Link Prediction in Knowledge Graphs with Probabilistic Logic Programming: Work in Progress

Damiano Azzolini[1], Elisabetta Gentili[2] and Fabrizio Riguzzi[3]

[1]*Dipartimento di Scienze dell'Ambiente e della Prevenzione – University of Ferrara*
[2]*Dipartimento di Ingegneria – University of Ferrara*
[3]*Dipartimento di Matematica e Informatica – University of Ferrara*

Abstract
Liftable probabilistic logic programs are a recently proposed restriction of probabilistic logic programs that impose a particular structure on the clauses such that inference can be performed in a lifted way. In this paper, we discuss a work in progress to perform link prediction in knowledge graphs by learning liftable probabilistic logic programs via regularization.

Keywords
Probabilistic Logic Programming, Knowledge Graph, Link Prediction.

1. Proposed Approach

Probabilistic Logic Programming [1] is one of the possible choices to express uncertainty with a logic-based language. A Logic Program with Annotated Disjunction [2] allows logical rules of the form:

$$h_1 : \pi_1; \ldots; h_m : \pi_m :- b_1, \ldots, b_n$$

with the meaning that, if the conjunction of literals b_i (the *body*) is true, then h_1 can be selected with probability $\pi_1, \ldots, h_m$ with probability π_m, where $\sum_i \pi_i = 1$. If the sum of the π_i is less than 1, an additional implicit head atom is considered, with probability $1 - \sum_i \pi_i$. To perform inference, these programs are converted into a compact form through a process called knowledge compilation [3]. In this new form, inference is much cheaper, but still remains in the #P complexity class.

The authors of [4] proposed *liftable* probabilistic logic programs. These programs contain only rules of the form

$$h : \pi :- b_1, \ldots, b_n.$$

The atom h has a distinguished predicate t/n, called *target*, and all the rules in the program have only t/n in the head. The literals b_i are defined by means of certain facts and rules (i.e., non probabilistic) built over *input* predicates. The computation of the probability of an atom q for the target predicate t/n (i.e., of a ground instantiation q of t/n) in a program with only these type of rules can be performed without knowledge compilation: first, we find the number

PLP 2023: The Tenth Workshop on Probabilistic Logic Programming
damiano.azzolini@unife.it (D. Azzolini); elisabetta.gentili1@unife.it (E. Gentili); fabrizio.riguzzi@unife.it (F. Riguzzi)

of ground instantiations of the clauses with q in the head and the body true. Each of these instantiations corresponds to a Boolean random variable X_i that is true with probability π_i and false with probability $1 - \pi_i$. Due to the structure of the clauses, the query q is true if at least one of the X_i is true. To compute the probability of q we can notice that all the X_is are mutually independent and that the probability that none of these is true is $\prod_{i=i}^{n}(1 - \pi_i)^{m_i}$ where m_i is the number of ground instantiations for a rule R_i and n is the total number of probabilistic clauses. The probability that q is true can be computed as 1 minus the probability that it is false. That is, $P(q) = 1 - \prod_{i=i}^{n}(1 - \pi_i)^{m_i}$. Overall, we just need to know the number of instantiations with the body true to compute the probability of a query.

This language, and probabilistic logic languages in general, are useful in the context of machine learning where usually we are given a set of interpretations (often called mega-interpretations) that define the input predicates. Each mega-interpretation is associated with a set of positive and negative examples, that are a set of atoms for the target predicate whose probabilities should be maximized and minimized respectively.

Knowledge Graph (KG) completion is a research field that attracted a lot of interest in the last years, with several approaches [5, 6, 7, 8, 9]. A knowledge graph is a set of triples $\langle s, p, o \rangle$ where s is the subject, p is the predicate, and o is the object. An example of a triple is $\langle bob, likes, gardening \rangle$. The knowledge graph completion task requires learning rules to predict the object or the subject given the predicate and the subject or the object respectively. These rules are usually in the form of First Order Rules with associated weights.

In this paper, we propose to learn liftable probabilistic logic programs for knowledge graph completion. The pipeline is the following: first, we define an appropriate language bias, i.e., a set of templates that specify the form of the learnable rules, such as the target predicate, its arity, and the literals that can appear in the body by means of mode declarations in the style of Progol [10]. Then, we iteratively apply LIFTCOVER+, a modified version of LIFTCOVER [4], to learn a program and its parameters (i.e., the probabilities of the head atoms). The goal of the learning task is to maximize the log likelihood of the examples, i.e., solve

$$argmax_P \left(\sum_{e \in e^+} logP(e_i) + \sum_{e \in e^-} log(1 - P(e_i)) \right)$$

where e^+ and e^- are respectively positive and negative examples and P is the program. Negative examples can be generated starting from some ground atoms from the KG of interest by changing the subject or the object. To search for candidate programs, LIFTCOVER+ performs a beam search over the space of clauses defined by the language bias. At the end of the clause search phase a set of promising clauses is found. Recently, the authors of [11] proposed to adopt *regularization* during the learning of *hierarchical* probabilistic logic programs, another restriction of probabilistic logic programs where rules are organized in a hierarchical way, where the goal is to set as many parameters as possible to 0, to keep only the relevant clauses. To learn the structure, LIFTCOVER+ learns the parameters with regularization over the program containing all promising clauses. After parameter learning, the clauses with a probability below a fixed threshold are discarded. In this sense it differs from LIFTCOVER, which learned the structure using greedy search.

Parameter learning involves the following objective function

$$argmax_\pi \left(\sum_{e \in e^+} logP(e_i) + \sum_{e \in e^-} log(1 - P(e_i)) \right)$$

that can be solved either with EM or gradient descent. In the first case, we can apply L1, L2 or Bayesian regularization. The L1 and L2 objective functions for the Maximization phase of EM are respectively [11]

$$J_1(\pi) = N_1 \cdot log\pi + N_0 \cdot log(1 - \pi) - \gamma\pi$$

and

$$J_2(\pi) = N_1 \cdot log\pi + N_0 \cdot log(1 - \pi) - \frac{\gamma}{2}\pi^2$$

where N_0 and N_1 are respectively the expected values for the number of times the variable associated with each rule takes value false and true and π is the value of the parameters (probabilities) and γ is a hyperparameter. The value of π that maximizes J_1 and J_2 can be analytically computed [11]. Differently, Bayesian regularization [12] consists in a Bayesian update of the parameters assuming a prior that takes the form of a Dirichlet with parameters $[a, b]$.

In the case of gradient descent, we can apply L1 or L2 regularization to the loss function to minimize that become respectively [11]

$$err_{L1} = \sum_{i=1}^{N} -y_i \cdot logP(e_i) - (1 - y_i) \cdot log(1 - P(e_i)) + \gamma \sum_{i=1}^{k} |\pi_i|$$

and

$$err_{L2} = \sum_{i=1}^{N} -y_i \cdot logP(e_i) - (1 - y_i) \cdot log(1 - P(e_i)) + \frac{\gamma}{2} \sum_{i=1}^{k} \pi_i^2$$

where N is the total number of examples, y_i indicates whether an example is positive ($y_i = 1$) or negative ($y_i = 0$), k is the number of parameters (the probabilities of the clauses), π_i represents the parameters of the clauses, e_i is the i-th example, and γ is a hyperparameter.

Preliminary experiments have shown that naively applying LIFTCOVER+ to commonly used KG datasets is not feasible, because the clause search space is too broad: the resulting programs have very low performance. Thus, we are investigating approaches based on learning chain rules of increasing length, as in [13]: at each iteration, we learn a set of rules of a fixed length together with their parameters and apply regularization to prune the ones with negligible probability. Then, we increase the rule length and continue this process until a certain accuracy is reached.

References

[1] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning, River Publishers, Gistrup, Denmark, 2018.

[2] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: B. Demoen, V. Lifschitz (Eds.), 20th International Conference on Logic Programming (ICLP 2004), volume 3131 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 431–445. doi:10.1007/978-3-540-27775-0_30.

[3] A. Darwiche, P. Marquis, A knowledge compilation map, Journal of Artificial Intelligence Research 17 (2002) 229–264. doi:10.1613/jair.989.

[4] A. Nguembang Fadja, F. Riguzzi, Lifted discriminative learning of probabilistic logic programs, Machine Learning 108 (2019) 1111–1135. doi:10.1007/s10994-018-5750-0.

[5] L. Galárraga, C. Teflioudi, K. Hose, F. M. Suchanek, Fast rule mining in ontological knowledge bases with amie++, The VLDB Journal 24 (2015) 707–730. doi:10.1007/s00778-015-0394-1.

[6] A. Sadeghian, M. Armandpour, P. Ding, D. Z. Wang, DRUM: end-to-end differentiable rule mining on knowledge graphs, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 15321–15331.

[7] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: C. E. Brodley, P. Stone (Eds.), Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada, AAAI Press, 2014, pp. 1112–1119.

[8] H. Wu, Z. Wang, K. Wang, Y. Shen, Learning typed rules over knowledge graphs, in: G. Kern-Isberner, G. Lakemeyer, T. Meyer (Eds.), Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022, 2022.

[9] F. Yang, Z. Yang, W. W. Cohen, Differentiable learning of logical rules for knowledge base reasoning, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 2316–2325.

[10] S. Muggleton, Inverse entailment and Progol, New Generation Computing 13 (1995) 245–286.

[11] A. Nguembang Fadja, F. Riguzzi, E. Lamma, Learning hierarchical probabilistic logic programs, Machine Learning 110 (2021) 1637–1693. doi:10.1007/s10994-021-06016-4.

[12] F. Burden, D. Winkler, Bayesian Regularization of Neural Networks, Humana Press, Totowa, NJ, 2009, pp. 23–42. doi:10.1007/978-1-60327-101-1_3.

[13] C. Meilicke, M. W. Chekol, D. Ruffinelli, H. Stuckenschmidt, Anytime bottom-up rule learning for knowledge graph completion, in: S. Kraus (Ed.), Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, ijcai.org, 2019, pp. 3137–3143.