds. 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 *LNCS* **8796**, (Springer, 2014), pp. 50–65.
39. Q. Ji, Z. Gao and Z. Huang, Reasoning with noisy semantic data, in *The Semanic Web: Research and Applications (ESWC 2011), Part II*2011, pp. 497–502.
40. A. Polleres, A. Hogan, A. Harth and S. Decker, Can we ever catch up with the web?, *Semantic Web Journal* **1**(1,2) (2010) 45–52.

41. J. Neville and D. Jensen, Iterative classification in relational data, in *Proc. AAAI Workshop on Learning Statistical Models from Relational Data*2000, pp. 13–20.
42. E. Oren, S. Gerke and S. Decker, Simple algorithms for predicate suggestions using similarity and co-occurrence, in *European Semantic Web Conference (ESWC 2007)* (Springer, 2007) pp. 160–174.
43. A. Pohl, Classifying the wikipedia articles in the opencyc taxonomy, in *Web of Linked Entities Workshop (WoLE 2012)*2012.
44. 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*2012, pp. 1361–1370.
45. 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.
46. 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.
47. 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.
48. M. Zhang and L. Wu, Lift: Multi-label learning with label-specific features, *IEEE Trans. Pattern Anal. Mach. Intell.* **37**(1) (2015) 107–120.
49. S. Muggleton, Inverse Entailment and Progol, *New Generation Computing, Special issue on Inductive Logic Programming* **13**(3-4) (1995) 245–286.
50. 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*, (ACM, New York, NY, USA, 2013), pp. 413–422.
51. L. Katz, A new status index derived from sociometric analysis, *Psychometrika* **18** (March 1953) 39–43.
52. W. Liu and L. Lü, Link prediction based on local random walk, *EPL (Europhysics Letters)* **89**(5) (2010) p. 58007.
53. N. Lao and W. W. Cohen, Relational retrieval using a combination of path-constrained random walks, *Mach. Learn.* **81** (October 2010) 53–67.
54. P. Ristoski and H. Paulheim, Rdf2vec: Rdf graph embeddings for data mining, in *International Semantic Web Conference*2016. To appear.
55. B. Y. C. M. Kai-Wei Chang, Scott Wen-tau Yih, Typed tensor decomposition of knowledge bases for relation extraction, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (ACL Association for Computational Linguistics, October 2014).
56. D. Krompaß, S. Baier and V. Tresp, Type-constrained representation learning in knowledge graphs, *CoRR* **abs/1508.02593** (2015).
57. P. Ristoski and H. Paulheim, Feature selection in hierarchical feature spaces, in *Discovery Science*2014.