Probabilistic Latent-Factor Database Models

Denis Krompaß[1], Xueyian Jiang[1], Maximilian Nickel[2], and Volker Tresp[1,3]

[1] Ludwig Maximilian University of Munich, Germany
[2] Massachusetts Institute of Technology, Cambridge, MA
and Istituto Italiano di Tecnologia, Genova, Italy
[3] Siemens AG, Corporate Technology, Munich, Germany

Abstract. We describe a general framework for modelling probabilistic databases using factorization approaches. The framework includes tensor-based approaches which have been very successful in modelling triple-oriented databases and also includes recently developed neural network models. We consider the case that the target variable models the existence of a tuple, a continuous quantity associated with a tuple, multiclass variables or count variables. We discuss appropriate cost functions with different parameterizations and optimization approaches. We argue that, in general, some combination of models leads to best predictive results. We present experimental results on the modelling of existential variables and count variables.

1 Introduction

Tensor models have been shown to efficiently model triple-oriented databases [10] where the main goal is to predict the probability for the existence of a triple. Here we generalize the approach in several directions. First, we show that relations with any arity can be modeled, not just triple stores. Second, we show that any set of target variables that is associated with a triple can be modelled. As examples one might predict the rating of a user for an item, the amount of a specific medication for a patient, or the number of times that team A played against team B. In each of these cases a different likelihood model might be appropriate and we discuss different likelihood functions, their different parameterizations and learning algorithms. Third, we discuss a more general framework that includes recently developed neural network models [13, 1]. Finally, we argue that model combinations sometimes offer greater flexibility and predictive power. We present experimental results on the modelling of existential variables and count variables using different likelihood models.

The paper is organized as follows. In the next section we describe the probabilistic setting and in Section 3 we introduce the factorization framework and some specific models. In Section 4 we describe the learning rules and Section 5 contains our experimental results. Section 6 describes extensions. Section 7 contains our conclusions.

2 Probabilistic Database Models

2.1 Database Notation

Consider a database as a set of M relations $\{r^k\}_{k=1}^M$. A relation is a table with attributes as columns and tuples $E_i = \{e_{l(i,1)}, e_{l(i,2)}, \ldots, e_{l(i,L^k)}\}$ as rows where L^k is the number of attributes or the arity of the relation r^k. $l(i,i')$ is the index of the domain entity in tuple E_i in column i'. A relation r^k is closely related to the predicate $r^k(E_i)$, which is a function that maps a tuple to true (or 1) if the E_i belongs to the relation and to false (or 0) otherwise. We model a triple (s,p,o) as a binary relation p where the first column is the subject s and the second column is the object o.

2.2 Probabilistic Database Model

We now associate with each instantiated relation $r^k(E_i)$ a target quantity x_i^k. Formally we increase the arity of the relation by the dimension of x_i^k, so a binary relation would become a ternary relation, if x_i^k is a scalar. Here, the target x_i^k can model different quantities. It can stand for the fact that the tuple exists ($x_i^k = 1$) or does not exist ($x_i^k = 0$) i.e., we model the predicate. In another application E_i might represent a user/item pair and x_i^k is the rating of the user for the item. Alternatively, x_i^k might be a count, for example the number of times that the relation $r^k(E_i)$ has been observed. In the following we form predictive models for x_i^k; thus we can predict, e.g., the likelihood that a tuple is true, or the rating of a user for an item, or the number of times that relation $r^k(E_i)$ has been observed.

2.3 Likelihood Functions and Cost Functions

Convenient likelihood functions originate from the (overdispersed) exponential family of distributions.

Bernoulli. The Bernoulli model is appropriate if the goal is to predict the existence of a tuple, i.e., if we model the predicate. With $x_i^k \in \{0,1\}$, we model

$$P(x_i^k = 1|\theta_i^k) = \theta_i^k$$

with $0 \leq \theta_i^k \leq 1$. From this equation we can derive the penalized log-likelihood cost function

$$\text{lossBe}_i^k = -(x_i^k + \alpha_i^k - 1)\log\theta_i^k - (\beta_i^k + K_i^k - x_i^k)\log(1-\theta_i^k). \quad (1)$$

Here, $K_i^k = 0$; $\alpha_i^k > 0$ and $\beta_i^k > 0$ are derived from the conjugate beta-distribution and can represent virtual data, in the sense that they represent $\alpha_i^k - 1$ additional observations of $x_i^k = 1$ and $\beta_i^k - 1$ additional observations of $x_i^k = 0$. The contribution of the prior drops out with $\alpha_i^k = 1, \beta_i^k = 1$.

Note that we have the constraints that $0 \leq \theta_i^k \leq 1$. A convenient re-parameterization can be achieved using the framework of the exponential family of distributions which suggests the parametrization $\theta_i^k = \text{sig}(\eta_i^k)$, where the natural parameter η_i^k is unconstraint and where $\text{sig}(arg) = 1/(1+\exp(-arg))$ is the logistic function.

Gaussian. The Gaussian model can be used to predict continuous quantities, e.g., the amount of a given medication for a given patient. The Gaussian model is

$$P(x_i^k|\theta_i^k) \propto \exp -\frac{1}{2\sigma^2}(x_i^k - \theta_i^k)^2$$

where we assume that either σ^2 is known or is estimated as a global parameter in a separate process. With a Gaussian likelihood function we get

$$\text{lossG}_i^k = \frac{1}{2(\sigma_i^k)^2}(x_i^k - \theta_i^k)^2 + \frac{1}{2(\alpha_i^k)^2}(c_i^k - \theta_i^k)^2.$$

Note that the first term is simply the squared error. The second term is derived from the conjugate Gaussian distribution and implements another cost term, which can be used to model a prior bias toward a user-specified c_i^k. The contribution of the prior drops out with $\alpha_i^k \to \infty$.

Binomial. If the Bernoulli model represents the outcome of the tossing of one coin, the binomial model corresponds to the event of tossing a coin K times. We get

$$P(x_i^k|\theta_i^k) \propto (\theta_i^k)^{x_i^k}(1-\theta_i^k)^{K-x_i^k}.$$

The cost function is identical to the cost function in the Bernoulli model (Equation 1), only that $K_i^k = K - 1$ and $x_i^k \in \{0, 1, \ldots, K\}$ is the number of observed "heads".

Poisson. Typical relational count data which can be modelled by Poisson distributions are the number of messages sent between users in a given time frame. For the Poisson distribution, we get

$$P(x_i^k|\theta_i^k) \propto (\theta_i^k)^{x_i^k} \exp(-\theta_i^k) \tag{2}$$

and

$$\text{lossP}_i^k = -(x_i^k + \alpha_i^k - 1)\log\theta_i^k + (\beta_i^k + 1)\theta_i^k$$

with $x_i^k \in \mathbb{N}_0$; $\alpha_i^k > 0, \beta_i^k > 0$ are parameters in the conjugate gamma-distribution. The contribution of the prior drops out with $\alpha_i^k = 1, \beta_i^k = 0$. Here, the natural parameter is defined as $\theta_i^k = \exp(\eta_i^k)$. Note that the cost function of the Poisson model is, up to parameter-independent terms, identical to the KL-divergence cost function [3].

Multinomial. The multinomial distribution is often used for textual data where counts correspond to how often a term occurred in a given document. For the multinomial model we get

$$\text{lossM}_i^k = -(x_i^k + \alpha_i^k - 1)\log\theta_i^k$$

with $\theta_i^k \geq 0, \sum_i \theta_i^k = 1$. The natural parameter is defined as $\theta_i^k = \exp(\eta_i^k)$ and for observed counts, $x_i^k \in \mathbb{N}_0$. The contribution of the Dirichlet prior drops out with $\alpha_i^k = 1$.

Ranking Criterion. Finally we consider the ranking criterion which is used in the Bernoulli setting with $x_i^k \in \{0,1\}$. It is not derived from an exponential family model but has successfully been used in triple prediction, e.g., in [13]. Consider a binary relation where the first attribute is the subject and the second attribute is the object. For a known true tuple with $x_i^k = 1$ we define $\text{lossR}_i^k = \sum_{c=1}^{C} \max\left(0, 1 - \theta_i^k + \theta_{i,c}^k\right)$ where $\theta_{i,c}^k$ is randomly chosen from all triples with the same subject and predicate but with a different object with target 0. Thus one scores the correct triple higher than its corrupted one up to a margin of 1. The use of a ranking criterion in relational learning was pioneered by [12] as Bayesian Personalized Ranking (BPR) with a related ranking cost function of the form $\text{lossBPR}_i^k = \sum_{c=1}^{C} \log \text{sig}(\theta_i^k - \theta_{i,c}^k)$.

Interpretation. After modelling, the probability $P(x_i^k|\theta_i^k)$, resp. $P(x_i^k|\eta_i^k)$, can be interpreted as the plausibility of the observation given the model. For example, in the Bernoulli model we can evaluate how plausible an observed tuple is and we can predict which unobserved tuples would very likely be true under the model.

3 A Framework for Latent-Factor Models

3.1 The General Setting

We consider two models where all relations have the same arity L^k. In the *multi-task setting*, we assume the model

$$\theta^k_{E_i=\{e_{l(i,1)}, e_{l(i,2)}, \ldots, e_{l(i,L^k)}\}} = f_{w^k}\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}\right). \tag{3}$$

Here a_l is a vector of $\gamma \in \mathbb{N}$ latent factors associated with e_l to be optimized during the training phase.[4] $l(i, i')$ maps attribute i' of tuple E_i to the index of the entity. This is a multi-task setting in the sense that for each relation r^k a separate function with parameters w^k is modelled.

In the *single-task setting*, we assume the model

$$\theta^k_{E_i=\{e_{l(i,1)}, e_{l(i,2)}, \ldots, e_{l(i,L^k)}\}} = f_{w}\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}, \tilde{a}_k\right).$$

Note that here we consider a single function with parameter vector w where a relation is represented by its latent factor $\tilde{a}_k$.

In case that we work with natural parameters, we would replace $\theta^k_{E_i}$ with $\eta^k_{E_i}$ in the last two equations.

3.2 Predictive Models

We now discuss models for $f_{w^k}(\cdot)$ and $f_w(\cdot)$. Note that not only the model weights are uncertain but also the latent factors of the entities. We first describe

[4] Here we assume that the rank γ is the same for all entities; this assumption can be relaxed in some models.

tensor approaches for the multi-task setting and the single-task setting and then describe two neural network models.

Tensor Models for the Multi-task Setting. Here, the model is

$$f_{w^k}\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}\right) \tag{4}$$
$$= \sum_{s_1=1}^{\gamma} \sum_{s_2=1}^{\gamma} \cdots \sum_{s_{L^k}=1}^{\gamma} w^k_{s_1,s_2,\ldots s_{L^k}} \left(a_{l(i,1),s_1} a_{l(i,2),s_2} \ldots a_{l(i,L^k),s_{L^k}}\right).$$

This equation describes a RESCAL model which is a special case of a Tucker tensor model with the constraint that an entity has a unique latent representation, independent of where it appears in a relation [10]. This property is important to achieve relational collective learning [10].

In the original RESCAL model, one considers binary relations with $L^k = 2$ (RESCAL2). Here A with $(A)_{l,s} = a_{l,s}$ is the matrix of all latent representations of all entities. Then Equation 4 can be written in tensor form as

$$\mathcal{F} = \mathcal{R} \times_1 A \times_2 A$$

with tensor $(\mathcal{F})_{l_1,l_2,k} = f_{w^k}(a_{l_1}, a_{l_2})$ and core tensor $(\mathcal{R})_{s_1,s_2,k} = w^k_{s_1,s_2}$.

Tensor Models for the Single-task Setting. Here, the model is

$$f_w\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}, \tilde{a}_k\right) \tag{5}$$
$$= \sum_{s_1=1}^{\gamma} \sum_{s_2=1}^{\gamma} \cdots \sum_{s_{L^k}=1}^{\gamma} \sum_{t=1}^{\gamma} w_{s_1,s_2,\ldots s_{L^k},t} \left(a_{l(i,1),s_1} a_{l(i,2),s_2} \ldots a_{l(i,L^k),s_{L^k}} \tilde{a}_{k,t}\right).$$

Note that the main difference is that now the relation is represented by its own latent factor $\tilde{a}_k$. Again, this equation describes a RESCAL model. For binary relations one speaks of a RESCAL3 model and Equation 5 becomes

$$\mathcal{F} = \mathcal{R} \times_1 A \times_2 A \times_3 \tilde{A}$$

where $(\tilde{A})_{k,t} = \tilde{a}_{k,t}$ and the core tensor is $(\mathcal{R})_{s_1,s_2,t} = w_{s_1,s_2,t}$.

If $\tilde{a}_{k,t}$ is a unit vector with the 1 at $k = t$, then we recover the multi-task setting. If all weights are 0, except for "diagonal" weights with $s_1 = s_2 = \ldots = s_{L^k} = t$, this is a PARAFAC model and only a single sum remains. The PARAFAC model is used in the factorization machines [12]. In factorization machines, attributes with ordinal or real values are modelled by $\bar{a}_{z(i)} = z(i)\bar{a}$ where $z(i)$ is the value of the attribute in E_i and $\bar{a}$ is a latent factor vector for the attribute independent of the particular value $z(i)$.

Please note that the L^k-order polynomials also contain all lower-order polynomials, if we set, e.g., $a_{l,1} = 1$, $\forall l$. In the factorization machine, the order of the polynomials is typically limited to 1 or 2, i.e. all higher-order polynomials obtain a weight of 0.

Neural Tensor Networks. Here, the model is

$$f_{w^k, v^k}\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}\right)$$

$$= \sum_{h=1}^{H} w_h^k \operatorname{sig}\left(\sum_{s_1=1}^{\gamma} \sum_{s_2=1}^{\gamma} \cdots \sum_{s_{L^k}=1}^{\gamma} v^{h,k}_{s_1, s_2, \ldots s_{L^k}} \left(a_{l(i,1), s_1} a_{l(i,2), s_2} \cdots a_{l(i,L^k), s_{L^k}}\right)\right).$$

The output is a weighted combination of the logistic function applied to H different tensor models. This is the model used in [13], where the $\tanh(\cdot)$ was used instead of the logistic function.

Google Vault Model. Here a neural network is used of the form

$$f_w\left(a_{l(i,1)}, a_{l(i,2)}, \ldots, a_{l(i,L^k)}, \tilde{a}_k\right)$$

$$= \sum_{h=1}^{H} w_h^k \operatorname{sig}\left(\sum_{s_1=1}^{\gamma} v_{1,s_1} a_{l(i,1), s_1} + \ldots + \sum_{s_{L^k}=1}^{\gamma} v_{L^k, s_{L^k}} a_{l(i,L^k), s_{L^k}} + \sum_{t=1}^{\gamma} \tilde{v}_t \tilde{a}_{k,t}\right).$$

The latent factors are simply the inputs to a neural network with one hidden layer. This model was used in [1] in context of the Google Knowledge Graph. It is related to tensor models for the single-task setting where the fixed polynomial basis functions are replaced by adaptable neural basis functions with logistic transfer functions.

4 Parameter Estimates

4.1 Missing Values

Complete Data. This means that for all relevant tuples the target variables are available.

Assumed Complete Data. This is mostly relevant when x_i^k is an existential variable, where one might assume that tuples that are not listed in the relation are false. Mathematically, we then obtain a complete data model and this is the setting in our experiments. Another interpretation would be that with sparse data $x_i^k = 0$ is a correct imputation for those tuples.

Missing at Random. This is relevant, e.g, when x_i^k represents a rating. Missing ratings might be missing at random and the corresponding tuples should be ignored in the cost function. Computationally, this can most efficiently be exploited by gradient-based optimization methods (see Section 4.3). Alternatively one can use α_i^k and β_i^k to implement prior knowledge about missing data.

Ranking Criterion. On the ranking criterion one does not really care if unobserved tuples are unknown or untrue, one only insists that the observed tuples should obtain a higher score by a margin than unobserved tuples.

4.2 Regularization

In all approaches the parameters and latent factors are regularized with penalty term $\lambda_A \|A\|_{\mathbf{F}}$ and $\lambda_W \|W\|_{\mathbf{F}}$ where $\|\cdot\|_{\mathbf{F}}$ indicates the Frobenius norm and where $\lambda_A \geq 0$ and $\lambda_W \geq 0$ are regularization parameters.

4.3 Optimizing the Cost Functions

Alternating Least Squares. The minimization of the Gaussian cost function lossG with complete data can be implemented via very efficient alternating least squares (ALS) iterative updates, effectively exploiting data sparsity in the (assumed) complete data setting [10, 7]. For example, RESCAL has been scaled up to work with several million entities and close to 100 relation types. The number of possible tuples that can be predicted is the square of the number of entities times the number of predicates: for example RESCAL has been applied to the Yago ontology with 10^{14} potential tuples [11].

Natural Parameters: Gradient-Based Optimization. When natural parameters are used, unconstrained gradient-based optimization routines like L-BFGS can be employed, see for example [6, 9].

Non-Negative Tensor Factorization. If we use the basis representation with θ_i^k parameters, we need to enforce that $\theta_i^k \geq 0$. One option is to employ non-negative tensor factorization which leads to non-negative factors and weights. For implementation details, consult [3].

Stochastic Gradient Descent (SGD). In principal, SGD could be applied to any setting with any cost function. In our experiments, SGD did not converge to any reasonable solutions in tolerable training time with cost functions from the exponential family of distributions and (assumed) complete data. SGD and batch SGD were successfully used with ranking cost functions in [12, 13] and we also achieved reasonable results with BPR.

5 Experiments

Due to space limitations we only report experiments using the binary multi-task model RESCAL2. We performed experiments on three commonly used benchmark data sets for relational learning:

Kinship 104 entities and $M = 26$ relations that consist of several kinship relations within the Alwayarra tribe.

Nations 14 entities and $M = 56$ relations that consist of relations between nations (treaties, immigration, etc). Additionally the data set contains attribute information for each entity.

UMLS 135 entities and $M = 49$ relations that consist of biomedical relationships between categorized concepts of the Unified Medical Language System (UMLS).

5.1 Experiments with Different Cost Functions and Representations

Here $x_i^k = 1$ stands for the existence of a tuple, otherwise $x_i^k = 0$. We evaluated the different methods using the area under the precision-recall curve (AUPRC) performing 10-fold cross-validation. Table 1 shows results for the three data sets ("nn" stands for non-negative and "nat" for the usage of natural parameters). In all cases, the RESCAL model with lossG ("RESCAL") gives excellent

Table 1. AUPRC for Different Data Sets

	Rand	RESCAL	nnPoiss	nnMulti	natBern	natPoiss	natMulti	SGD	stdev
Nations	0.212	0.843	0.710	0.704	**0.850**	**0.847**	0.659	0.825	0.05
Kinship	0.039	0.962	0.918	0.889	**0.980**	**0.981**	0.976	0.931	0.01
UMLS	0.008	**0.986**	0.968	0.916	**0.986**	0.967	0.922	0.971	0.01

Table 2. AUPRC for Count Data

	Rand	RESCAL	nnPoiss	nnMulti	natBin	natPoiss	natMulti	RES-P	stdev
Nations	0.181	0.627	0.616	0.609	**0.637**	0.632	0.515	**0.638**	0.01
Kinship	0.035	0.949	0.933	0.930	**0.950**	**0.952**	**0.951**	**0.951**	0.01
UMLS	0.007	0.795	0.790	0.759	**0.806**	**0.806**	0.773	**0.806**	0.01

performance and the Bernoulli likelihood with natural parameters ("natBern") performs even slightly better. The Poisson model with natural parameters also performs quite well. The performance of the multinomial models is significantly worse. We also looked at the sparsity of the solutions. As can be expected only the models employing non-negative factorization lead to sparse models. For the Kinship data set, only approximately 2% of the coefficients are nonzero, whereas models using natural parameters are dense. SGD with the BPR ranking criterion and AdaGrad batch optimization was slightly worse than RESCAL.

Another issue is the run-time performance. RESCAL with lossG is fastest since the ALS updates can efficiently exploit data sparsity, taking 1.9 seconds on Kinship on an average Laptop (Intel(R) Core(TM) i5-3320M with 2.60 GHz). It is well-known that the non-negative multiplicative updates are slower, having to consider the constraints, and take approximately 90 seconds on Kinship. Both the non-negative Poisson model and the non-negative multinomial model can exploit data sparsity. The exponential family approaches using natural parameters are slowest, since they have to construct estimates for all ground atoms in the (assumed) complete-data setting, taking approximately 300 seconds on Kinship. SGD converges in 108 seconds on Kinship.

5.2 Experiments on Count Data

Here $x_i^k \in \mathbb{N}_0$ is the number of observed counts. Table 2 shows results where we generated 10 database instances (worlds) from a trained Bernoulli model and generated count data from 9 database instances and used the tenth instance for testing. Although RESCAL still gives very good performance, best results are

Table 3. AUPRC for Combined Models

	RESCAL2	SUNS-S	SUNS-P	Combined	stdev
Nations	0.855	0.761	0.831	**0.886**	0.01
Kinship	0.960	0.916	0.004	**0.968**	0.01
UMLS	**0.979**	0.942	0.293	**0.980**	0.01

obtained by models more appropriate for count data, i.e., the binomial model and the Poisson model using natural parameters. The non-negative models are slightly worse than the models using natural parameters.

5.3 Probabilities with RESCAL

Due to its excellent performance and computational efficiency, it would be very desirable to use the RESCAL model with lossG and ALS, whenever possible. As discussed in [5], by applying post-transformations RESCAL predictions can be mapped to probabilities in Bernoulli experiments. For Poisson data we can assume a natural parameter model with $\alpha_i^k = 2$ and model $\tilde{x}_i^k = \log(1 + x_i^k)$ which leads to a sparse data representation that can efficiently be modelled with RESCAL and lossG. The results are shown as RES-P in Table 2 which are among the best results for the count data!

6 Extensions

6.1 SUNS Models

Consider a triple store. In addition to the models described in Section 3 we can also consider the following three model for $f(\mathit{subject}, \mathit{predicate}, \mathit{object})$

$$\sum_{m=1}^{\gamma}\sum_{u=1}^{\gamma} a_{subject,m} a_u^{po}, \quad \sum_{m=1}^{\gamma}\sum_{u=1}^{\gamma} a_{object,m} a_u^{sp}, \quad \sum_{m=1}^{\gamma}\sum_{u=1}^{\gamma} a_{predicate,m} a_u^{so}.$$

These are three Tucker1 models and were used as SUNS models (SUNS-S, SUNS-O, SUNS-P) in [14]. a^{po}, a^{sp}, and a^{so} are latent representations of (p, o), (s, p), and (s, o), respectively.

6.2 Model Combinations

The different SUNS models and RESCAL models have different modelling capabilities and often a combination of several models gives best results [2, 8]. Table 3 shows the performance for the RESCAL model, two SUNS models and the performance of an additive model of all three models. For Nations, SUNS-P performs well and boosts the performance of the combined model. SUNS-P can model correlations between relations, e.g., between *likes* and *loves.*

7 Conclusions

We have presented a general framework for modelling probabilistic databases with factor models. When data are complete and sparse, the RESCAL model with a Gaussian likelihood function and ALS-updates is most efficient and highly scalable. We show that this model is also applicable for binary data and for count data. Non-negative modelling approaches give very sparse factors but performance decreases slightly. An issue is the model rank γ. In [8] it has been shown that the rank can be reduced by using a combination of a factor model with a model for local interactions, modelling for example the triangle rule. Similarly, the exploitation of type-constraints can drastically reduce the number of plausible tuples and reduces computational load dramatically [4, 1].

Acknowledgements. M. N. acknowledges support by the Center for Brains, Minds and Machines, funded by NSF STC award CCF-1231216.

References

1. X. L. 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 *KDD*, 2014.
2. X. Jiang, V. Tresp, Y. Huang, and M. Nickel. Link prediction in multi-relational graphs using additive models. In *SeRSy workshop, ISWC*, 2012.
3. D. Krompaß, M. Nickel, X. Jiang, and V. Tresp. Non-negative tensor factorization with rescal. In *Tensor Methods for Machine Learning, ECML workshop*, 2013.
4. D. Krompaß, M. Nickel, and V. Tresp. Factorizing large heterogeneous multi-relational-data. In *Int. Conf. on Data Science and Advanced Analytics*, 2014.
5. D. Krompaß, M. Nickel, and V. Tresp. Querying factorized probabilistic triple databases. In *ISWC*, 2014.
6. B. London, T. Rekatsinas, B. Huang, and L. Getoor. Multi-relational learning using weighted tensor decomposition with modular loss. In *arXiv:1303.1733*, 2013.
7. M. Nickel. *Tensor factorization for relational learning.* PhD-thesis, Ludwig-Maximilian-University of Munich, Aug. 2013.
8. M. Nickel, X. Jiang, and V. Tresp. Learning from latent and observable patterns in multi-relational data. In *NIPS*, 2014.
9. M. Nickel and V. Tresp. Logistic tensor factorization for multi-relational data. In *WSTRUC WS at the ICML*, 2013.
10. M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.
11. M. Nickel, V. Tresp, and H.-P. Kriegel. Factorizing yago: scalable machine learning for linked data. In *WWW*, 2012.
12. S. Rendle, L. B. Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD*, 2009.
13. R. Socher, D. Chen, C. D. Manning, and A. Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 2013.
14. V. Tresp, Y. Huang, M. Bundschus, and A. Rettinger. Materializing and querying learned knowledge. In *IRMLeS, ESWC workshop*, 2009.