Factorizing YAGO

Scalable Machine Learning for Linked Data

Maximilian Nickel
Ludwig-Maximilians University
Munich
Oettingenstr. 67
Munich, Germany
nickel@dbs.ifi.lmu.de

Volker Tresp
Siemens AG
Corporate Technology
Otto-Hahn Ring 6
Munich, Germany
volker.tresp@siemens.com

Hans-Peter Kriegel
Ludwig-Maximilians University
Munich
Oettingenstr. 67
Munich, Germany
kriegel@dbs.ifi.lmu.de

ABSTRACT

Vast amounts of structured information have been published in the Semantic Web's Linked Open Data (LOD) cloud and their size is still growing rapidly. Yet, access to this information via reasoning and querying is sometimes difficult, due to LOD's size, partial data inconsistencies and inherent noisiness. Machine Learning offers an alternative approach to exploiting LOD's data with the advantages that Machine Learning algorithms are typically robust to both noise and data inconsistencies and are able to efficiently utilize non-deterministic dependencies in the data. From a Machine Learning point of view, LOD is challenging due to its relational nature and its scale. Here, we present an efficient approach to relational learning on LOD data, based on the factorization of a sparse tensor that scales to data consisting of millions of entities, hundreds of relations and billions of known facts. Furthermore, we show how ontological knowledge can be incorporated in the factorization to improve learning results and how computation can be distributed across multiple nodes. We demonstrate that our approach is able to factorize the YAGO 2 core ontology and globally predict statements for this large knowledge base using a single dual-core desktop computer. Furthermore, we show experimentally that our approach achieves good results in several relational learning tasks that are relevant to Linked Data. Once a factorization has been computed, our model is able to predict efficiently, and without any additional training, the likelihood of any of the $4.3 \cdot 10^{14}$ possible triples in the YAGO 2 core ontology.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Parameter Learning*; E.1 [**Data Structures**]: Graphs and Networks

General Terms

Algorithms, Performance, Experimentation

Keywords

Large-Scale Machine Learning, Semantic Web, Linked Open Data, Tensor Factorization, Relational Learning

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

1. INTRODUCTION

The Semantic Web's Linked Open Data (LOD) [6] cloud is growing rapidly. At the time of this writing, it consists of around 300 interlinked databases, where some of these databases store billions of facts in form of RDF triples.[1] Thus, for the first time, relational data from heterogeneous, interlinked domains is publicly available in large amounts, which provides exciting opportunities for Machine Learning. In particular, much progress has been made in recent years in the subfield of *Relational* Machine Learning to learn efficiently from attribute information and information about the entities' relationships in interlinked domains. Some Relational Machine Learning approaches can exploit contextual information that might be more distant in the relational graph, a capability often referred to as collective learning. State-of-the-art collective learning algorithms can therefore be expected to utilize much of the information and patterns that are present in LOD data. Moreover, the Semantic Web itself can benefit from Machine Learning. Traditional Semantic Web approaches such as formal semantics, reasoning or ontology engineering face serious challenges in processing data in the LOD cloud, due to its size, inherent noisiness and inconsistencies. Consider, for example, that `owl:sameAs` is often misused in the LOD cloud, leading to inconsistencies between different data sources [13]. Further examples include malformed datatype literals, undefined classes and properties, misuses of ontological terms [16] or the modeling of a simple fact such as *Nancy Pelosi voted in favor of the Health Care Bill* using eight RDF triples [15]. Partial inconsistencies in the data or noise such as duplicate entities or predicates are direct consequences of the open nature of Linked Open Data. For this reason, it has been recently proposed to look at alternative approaches for new Semantic Web reasoning paradigms [15]. The underlying idea is that reasoning can often be reduced to the task of classifying the truth value of potential statements. By abandoning requirements such as logical soundness and completeness, this classification could be carried out by approximate methods, such as Machine Learning. Moreover, it is reasonable to assume that there exist many dependencies in the LOD cloud which are rather statistical in nature than deterministic, such that statistical methods have the potential to add significant surplus value in combination with what reasoning already provides. Reliable predictions of unknown triples in the scale of entire knowledge bases could mark a first step towards

[1] Information taken from http://lod-cloud.net

this new reasoning paradigm for the Semantic Web. Here, in our approach to this challenge, we focus on the YAGO 2 ontology [27], a large knowledge base that lies, along with other databases such as DBpedia [2], at the core of the LOD cloud.

Applying Machine Learning to Linked Data at this scale however, is not trivial. For instance, due to the linked nature of the data, using a relational learning approach is mandatory. But relational learning algorithms often require a considerable amount of prior knowledge about the domain of discourse, e.g. a knowledge base for Markov Logic Networks (MLN) [24] or the structure of a Bayesian Network. This can become a obstacle when applying Machine Learning to Linked Open Data, since it is difficult and expensive to gather this kind of knowledge manually or automatically. Also, many relational learning algorithms have problems to process data of the size that is required to approach serious, real-life Semantic Web problems as these algorithms usually do not scale well with the number of known facts or entities in the data. Another challenge for Machine Learning is that knowledge bases in the Linked Open Data cloud often contain only positive examples of instantiated relations and it is not valid to infer the non-existence of a relation from its absence in the data, due to an underlying open-world assumption.

Here, we address these challenges in the following way. First, we opted for tensor factorization as the learning approach. Tensor factorizations, just as matrix factorizations, have shown excellent performance in high dimensional and sparse domains. We will demonstrate, that tensors are suitable for Semantic Web data and fit nicely to the triple-structure of RDF(S) data due to their multi-way nature. In particular, we employ RESCAL, a tensor factorization for relational learning, which has been shown to produce very good results on canonical relational learning tasks. We present a novel implementation to compute this factorization that honors the sparsity of LOD data and can scale to large knowledge bases, even on commodity hardware. Furthermore, we present an extension to the RESCAL algorithm to handle attributes of entities efficiently. By using this extension in combination with RESCAL, handling of Semantic Web data becomes straightforward. RDF(S) data can be transformed automatically into a tensor representation, while training the model only requires minimal prior information about the domain of discourse such as the number of latent variables in the data and, for complex models, optional regularization parameters. To improve scalability beyond the capabilities of a single computer, we also show how the factorization can be computed across multiple nodes, using distributed computing paradigms such as map-reduce.

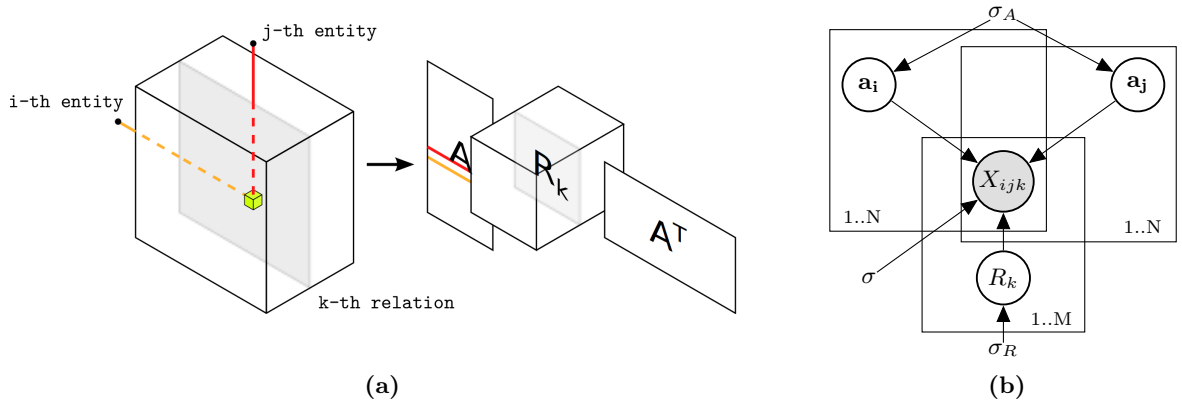
2. RELATED WORK

Despite their long tradition in fields like psycho- and chemometrics, tensors and tensor factorizations have only recently been applied to Machine Learning, e.g. to incorporate dynamic aspects in network models. For instance, [28] presents methods for dynamic and streaming tensor analysis and applies them to network traffic and bibliographic data. In [23], a specialized tensor factorization for personalized item recommendations is used to include information of the preceding transaction. For relational learning, [29] introduced the Bayesian Clustered Tensor Factorization (BCTF) and applied it to various data sets of smaller and medium size. An extensive review of tensor decompositions and their applications can be found in [20]. In the context of the Semantic Web, Inductive Logic Programming (ILP) and kernel learning have been the dominant Machine Learning approaches so far [7, 9, 11]. Furthermore, [17] uses regularized matrix factorization to predict unknown triples in Semantic Web data. Recently, [21] proposed to learn Relational Bayesian Classifiers for RDF data via queries to a SPARQL endpoint. Also, the work on SPARQL-ML [18] extends SPARQL queries to support data mining constructs. [5] employs a coevolution-based genetic algorithm to learn kernels for RDF data. Probably most similar to our approach is TripleRank [12], which applies the CP [8] tensor decomposition to RDF graphs for faceted browsing. However, in contrast to the tensor factorization employed in this paper, CP isn't capable of collective learning, which is an important feature for learning on the Semantic Web. Recently, methods such as association rule mining and knowledge base fragment extraction have been applied to large Semantic Web databases for tasks like schema induction and learning complex class descriptions [31, 14]. To the best of our knowledge, there have yet not been any attempts to apply a general relational learning approach to knowledge bases of the size considered in this paper.

3. THE MODEL

Our approach to large-scale learning on the Semantic Web is based on RESCAL, a tensor factorization that has shown very good results in various canonical relational learning tasks such as link prediction, entity resolution or collective classification [22]. The main advantage of RESCAL, if compared to other tensor factorizations, is that it can exploit a collective learning effect when applied to relational data. Collective learning refers to the automatic exploitation of attribute and relationship correlations across multiple interconnections of entities and relations. It is known that applying a collective learning method to relational data can improve learning results significantly [26]. For instance, consider the task of predicting the party membership of a president of the United States of America. Naturally, the party membership of the president and his vice president are highly correlated, since both persons have mostly been members of the same party. These correlations along the `vicePresidentOf` and `partyOf` relations can be exploited by a collective learning method to infer the correct party membership of a person in this domain. It has been shown in [22] that RESCAL is able to detect such correlations, because it has been designed to account for the inherent structure of dyadic relational data. Since attributes and complex relations are often connected by intermediary nodes such as blank nodes or abstract entities when modeled according to the RDF formalism, this collective learning ability of RESCAL is a very important feature for learning on the Semantic Web.[2] The following sections will present in more detail the RESCAL algorithm, will discuss how RDF(S) data is modeled as a tensor in RESCAL and will introduce some novel extensions to the algorithm.

[2]For instance, in the current version of the DBpedia ontology (3.7), geographical locations such as river mouth locations are modeled by the following pattern `(Rhone, mouthPosition, Rhone-mouthPosition)`, `(Rhone-mouthPosition, longitude, 4.845555782318115)`.



(a) (b)

Figure 1: Graphical representations of RESCAL. (a) Illustration of data representation and factorization in RESCAL. (b) Graphical model of the RESCAL factorization in plate notation. Observed variables are shaded gray.

3.1 Modeling Semantic Web Data

Let a relational domain consist of n entities and m dyadic relation types. Using RESCAL, such data is modeled as a three-way tensor $\mathcal{X}$ of size $n \times n \times m$, where the entries on two modes of the tensor correspond to the combined entities of the domain of discourse and the third mode holds the m different types of relations. A tensor entry $\mathcal{X}_{ijk} = 1$ denotes the fact that the relation *k-th Relation(i-th entity, j-th entity)* exists. Otherwise, for non-existing or unknown relations, $\mathcal{X}_{ijk}$ is set to zero. This way, RESCAL approaches the problem of learning from positive examples only, by assuming that missing triples are very likely not true, an approach that makes sense in a high-dimensional but sparse domain. Figure 1a shows an illustration of this modeling method. Each frontal slice $X_k = \mathcal{X}_{:,:,k}$ of $\mathcal{X}$ can be interpreted as the adjacency matrix of the relational graph for the respective relation k.

Creating such a tensor representation for RDF(S) data is straightforward. The entities are given by the set of all resources, classes and blank nodes in the data, while the set of relations consists of all predicates that include entity-entity relationships. For each existing triple `(i-th entity, k-th predicate, j-th entity)`, the corresponding entry $\mathcal{X}_{ijk}$ is set to one, otherwise it is set to zero. Since the original RESCAL model assumes that two of the three modes are defined by entities, this procedure is constrained to resources. However, much of the information in the LOD cloud is given as literal values. For this reason, we present an efficient extension to RESCAL in Section 3.5, such that attributes of entities, i.e. literal values, can be included in the factorization.

It is also important to note that in this modeling of RDF(S) data, we do not draw a distinction between ontological knowledge (i.e. RDFS in the $\mathcal{T}$-Box) and instance data (the $\mathcal{A}$-Box). Instead, for a given domain, classes *and* instances of these classes are modeled equally as entities in the tensor $\mathcal{X}$. Furthermore, all predicates from the $\mathcal{T}$-Box *and* the $\mathcal{A}$-Box form the slices X_k of $\mathcal{X}$. This way, ontological knowledge is represented similarly to instance data by an appropriate entry $\mathcal{X}_{ijk} = 1$, such that facts about instances as well as data from ontologies are integrated simultaneously in one tensor representation. In doing so, ontologies are handled like soft constraints, meaning that the additional information present in an ontology guides the factorization to semantically more reasonable results, but doesn't impose hard constraints on the model. Consequently, our modeling has aspects of both a pure data-centric and an ontology-driven Semantic Web approach.

3.2 Factorizing Semantic Web Data

Given a tensor $\mathcal{X}$ of size $n \times n \times m$ that has been constructed as described in Section 3.1, RESCAL computes a factorization of $\mathcal{X}$, such that each frontal slice X_k of $\mathcal{X}$ is factorized into the matrix product

$$X_k \approx A R_k A^T, \text{for } k = 1, \ldots, m$$

where A is a $n \times r$ matrix, R_k is a full, asymmetric $r \times r$ matrix and r is a user-given parameter that specifies the number of latent components or factors. The factor-matrices A and R_k are computed by solving the optimization problem

$$\min_{A,R} f_{loss}(A, R) + f_{reg}(A, R) \tag{1}$$

where

$$f_{loss}(A, R) = \frac{1}{2}\left(\sum_k \|\mathcal{X}_k - A R_k A^T\|_F^2\right) \tag{2}$$

and f_{reg} is the regularization term

$$f_{reg}(A, R) = \lambda_A \|A\|_F^2 + \lambda_R \sum_k \|R_k\|_F^2 \tag{3}$$

which is included to prevent overfitting of the model.

RESCAL can be regarded as a latent-variable model for multi-relational data. Let $\mathbf{a}_i$ denote the i-th row of A. Then, (1) explains observed variables, i.e. $\mathcal{X}_{ijk}$, through latent feature vectors $\mathbf{a}_i$, $\mathbf{a}_j$ and R_k. Figure 1b illustrates this interpretation as a graphical model in plate notation. In this model, $\mathbf{a}_i$ and $\mathbf{a}_j$ are representations of the i-th and j-th entity by latent components, i.e. the columns of A, which have been derived by the factorization to explain the observed variables.[3] Furthermore, an additional interpretation of A is as an embedding of the entities into a latent-component

[3]For instance, in the US presidents example, a latent-variable model could try to explain observed data such as party membership via the latent components *conservative politician*, *liberal politician*, *conservative party*, *liberal party* etc. Unfortunately, in many cases, including RESCAL, the invented latent components are not easily interpretable.

space, where the entities' similarity in this space reflects their similarity in the relational domain. R_k, on the other hand, models the interactions of the latent components in the k-th predicate. Expressing data in terms of newly invented latent components is often referred to as predicate invention in statistical relational learning and considered a powerful asset [19].

To solve (1), [22] presents an efficient alternating least squares algorithm, which updates A and R_k iteratively until a convergence criterion is met. In the following, we will refer to this algorithm as RESCAL-ALS. In detail, updates for A and R are computed by

Update A:

$$A \leftarrow \left[\sum_{k=1}^{m} X_k A R_k^T + X_k^T A R_k\right] \left[\sum_{k=1}^{m} B_k + C_k + \lambda_A \mathbf{I}\right]^{-1}$$

where

$$B_k = R_k A^T A R_k^T, \quad C_k = R_k^T A^T A R_k$$

Update R_k:

$$R_k \leftarrow \left(Z^T Z + \lambda_R \mathbf{I}\right)^{-1} Z^T \mathbf{vec}(X_k)$$

where $Z = A^T \otimes A^T$ and $\otimes$ denotes the Kronecker product.

However, computing the update steps of R_k in this form would be intractable for large-scale data, since it involves the $r^2 \times n^2$ matrix Z. Fortunately, similar to the ASALSAN algorithm [4], it is possible to use the QR decomposition of A to simplify the update steps for R_k significantly. The basic idea is to minimize for each R_k a function that is equivalent to (2), namely

$$\min_{R_k} \|\hat{X}_k - \hat{A} R_k \hat{A}^T\|_F^2$$

where $A = Q^T \hat{A}$ is the result of the QR decomposition of A and $\hat{X}_k = Q^T X_k Q$. By using $\hat{X}_k$ and $\hat{A}$ as replacements for A and X_k in the update of R_k, this step is now only dependent on the number of latent components, since $\hat{A}$ and $\hat{X}_k$ are only $r \times r$ matrices.

The starting points for A and R_k can be random matrices or, as an alternative, A is initialized from the eigendecomposition of $\sum_k (X_k + X_k^T)$. To compute the factor matrices, the algorithm performs alternating updates of A and all R_k until $\frac{f(A,R)}{\|\mathcal{X}\|_F^2}$ converges to some small threshold ϵ or a maximum number of iterations is exceeded. However, for large-scale learning, these convergence criteria are not applicable, since computing a dense, $n \times n$ matrix $A R_k A^T$ would often exceed available memory. To overcome this problem, we employ two alternative methods to measure the progress of the algorithm in order to determine a reasonable number of iterations. One method is to compute the fit on a smaller sample of $\mathcal{X}$ and the corresponding rows of A. The drawback of this approach is that the fit value measured on a sample of the original data will not decrease monotonically, since (1) optimizes a slightly different function. However, when the true fit value approaches convergence, the surrogate function will also converge. Another option is to analyze the change of the model in each iteration using some measure, e.g. $\|A - A^{old}\|_F^2 + \sum_k \|R_k - R_k^{old}\|_F^2$. The minor drawback of this approach is that the factor matrices of the previous iteration are required.

3.3 Usage Scenarios on the Semantic Web

Once the factorization has been computed, RESCAL can be applied to various relational learning problems relevant to the Semantic Web. In the following we briefly describe some of these scenarios from a Semantic Web perspective.

3.3.1 Prediction of Unknown Triples

RESCAL can be used to predict the existence of unknown triples. In particular, the matrix-vector product $\hat{X}_{ijk} = \mathbf{a}_i^T R_k \mathbf{a}_j$ can be interpreted as the score that the model assigns to the existence of the triple `(i-th entity, k-th predicate, j-th entity)`. This value $\hat{X}_{ijk}$ can then be compared to a given threshold θ in order to determine whether the triple in question is expected to exist. However, due to a general sparseness of relationships there is a strong bias towards zero, which makes it difficult to select a reasonable threshold θ. When it is not necessary to determine whether a particular triple exists, but the objective is to retrieve triples by their likelihood, for instance the most likely persons in the data to be born in Lyon, a better alternative is to create a ranking of the entries in question. Note that inference is very fast. To determine which entities are most likely to have a specific link to entity a_j, it is sufficient to compute the matrix product $A R_k \mathbf{a}_j$. This is a nice property of RESCAL compared to other relational learning approaches where exact inference is often intractable.

3.3.2 Retrieval of Similar Entities

A particular strength of the RESCAL factorization is that it computes a global latent-component representation of the entities, i.e. the matrix A, and local interaction-models of the latent variables for each predicate, i.e. the matrices R_k. Analogous to the retrieval of documents via latent-variable models, the entities' latent-component representations that have been computed with RESCAL can be used to retrieve similar entities. As mentioned in Section 3.2, the matrix A can be interpreted as an embedding of the entities into a latent-component space that reflects their similarity over all relations in the domain of discourse. Therefore, in order to retrieve entities that are similar to a particular entity e with respect to all relations in the data, it is sufficient to compute a ranking of entities by their similarity to e in A. This can be done efficiently, since A is an $n \times r$ matrix.

3.3.3 Decision Support for Knowledge Engineers

Another very interesting application of RESCAL is the automatic creation of taxonomies from instance data. Recently, it has been proposed that Machine Learning methods should assist knowledge engineers in the creation of ontologies, such that an automated system suggests new axioms for an ontology, which are added under the supervision of an engineer [3]. Here, we focus on the simpler task of learning a taxonomy from instance data. A taxonomy can be interpreted as a hierarchical grouping of instances. Consequently, a natural approach to learning a taxonomy for a particular domain is to compute a hierarchical clustering of the entities in this domain and to interpret the resulting clusters according to their members. However, there are only very few approaches that are able to compute a *hierarchical* clustering for *multi-relational* data [25]. To compute such a clustering with RESCAL, we use again the property of A to reflect the similarity of entities in the relational domain, and simply compute a clustering in the latent-component space.

Table 1: Computational complexity for operations in the update steps of A and R

Update A		
Computation	Complexity	
$X_k A R_k^T \wedge X_k^T A R_k$	$O(pnr) + O(nr^2)$	each
$B_k \wedge C_k$	$O(nr^2)$	each
Matrix inversion	$O(r^3)$	
Update R_k		
Computation	Complexity	
QR decomp. of A	$O(nr^2)$	
Projection $Q^T X_k Q$	$O(pnr^2)$	
Matrix inversion	$O(r^3)/O(r^5)$	$\lambda = 0/\lambda \neq 0$
$(Z^T Z)^{-1} Z^T \mathbf{vec}(X_k)$	$O(pr^3)$	

This has the advantage that any feature-based hierarchical clustering algorithm can readily be applied to this matrix, since A represents entities only by their participation in the latent components. The clustering, however, will still be determined by the entities' similarities in the relational domain. While our approach differs in some important aspects from the system envisioned in [3], it can be used to address some of the discussed challenges, in particular scalability.

3.4 Complexity of a Sparse Implementation

To be able to scale to large knowledge bases, it is necessary for a learning algorithm to have low computational complexity and low memory usage. Relational data is usually very sparse, what causes each predicate slice X_k in the tensor representation of RESCAL to be also very sparse. By using sparse linear algebra, we can exploit this property of X_k and provide a very scalable implementation of RESCAL-ALS. In the following we present an analysis of a sparse implementation under the assumption that X_k is a sparse matrix, while A and R_k are dense matrices. We will show that such an implementation features the very desirable property to have only linear computational complexity with regard to the number of entities or predicates in the dataset as well as with regard to the number of known facts.

The update steps for A and R are obviously linear in the number of predicates m, regardless of the sparsity of $\mathcal{X}$, since this parameter occurs only in the summation indices of (1). However, the computational complexity with regard to the number of entities n, the number of nonzero entries p in $\mathcal{X}$ and the model complexity, i.e. the number of latent components r, is more elaborate. Table 1 shows the complexity of each operation in the update steps of A and R in terms of these parameters. In this analysis, we used $O(pnl)$ as the runtime complexity for the matrix product of a sparse matrix U with a dense $n \times l$ matrix V, where p is the number of non-zeros in U. The operations listed in Table 1 are iterated only a small number of times until the algorithm converges or a maximum number of iterations is reached. Consequently, the computational complexity of a sparse implementation of the RESCAL-ALS algorithm is *linear in* n *or* m and superlinear only in the model complexity r. As it holds that

$$(Z^T Z)^{-1} Z^T = (A^T A)^{-1} A^T \otimes (A^T A)^{-1} A^T$$

the $O(n^5)$-operations in the update step for R_k can be reduced to $O(n^3)$ complexity in the non-regularized case. Also, since the number of non-zeros in a particular slice p occurs only as a linear factor, the algorithm has also linear computational complexity with regard to the number of known facts.

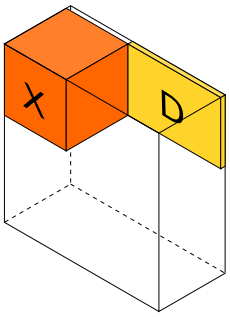
Concerning the scalability of RESCAL-ALS, it is also interesting to note that the algorithm can be computed easily in a distributed way. For models with complexity r the dominant costs in each update step are the matrix multiplications $X_k A R_k^T + X_k^T A R k$, and $(Z^T Z)^{-1} Z^T \mathbf{vec}(X_k)$, since $O(r^3) < O(pnr^2)$ for $p > 1$. Due to the sums in the update of A, this step can be computed distributedly by using a map-reduce approach. First, the current state of A and R_k is distributed to a number of available computing nodes. Then, these nodes compute $X_k A R_k^T + X_k^T A R_k$ and $B_k + C_k$ locally, for those k that have been assigned to them. Given the results of these computations, the master node can reduce the results and compute the matrix inversion, which only involves $r \times r$ matrices and the final matrix product. Since the updates of R_k are independent of each other, these steps can be computed in a similar way.

Let's now consider the memory complexity of RESCAL: In each iteration, only one frontal slice X_k has to be kept in memory. Since sparse matrices usually have a linear memory complexity $O(p)$ with regard to the number of nonzero elements in a matrix, our approach scales up to billions of known facts. However, the factor matrices are more demanding in terms of memory, especially A, a dense $n \times r$ matrix, such that if the domain contains a very large number of entities, additional dimensionality reduction techniques such as the "hashing trick" [32] might be required.

3.5 Adding Attributes to the Factorization

In its original form, RESCAL doesn't recognize attributes of entities explicitly. However, much information in the LOD cloud is in the form of attributes. Also, attributes would allow us to include simple, automatically aggregated features, such as `(entity, hasRelation, relation name)` that can be useful in some prediction tasks. A naïve approach to include attributes in the factorization would be to add an additional slice $m + 1$ for attributes and to apply some form of preprocessing to the attribute values in the data[4] The results of this preprocessing step would then be added together with their respective predicate as new indices $n + 1, \ldots, n + l$ to the entity modes of $\mathcal{X}$. Furthermore, for each attribute value o of an entity i, those entries $X_{i,m+1,n+j}$ would be set to one, where $n + j$ corresponds to o. For instance, assuming that textual data is preprocessed by tokenizing, an attribute `(Albert_Einstein, foaf:name, 'Albert Einstein')` would set those entries $\mathcal{X}_{ijk} = 1$ where $i \equiv$ `Albert_Einstein`, $k \equiv m + 1$, and $j \equiv$ `<foaf:name, 'Albert'>` or $j \equiv$ `<foaf:name, 'Einstein'>`. While this handling of attributes can work for small datasets, it is not applicable for large-scale learning on the Semantic Web, as it would increase the dimensionality of the entity modes dramatically. The main problem associated with this procedure is that attributes are included as true entities, although they never occur as subjects in a relation. Since RESCAL assumes a tensor with frontal slices of size $n \times n$, a huge amount of entries would be wasted in the tensor, what in

[4]For instance discretizing continuous variables or tokenizing and stemming textual data



(a)

Update A	
Additional Computations	Complexity
DV^T	$O(pnr)$
VV^T	$O(nr^2)$
Update V	
Computation	Complexity
$(A^TA + \gamma\mathbf{I})^{-1}A^TD$	$O(pnr) + O(r^3) + O(nr^2)$

(b)

Figure 2: (a) RESCAL model with attributes. (b) Complexity added through attributes.

turn would lead to an increased runtime since a significantly larger tensor would have to be factorized. Figure 2a shows an illustration of this effect. To overcome this problem, we propose to handle attributes by a separate matrix factorization which we perform jointly with the tensor factorization. The basic idea is to process attribute values just as described above, but to add the `<predicate, value>` pairs to a separate entity-attributes matrix D and not to the tensor $\mathcal{X}$. Therefore, D is constructed similar as in the relational learning algorithm SUNS [17]. The entity-attributes matrix D is then factorized into

$$D \approx AV$$

where A is the entities' latent-component representation of the RESCAL model and V is an $r \times l$ matrix, which provides a latent-component representation of the attributes. To include this matrix factorization as an additional constraint on A in the tensor factorization of $\mathcal{X}$, we add the term $f_{attr}(A, V)$ to the minimization problem (1), such that

$$\min_{A,R,V} f_{loss}(A, R) + f_{reg}(A, R) + f_{attr}(A, V)$$

and

$$f_{attr}(A, V) = \|D - AV\|_F^2 + \lambda_V \|V\|_F^2$$

To adapt the RESCAL-ALS algorithm to this new objective, the update step of A has to be changed and a new update step for V has to be included. In particular, the update steps for A and V become

Update A:

$$A \leftarrow \left[DV^T + \sum_{k=1}^{m} X_k A R_k^T + X_k^T A R_k\right] \times \left[VV^T + \sum_{k=1}^{m} B_k + C_k + \lambda_A \mathbf{I}\right]^{-1}$$

where B_k, C_k are identical as in updates without attributes.

Update V:

$$V \leftarrow \left(A^TA + \lambda_V \mathbf{I}\right)^{-1} A^TD$$

Handling attributes in this way is essentially equivalent to the naïve approach, but significantly more efficient to compute. Figure 2b lists the computational complexity of the additional operations that are necessary to include attributes in the factorization.[5] It can be seen, that while the additional operations will increase the runtime of the algorithm, they do not alter the linear scalability.

4. EVALUATION

In order to evaluate the ability of our approach to factorize large knowledge bases, we conducted various experiments on both the YAGO 2 core ontology and on synthetic data. Some statistics of the current version[6] of the YAGO ontology are listed in Table 2. Out of the 87 predicates that are included

YAGO 2 core ontology	
Number of Entities	2.6 million
Number of Classes	340,000
Number of Predicates	87
Number of Known Facts	33 million

Table 2: YAGO 2 core statistics

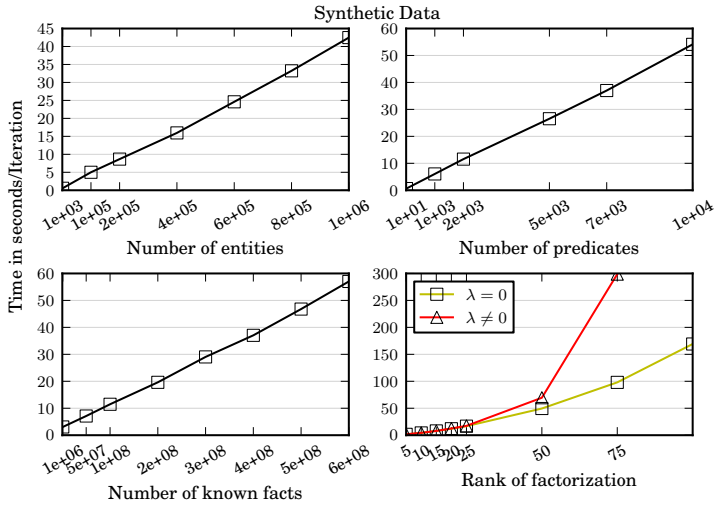
in this knowledge base, we treat 38 predicates as entity-to-entity relations, while the rest is handled as attributes. Furthermore, we included the materialization of all `rdf:type` triples and transitive rules, which can be done conveniently via the YAGO conversion tools.[7] This resulted in a total of 64 million triples. From this raw data we constructed a tensor $\mathcal{X}$ of size $3000417 \times 3000417 \times 38$ and an attribute matrix D of size 3000417×1138407. D has been created by tokenizing and stemming attribute values of textual attributes, such as `rdfs:label`, `yago:hasPreferredMeaning` etc. The tensor $\mathcal{X}$ has approximately 41 million entries, while D has around 35.4 million entries. It can be seen that both $\mathcal{X}$ and D are very sparse. A tensor for the YAGO 2 core ontology has $4.3 \cdot 10^{14}$ possible entries (of which $2.4 \cdot 10^{13}$ are valid according to `rdfs:range` and `rdfs:domain` constraints), but only $4 \cdot 10^7$ non-zero entries. Once a factorization has been computed, our model is able to predict, without any additional training, the likelihood of any of the $4.3 \cdot 10^{14}$ possible triples in the YAGO 2 ontology.

In the following experiments, all algorithms have been implemented in Python and NumPy, respectively, and were

[5]The operation A^TA is not included, since it can be reused from the update step of R_k

[6]Version 20110315 from `http://yago-knowledge.org`

[7]Version 20111027 from `http://yago-knowledge.org`



(a)

Parameter	Value Range	
	Min	Max
Entities	$1 \cdot 10^4$	$1 \cdot 10^6$
Predicates	$1 \cdot 10^1$	$1 \cdot 10^4$
Facts	$1 \cdot 10^6$	$6 \cdot 10^8$
Rank	5	100

Type	Number of entities
wordnet:person	884,261
wordnet:location	429,828
wordnet:movie	62,296

(b)

Figure 3: (a) Runtime experiments on synthetic data (b) Statistics for selected classes in YAGO 2 core

evaluated on a single Intel Core 2 Duo machine with two 2.5GHz cores and 4GB RAM, except where otherwise noted.

4.1 Large-Scale Prediction of Unknown Triples

A central requirement of an algorithm for large-scale learning on Semantic Web data is to scale well with the number of entities, predicates and known facts in the data. Therefore, to evaluate the scalability of our approach, we first conducted experiments on synthetic data, where various sparse datasets were created with different numbers of entities n, numbers of predicates m, and nonzero values p. In each experiment we varied exactly one of these parameters, while keeping all other parameters fixed. To evaluate how well our approach scales with regard to a particular parameter, we computed a factorization of the synthetic data with r latent components and recorded the average runtime per iteration. Additionally we also evaluated the runtime with respect to different values for r, while keeping n, m and p fixed. Figure 3 lists the value ranges that were used in the experiments for each parameter as well as the runtime results. It can be seen that RESCAL scales indeed linearly with respect to the number of entities, the number of predicates and the number of known facts, while it scales superlinear with regard to the latent components of the factorization. Even for large parameter values such as $6 \cdot 10^8$ nonzero entries, the runtime for a single iteration is still measured in seconds.

Given these very promising results in terms of runtime scalability, the next objective in our experiments was to evaluate the capabilities of RESCAL to predict unknown triples in a large-scale setting. For this reason, we conducted several link-prediction experiments on the *entire* YAGO 2 core ontology. The objective of these experiments was to correctly predict links for the `rdf:type` predicate for various higher level classes, namely `wordnet:person`, `wordnet:location` and `wordnet:movie`. The choice of these classes is motivated by the fact that they occur on different levels in the subclass hierarchy and are of different size. Some statistics for these classes are available in Figure 3b. For each of these classes we performed 5-fold stratified cross-validation over all entities in the data in two different settings.

a) Only the `rdf:type` triples that include the class C that should be predicted were removed from the test fold. All other type triples, including subclasses of C, are still present.

b) *All* `rdf:type` triples have been deleted in the test fold.

The objective in setting a) is unusual for Machine Learning, since obviously a high correlation exists between classes and their subclasses. However, it is a very common Semantic Web problem, as it corresponds to the materialization of `rdf:type` triples, given an ontology. Setting b), on the other hand is a very common Machine Learning problem, as it corresponds to the classification of instances, given their relations and attributes. Since there is a large skew in the distribution of existing and non-existing triples, we report the area under the precision-recall curve (AUC-PR) to evaluate the results, what is suitable for this setting [10].

Figure 4 shows the results of these experiments. It can be seen that RESCAL learns a reasonable model in both settings. The results for setting a) indicate that given enough support in the data, RESCAL can predict triples that originate from transitive rules such as

$$\forall xyz : \text{classOf}(x,y) \wedge \text{subClassOf}(y,z) \Rightarrow \text{classOf}(x,z)$$

with high precision and recall. But the results for setting b) are also very encouraging, since they are close to the results of setting a). Not surprisingly, to achieve good results in setting b) it is necessary to train more complex models. For instance, the results to predict `wordnet:person` in setting a) were computed with $r = 7$, while setting b) required $r = 15$. This is consistent with the higher degree of difficulty for setting b). For persons and locations the factorization was computed without attribute information. However, for movies we performed an additional experiment, where we added values of the predicate `yago:hasWikipediaCategory` from the full YAGO ontology as attributes to the factorization. The rationale behind this procedure was that this

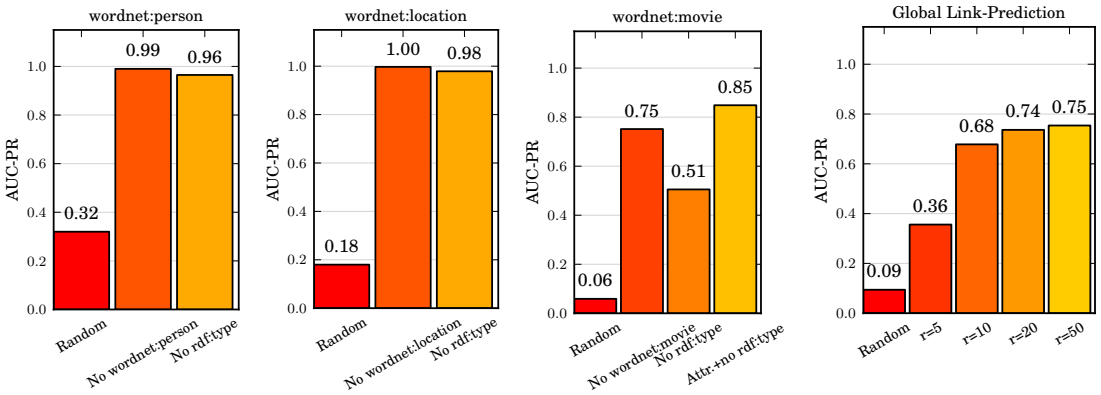


Figure 4: Link-prediction experiments for `rdf:type`

predicate provides a textual description for movies that usually includes the token "films", e.g. "French comedy films", "1997 films" etc. It can be seen from the significantly improved results that RESCAL can detect these regularities by using the attribute extension.

Furthermore, we carried out general link prediction experiments for all relation types in the YAGO 2 ontology. Due to the enormous size of potential triples and the very unbalanced distribution of triples across predicates (e.g. `rdf:type` holds approximately 40 million triples, while `yago:hasNeighor` holds only around 600) we created a train/test-split of the data, by randomly selecting 100 existing and 1000 non-existing triples for each predicate. Figure 4 shows the area under the precision-recall curve for different numbers of latent components. Models for $r = 50$ and higher have been computed on a machine with 64GB RAM, due to memory requirements exceeding 4GB. The results support the intuitive interpretation that abstract and general relations (e.g. `rdf:type wordnet:person`) can be ranked successfully with models of relatively low complexity as they fit nicely to a latent-variable model and have a significant number of examples in the data. However, more detailed or complex relations (e.g. `yago:isMarriedTo`) possibly need a very large number of latent components.

4.2 Collective Learning on the Semantic Web

As previously stated, collective learning is an important feature of an algorithm for learning on the Semantic Web, due to the way data is modeled in RDF. To demonstrate collective learning, we carried out a specifically designed link-prediction experiment on YAGO, with the objective to predict links of the kind `rdf:type wikicategory:<?>_writer`, where `<?>` can be any nationality such as *French*, *American*, etc. Naturally, the classification of a person as a writer of a particular nationality is dependent on the birthplace of the writer in question. Unfortunately, the birthplace of a person is usually not given as a country, for which a clear correlation to the nationality of a person would exist, but as the city the person was born in. However, the city usually has a link, e.g. `yago:isLocatedIn`, to the corresponding country (See Figure 5a for an example of this modeling). In order to be able to predict the correct `rdf:type` triples from the relations `yago:wasBornIn` and `yago:isLocatedIn`, it is necessary for an algorithm to detect the correlations between the type of a writer and the country of the writer's birthplace. Consequently, this is a collective learning problem.

To evaluate the performance of RESCAL on this task, we compared it to algorithms that were previously used for Machine Learning on Semantic Web data. CP [8] is a tensor factorization that has been used in TripleRank [12] to rank RDF data for faceted browsing. Moreover, we included SUNS [17], a relational learning algorithm for large-scale data which has also been applied to RDF data. However, both of these algorithms have only limited collective learning capabilities. We also included MLNs, where we *manually*[8] specified the rule

$$\text{wasBornIn(x,y)} \wedge \text{isLocatedIn(y,+z)} \Rightarrow \text{type(x,+c)}$$

and learned the weights for this rule via the Alchemy package.[9] To create a confined setting, we extracted a subgraph of the YAGO ontology consisting only of American, French, German and Japanese writers, their birthplaces and their respective countries as well as the predicates `yago:wasBornIn`, `yago:isLocatedIn` and `rdf:type`. Learning only on this subgraph has the advantage that it is a very controlled setting, i.e. a relational learning algorithm should only be successful on this data when it can detect the correlation between the `rdf:type` of a person and the country of the person's birthplace. The subgraph was constructed with SPARQL queries such as

```
SELECT ?writer, ?birthPlace, ?location WHERE
{
  ?writer rdf:type wikicategory:French_writer .
  ?writer yago:wasBornIn ?birthPlace .
  ?birthPlace yago:isLocatedIn ?location
}
```

Converting the raw RDF data to a tensor representation, resulted in a tensor of size $404 \times 404 \times 3$. On this data we performed 10-fold cross-validation where the objective was to correctly predict the `rdf:type` relations. In each iteration, all `rdf:type` triples were removed from the test

[8]Unfortunately, we were not able to get reasonable results with structure learning for MLNs.

[9]$+z$ and $+y$ is Alchemy-specific syntax that replaces the variable with all occurring constants

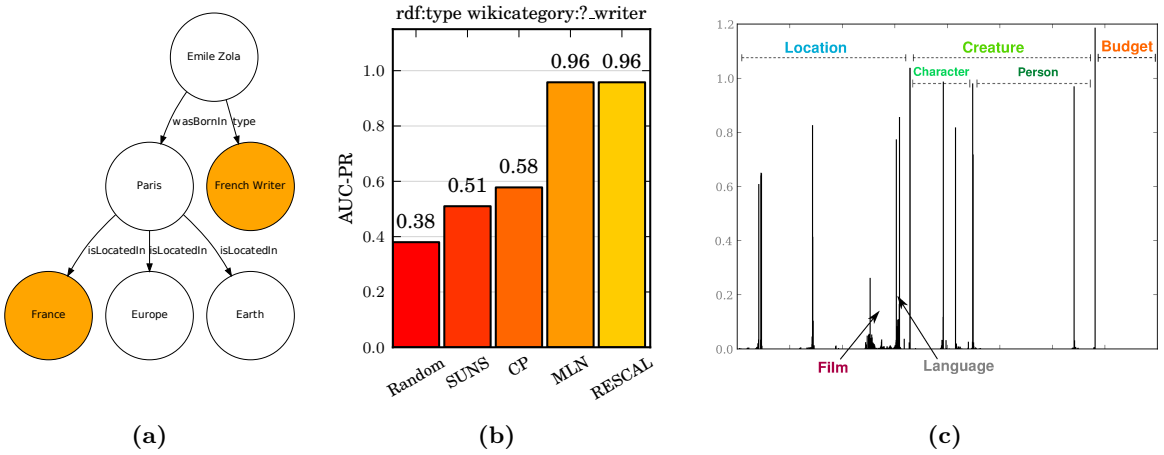


Figure 5: (a) Collective learning example. (b) Results of 10-fold cross-validation on writers data. (c) Reachability-plot of an OPTICS clustering on the IIMB data. "Valleys" in the plot indicate clusters, nested valleys correspond to the cluster hierarchy.

data. Figure 5b shows the results of this experiment. It can be seen that RESCAL is very successful in predicting the correct `rdf:type` triples, while CP and SUNS struggle to learn something meaningful on this data, what is due to their missing collective learning ability.

4.3 Learning Taxonomies

In Section 3.3.3 we briefly described how RESCAL can be used to learn taxonomies on Semantic Web data. To evaluate the applicability of our approach, we conducted experiments with the objective to rebuild an existing taxonomy as closely as possible in a fully unsupervised setting, i.e. only from instance data. For this purpose we used the large version of the IIMB 2010 benchmark provided by the Ontology Alignment Evaluation Initiative,[10] which contains around 1400 instances of a movie domain. These instances are organized in an ontology that consists of 5 distinct top-level concepts, namely `Budget`, `Creature`, `Film`, `Language`, and `Location`. The concepts `Creature`, `Film` and `Location` are again subdivided into multiple concepts such as `Person` and `Character`, `Anime` and `Action Movie` or `Country` and `City`. In total there exist 80 concepts and the maximum subclass-level is 3. A tensor representation of this data is of size $1519 \times 1519 \times 35$. We selected OPTICS [1] as the hierarchical clustering algorithm to work in the latent-component space A. OPTICS is a density-based hierarchical clustering algorithm, which also provides an interpretable visual representation of its results. An example of this representation is shown in Figure 5c. To evaluate the quality of our clustering, we followed the procedure suggested in [30] and assign that F-measure score to a particular concept that is the highest for this concept out of all clusters. The idea behind this approach is that there should exist one cluster for each concept that is pure and holds most of this concept's instances. Table 3 shows the results of our evaluation, using a RESCAL model with $r = 10$ and an OPTICS clustering with $minpts = 1$. It can be seen that our approach achieves good results throughout all levels, especially for top-level concepts. One reason for this behaviour is that, on this level, every concept is represented by a sufficient number of instances, while e.g. some level 2 movie concepts include only two or three instances and therefore are hard to recognize. Although more sophisticated taxonomy-extraction methods could be applied, the results are encouraging since they indicate that A provides a meaningful representation for this task.

[10]available from `http://oaei.ontologymatching.org/2010/im/index.html`

Table 3: F-measure for selected concepts and weighted F-measure for all concepts per subclass-level

Level 1		**Level 2**		**Level 3**	
Locations	0.89	City	0.94	Capital	0.94
Films	1.0	Sci. Fiction	0.81	Director	0.81
Creature	0.96	Character	0.73	C. Creator	0.31
Budget	1.0	Person	0.97	Actor	0.58
Language	0.92	Country	0.59		
All	0.958	**All**	0.76	**All**	0.79

5. CONCLUSION AND FUTURE WORK

We have demonstrated that tensor factorization in form of the RESCAL decomposition is a suitable relational learning approach for the dyadic relational data of the Semantic Web and showed that the presented approach can scale to large knowledge bases. By exploiting the sparsity of LOD-type Semantic Web data we were able to factorize the YAGO 2 core ontology, consisting of millions of entities and known facts, on a single desktop computer. Furthermore, we demonstrated on YAGO 2 that the factorization is able to successfully predict unknown triples on a large scale and validated its efficiency in collective learning.

While sparsity allows RESCAL to scale, it might also become a problem, e.g., when a particular relation is underrepresented and only very complex models have a chance to capture its regularities. Therefore, a possible line of future research is to reduce the effective sparsity of a tensor repre-

sentation by introducing typed relations to the factorization, such that only those tensor entries $\mathcal{X}_{ijk}$ are considered in the factorization that are in compliance with the `rdfs:range` and `rdfs:domain` constraints of predicate k. Furthermore, determining good choices of regularization parameters via cross-validation can be tedious on large-scale data. Efficient methods to finding good parameter values, for instance scalable Bayesian methods such as [33], could provide a solution.

Acknowledgements

We acknowledge funding by the German Federal Ministry of Economy and Technology (BMWi) under the THESEUS project and by the EU FP 7 Large-Scale Integrating Project LarKC. Maximilian Nickel is supported by a graduate fellowship from Siemens AG.

6. REFERENCES

[1] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. In *ACM SIGMOD Record*, volume 28, page 49–60, 1999.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, page 722–735, 2008.

[3] S. Auer and J. Lehmann. Creating knowledge out of interlinked data. *Semantic Web*, 1(1):97–104, Jan. 2010.

[4] B. W. Bader, R. A. Harshman, and T. G. Kolda. Temporal analysis of semantic graphs using ASALSAN. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 33–42, Omaha, NE, USA, Oct. 2007.

[5] V. Bicer, T. Tran, and A. Gossen. Relational kernel machines for learning from Graph-Structured RDF data. *The Semantic Web: Research and Applications*, page 47–62, 2011.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[7] S. Bloehdorn and Y. Sure. Kernel methods for mining instance data in ontologies. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, page 58–71, 2007.

[8] R. Bro. PARAFAC. tutorial and applications. *Chemometrics and Intelligent Laboratory Systems*, 38(2):149–171, 1997.

[9] C. d'Amato, N. Fanizzi, and F. Esposito. Non-parametric statistical learning methods for inductive classifiers in semantic knowledge bases. In *Proceedings of the 2008 IEEE International Conference on Semantic Computing*, page 291–298, Washington, DC, USA, 2008. IEEE Computer Society.

[10] J. Davis and M. Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, page 233–240, 2006.

[11] N. Fanizzi, C. D'Amato, and F. Esposito. DL-FOIL concept learning in description logics. In *Proceedings of the 18th international conference on Inductive Logic Programming*, ILP '08, page 107–121, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] T. Franz, A. Schultz, S. Sizov, and S. Staab. Triplerank: Ranking semantic web data by tensor decomposition. *The Semantic Web-ISWC 2009*, page 213–228, 2009.

[13] H. Halpin, P. Hayes, J. McCusker, D. Mcguinness, and H. Thompson. When owl: sameAs isn't the same: An analysis of identity in linked data. *The Semantic Web–ISWC 2010*, page 305–320, 2010.

[14] S. Hellmann, J. Lehmann, and S. Auer. Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst*, 5(2):25–48, 2009.

[15] P. Hitzler and F. van Harmelen. A reasonable semantic web. *Semantic Web*, 1(1):39–44, 2010.

[16] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. Linked Data on the Web (LDOW 2010), 2010.

[17] Y. Huang, V. Tresp, M. Bundschus, and A. Rettinger. Multivariate structured prediction for learning on semantic web. 2010.

[18] C. Kiefer, A. Bernstein, and A. Locher. Adding data mining support to SPARQL via statistical relational learning methods. In *Proceedings of the 5th European semantic web conference*, pages 478—492, 2008.

[19] S. Kok and P. Domingos. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, page 433–440, 2007.

[20] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455, 2009.

[21] H. T. Lin, N. Koul, and V. Honavar. Learning relational bayesian classifiers from RDF data. In *Proceedings of the International Semantic Web Conference (ISWC 2011)*, 2011. In press.

[22] M. Nickel, V. Tresp, and H. Kriegel. A Three-Way model for collective learning on Multi-Relational data. In *Proceedings of the 28th International Conference on Machine Learning*, ICML '11, pages 809—816, Bellevue, WA, USA, 2011. ACM.

[23] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, page 811–820, 2010.

[24] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1):107–136, 2006.

[25] D. Roy, C. Kemp, V. Mansinghka, and J. Tenenbaum. Learning annotated hierarchies from relational data. *Advances in neural information processing systems*, 19:1185, 2007.

[26] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[27] F. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, page 697–706, 2007.

[28] J. Sun, D. Tao, and C. Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 374–383, 2006.

[29] I. Sutskever, R. Salakhutdinov, and J. B. Tenenbaum. Modelling relational data using bayesian clustered tensor factorization. *Advances in Neural Information Processing Systems*, 22, 2009.

[30] P. Tan, M. Steinbach, V. Kumar, et al. *Introduction to data mining*. Pearson Addison Wesley Boston, 2006.

[31] J. Völker and M. Niepert. Statistical schema induction. *The Semantic Web: Research and Applications*, page 124–138, 2011.

[32] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, page 1113–1120, 2009.

[33] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.