Systems and Learning Algorithms for Probabilistic Logical Knowledge Bases

Giuseppe Cota[1]
Supervisors: Evelina Lamma[1], Fabrizio Riguzzi[2]

[1] Dipartimento di Ingegneria – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[giuseppe.cota,evelina.lamma,fabrizio.riguzzi]@unife.it

Abstract. In real world domains the information is often uncertain, hence it is of foremost importance to be able to model uncertainty and to reason over it. In this paper we show tools and learning systems under development for probabilistic structured data. Four systems will be considered and an overview of the related issues and of future work will be given. The first described system is `cplint` on SWISH, a web application that allows the user to write Probabilistic Logic Programs and submit the computation of the probability of queries with a web browser. Then two distributed structure learning algorithm are illustrated: SEMPRE ("distributed Structure lEarning by MaPREduce") and LEAP^{MR} ("LEArning Probabilistic description logics by MapReduce"), the former learns new clauses of Probabilistic Logic Programs, the latter is used in the context of Probabilistic Description Logics. The last system taken into account is SML-Bench, developed by the research group AKSW of Leipzig, a benchmarking tool for structured data that has been extended to deal with algorithms for probabilistic structured data.

Keywords: Probabilistic Structured Data, Probabilistic Logic Programming, Probabilistic Description Logics, Structure Learning, MapReduce

1 Introduction

Representing uncertain information and being able to reason over it is of foremost importance for real world applications. In the last decades several semantics where proposed to represent uncertainty, one of the most prominent approaches for representing probabilistic information in Logic Programming is the distribution semantics [15]. This semantics is at the basis of many languages, such as Independent Choice Logic, PRISM, Logic Programs with Annotated Disjunctions (LPADs) and ProbLog.

In [3] the authors proposed an application of the distribution semantics to Description Logics (DLs) and called the resulting semantics DISPONTE ("DIstribution Semantics for Probabilistic ONTologiEs").

With these two semantics it is possible to build algorithms and applications for Probabilistic Logic Programming and Description Logics. Here we discuss the current development of tools and learning systems for probabilistic logical knowledge bases (KBs) that follow either the distribution semantics or DISPONTE.

We proceed as follows. Section 2 provides a brief introduction to the syntax and the semantics on which the discussed systems are based. Section 3 presents `cplint` on SWISH, a web application that allows the user to write Probabilistic Logic Programs and submit the computation of the probability of queries with a web browser. Section 4 illustrates SEMPRE and LEAP$^{\mathrm{MR}}$, two distributed structure learning algorithm. Section 5 quickly describe the changes made to SML-Bench, a benchmarking tool developed by the research group AKSW of Leipzig, in order to deal with algorithms for probabilistic structured data. Finally section 6 discuss the future work and draws conclusions.

2 Syntax and Distribution Semantics

LPADs [16] consist of a finite set of annotated disjunctive clauses C_i of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. Here, $b_{i1}, \ldots, b_{im_i}$ are logical literals which form the *body* of C_i, denoted by $body(C_i)$, while $h_{i1}, \ldots h_{in_i}$ are logical atoms and $\{\Pi_{i1}, \ldots, \Pi_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. Otherwise, if $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$.

Given an LPAD P, the grounding $ground(P)$ is obtained by replacing variables with terms from the Herbrand universe in all possible ways. If P does not contain function symbols and P is finite, $ground(P)$ is finite as well. $ground(P)$ is still an LPAD from which we can obtain a normal logic program by selecting a head atom for each ground clause. In this way we obtain a so-called *world* to which we can assign a probability by multiplying the probabilities of all the head atoms chosen. We thus get a probability distribution over worlds from which we can define a probability distribution over the truth values of a ground atom: the probability of an atom q being true is the sum of the probabilities of the worlds where q is true[3].

Description Logics (DLs) are a family of logic based knowledge representation formalisms which are of particular interest for representing ontologies and for the Semantic Web. For an extensive introduction to DLs we refer to [2].

DISPONTE [3], like the distribution semantics, defines a probability distribution over regular knowledge bases (also called worlds). A probabilistic knowledge base is a set of certain axioms or probabilistic axioms. Certain axioms take the form of regular DL axioms. Probabilistic axioms take the form $p :: E$ where p is a real number in $[0, 1]$. To create a world, we decide whether to include or not each probabilistic axiom, then we multiply the probability of the choices done to

[3] We assume that the worlds all have a two-valued well-founded model.

compute the probability of the world. The probability of a query is then obtained from the joint probability of the worlds and the query by marginalization.

3 cplint on SWISH

To reach a wider audience and popularize Probabilistic Logic Programming we developed `cplint` on SWISH [13]. This is a web application for running the Probabilistic Logic Programming system `cplint` [10] with just a web browser: the algorithms run on a server and the user can post queries and see the results in his browser. The application is available at `http://cplint.lamping.unife.it`. In recent times the system has been extended with the inclusion of algorithms for computing conditional probabilities with exact, rejection sampling and Metropolis-Hasting methods. Moreover, the system now allows hybrid programs, i.e., programs where some of the random variables are continuous. To perform inference on such programs likelihood weighting is used that makes it possible to also have evidence on continuous variables. `cplint` on SWISH offers also the possibility of sampling arguments of goals, a kind of inference rarely considered but useful especially when the arguments are continuous variables. Finally, `cplint` on SWISH offers the possibility of graphing the results, for example by drawing the distribution of the sampled continuous arguments of goals.

4 Distributed Structure Learning Systems

In order to reduce the learning time, we tried to distribute it by using a MapReduce approach. Two algorithms has been proposed: SEMPRE and LEAP$^{\mathrm{MR}}$. The former learns new clauses for Probabilistic Logic Programs that follow the distribution semantics [15], the latter learns new axioms for Probabilistic Description Logics that follow DISPONTE [3].

SEMPRE [14] parallelizes three operations of the structure learning algorithm SLIPCOVER [5] by employing n workers, one master and $n-1$ slaves. All the workers initially receive all the input data.

The first operation is scoring clause refinements: when the revisions for a clause are generated, the master process splits them evenly into n subsets. Then, SEMPRE enters the *Map phase*, where each worker scores a set of refinements and returns with their log-likelihood (LL). Scoring is performed using (serial) EMBLEM [4] which is run over a theory containing only one clause. Once the master has received all sets of scored refinements from the workers, it enters the *Reduce phase*, where it updates the beam of promising clauses and the sets of target and background clauses (TC and BC respectively).

The second parallelized operation is parameter learning for the theories. In this phase, each clause from TC is tentatively added to the theory. In the end, it contains all the clauses that improved its LL (search in the space of theories). During this phase a MapReduce version of EMBLEM called EMBLEM$^{\mathrm{MR}}$ is used.

The third parallelized operation is the final parameter optimization for the theory including also the background clauses. All the background clauses are added to the theory previously learned and the parameters of the theory are learned by means of EMBLEM^{MR}.

SEMPRE was tested on the following seven real world datasets: Hepatitis, Mutagenesis, UWCSE, Carcinogenesis, IMDB, HIV and WebKB. The speedup is always larger than 1 and grows with the number of workers, except for HIV and IMDB, where there is a slight decrease for 16 and 32 workers due to the overhead; however, these two datasets were the smallest and less in need of a parallel solution.

LEAP^{MR} [6] is an evolution of the LEAP system [12] that performs structure and parameter learning of probabilistic ontologies under DISPONTE. While the latter exploits EDGE [11], the former was adapted to be able to perform EDGE^{MR} [7]. EDGE is a system for learning the parameters of DISPONTE KB and EDGE^{MR} is its distributed version.

In order to learn an ontology, LEAP^{MR} first searches for good candidate probabilistic subsumption axioms by means of CELOE [9], then it performs a greedy search in the space of theories using EDGE^{MR} to learn the parameters and to evaluate the theories using the log-likelihood as heuristic.

LEAP^{MR} takes as input the knowledge base $\mathcal{K}$ and the configuration settings for CELOE and EDGE^{MR}, then generates a set of candidate axioms by exploiting CELOE and the sets of positive and negative examples (concept membership axioms) for EDGE^{MR}. Then LEAP^{MR} adds to $\mathcal{K}$ one probabilistic subsumption axiom at a time. After each addition, EDGE^{MR} is performed on the extended KB to compute the LL of the data and the parameters. If the LL is better than the current best, the new axiom is kept in the knowledge base and the parameters of probabilistic axioms are updated, otherwise the learned axiom is removed from the ontology and the previous parameters are restored. The final theory is obtained from the union of the initial ontology and the probabilistic axioms learned.

In order to test how much the exploitation of EDGE^{MR} can improve the performances of LEAP^{MR}, we did a preliminary test where we considered the Moral KB which qualitatively simulates moral reasoning. We performed the experiments on a cluster of Linux machines.For each experiment 2 candidate probabilistic axioms are generated by using CELOE and a maximum of 3 explanations per query was set for EDGE^{MR}. The obtained speedup is significant even if it is sublinear, showing that a certain amount of overhead (the resources, and thereby the time, spent for the MPI communications) is present.

5 SML-Bench

When a new learning system is under development a lot of time is spent for its evaluation. If you want to compare your new system with existing other ones, you have to learn how to use the other systems (usually a command line interface), write a bunch of scripts, manually write the results, etc. In extreme

cases (especially if do not have any datasets) the setup of the experiment session could be more time consuming than the actual development of your new algorithm/system.

The AKSW research group of Leipzig is currently developing SML-Bench[4], a benchmark tool to ease the testing and the comparison of learning systems for structured data. Unfortunately SML-Bench currently supports only non-probabilistic algorithms and provides measures for evaluation that are not suitable for probabilistic learners. AUCROC ("Area Under the Receiver Operating Characteristic curve") and AUCPR ("Area Under the Precision Recall curve") are measures widely used for the evaluation of *probabilistic* and *scoring* classifiers. We extended SML-Bench to use this kind of measures. Moreover, we added the probabilistic structure learning systems SLIPCOVER and LEAP in this benchmark. We are currently evaluating LEAP by means of this benchmarking tool.

6 Conclusions and Future Work

LEAP has been integrated into DL-Learner [8] and it is part of the release 1.3. In such a manner, we extended this framework to the field of Probabilistic Description Logics. As next step in the immediate future we plan to integrate LEAP$^{\mathrm{MR}}$ into DL-Learner and to adapt SML-Bench in order to be able to use distributed system as such LEAP$^{\mathrm{MR}}$ and SEMPRE.

The main problem of structure learning algorithms is that they often scale poorly. This is problematic to handle Big Data. Several solutions could be adopted. In order to fit a dataset in main memory, distributed reasoning approaches could be used [1]. Reducing the knowledge base by removing the irrelevant parts is another way to reduce the reasoning time.

For LEAP$^{\mathrm{MR}}$ we are currently working for distributing both the structure and the parameter learning of probabilistic knowledge bases by exploiting EDGE$^{\mathrm{MR}}$ also when building class expressions. We would like to distribute the scoring function used to evaluate the obtained refinements. In this function EDGE$^{\mathrm{MR}}$ takes as input a KB containing only the individuals and the class expression to test. Finally, the class expressions found are sorted according to the LL returned by EDGE$^{\mathrm{MR}}$ and their initial probability are the probability learned during the execution of EDGE$^{\mathrm{MR}}$. Currently LEAP and LEAP$^{\mathrm{MR}}$ support only supervised learning, we plan to add semi-supervised or unsupervised learning. Another branch of research is to adapt LEAP$^{\mathrm{MR}}$ to exploit Apache Spark and to run the queries on GPUs.

References

1. B. Ahmadi, K. Kersting, M. Mladenov, and S. Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Mach. Learn.*, 92(1):91–132, 2013.

[4] https://github.com/AKSW/SML-Bench

2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.
3. E. Bellodi, E. Lamma, F. Riguzzi, and S. Albani. A distribution semantics for probabilistic ontologies. In *URSW-2011*, volume 778, pages 75–86, Aachen, Germany, 2011. Sun SITE Central Europe.
4. E. Bellodi and F. Riguzzi. Expectation Maximization over Binary Decision Diagrams for probabilistic logic programs. *Intell. Data Anal.*, 17(2):343–363, 2013.
5. E. Bellodi and F. Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theor. Pract. Log. Prog.*, 15(2):169–212, 2015.
6. G. Cota, R. Zese, E. Bellodi, E. Lamma, and F. Riguzzi. Structure learning with distributed parameter learning for probabilistic ontologies. In J. Hollmen and P. Papapetrou, editors, *Doctoral Consortium of ECMLPKDD 2015*, pages 75–84, 2015.
7. G. Cota, R. Zese, E. Bellodi, F. Riguzzi, and E. Lamma. Distributed parameter learning for probabilistic ontologies. In K. Inoue, H. Ohwada, and A. Yamamoto, editors, *25th International Conference on Inductive Logic Programming (ILP 2015)*, 2015.
8. J. Lehmann. DL-Learner: learning concepts in description logics. *J. Mach. Learn. Res.*, 10:2639–2642, 2009.
9. J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(1):71–81, 2011.
10. F. Riguzzi. A top down interpreter for LPAD and CP-logic. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence*, volume 4733 of *LNAI*, pages 109–120. Springer, 2007.
11. F. Riguzzi, E. Bellodi, E. Lamma, and R. Zese. Learning the parameters of probabilistic description logics. In *Inductive Logic Programming Late Breaking papers. CEUR Workshop Proceedings*, volume 1187, pages 46–51. Sun SITE Central Europe, 2014.
12. F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, and G. Cota. Learning probabilistic description logics. In F. Bobillo, R. N. Carvalho, P. C. Costa, C. d'Amato, N. Fanizzi, K. B. Laskey, K. J. Laskey, T. Lukasiewicz, M. Nickles, and M. Pool, editors, *Uncertainty Reasoning for the Semantic Web III*, LNCS, pages 63–78. Springer International Publishing, Berlng, Heidelberg, 2014.
13. F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, and G. Cota. Probabilistic logic programming on the web. *Software Pract. and Exper.*, 2015.
14. F. Riguzzi, E. Bellodi, R. Zese, G. Cota, and E. Lamma. Structure learning of probabilistic logic programs by MapReduce. In K. Inoue, H. Ohwada, and A. Yamamoto, editors, *25th International Conference on Inductive Logic Programming (ILP 2015)*, 2015.
15. T. Sato. A statistical learning method for logic programs with distribution semantics. In L. Sterling, editor, *ICLP-95*, pages 715–729, Cambridge, Massachusetts, 1995. MIT Press.
16. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *20th International Conference on Logic Programming*, volume 3131 of *LNCS*, pages 195–209, Berlin Heidelberg, Germany, 2004. Springer.