Hypergraph Lifting for Structure Learning in Markov Logic Networks

Stanley Kok KOKS@CS.WASHINGTON.EDU
Pedro Domingos PEDROD@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195, USA

1. Introduction

In recent years, there has been a surge of interest in combining statistical and relational learning approaches (Getoor & Taskar, 2007), driven by the realization that many applications require both. Recently, Richardson and Domingos (2006) introduced Markov logic networks (MLNs), a statistical relational language combining first-order logic (Genesereth & Nilsson, 1987) and Markov networks (Pearl, 1988). An MLN consists of weighted first-order logic formulas, viewed as templates for Markov network features. Learning MLN structure is an important but challenging task, and to date only a few approaches have been proposed (Kok & Domingos, 2005; Mihalkova & Mooney, 2007; Biba et al., 2008b; etc.).

Most of these approaches systematically enumerate candidate clauses by starting from an empty clause, greedily adding literals to it, and testing the resulting clause's empirical fit to training data. Such a strategy has two shortcomings: searching the large space of clauses is computationally expensive; and it is susceptible to converging to a local optimum, missing potentially useful clauses. These shortcomings can be ameliorated by using the data to *a priori* constrain the space of candidates. This is the basic idea in *relational pathfinding* (Richards & Mooney, 1992)), which finds paths of true ground atoms that are linked via their arguments and then generalizes them into first-order rules. Each path corresponds to a conjunction that is true at least once in the data. Since most conjunctions are false, this helps to concentrate the search on regions with promising rules. However, pathfinding potentially amounts to exhaustive search over an exponential number of paths. Hence, systems using relational pathfinding (e.g., BUSL (Mihalkova & Mooney, 2007)) typically restrict themselves to very short paths, creating short clauses from them and greedily joining them into longer ones.

In this paper, we present LHL (Kok & Domingos, 2009), an approach that uses relational pathfinding to a fuller extent than previous ones. It mitigates the exponential search problem by first inducing a more compact representation of data, in the form of a hypergraph over clusters of constants. Pathfinding on this 'lifted' hypergraph is typically at least an order of magnitude faster than on the ground training data, and produces MLNs that are more accurate than previous state-of-the-art approaches. LHL is short for Learning via Hypergraph Lifting.

2. Learning via Hypergraph Lifting

In LHL, we make use of *hypergraphs*. A hypergraph is a generalization of a graph in which an edge can link any number of nodes, rather than just two. More formally, we define a hypergraph as a pair (V, E) where V is a set of nodes, and E is a multiset of labeled non-empty ordered subsets of V called hyperedges. In LHL, we find *paths* in a hypergraph. A path is defined as a set of hyperedges such that for any two hyperedges e_0 and e_n in the set, there exists an ordering of (a subset of) hyperedges in the set $e_0, e_1, \dots, e_{n-1}, e_n$ such that e_i and e_{i+1} share at least one node.

A database can be viewed as a hypergraph with constants as nodes, and true ground atoms as hyperedges. Each hyperedge is labeled with a predicate symbol. Nodes (constants) are linked by a hyperedge (true ground atom) if and only if they appear as arguments in the hyperedge. (Henceforth we use *node* and *constant* interchangeably, and likewise for *hyperedge* and *true ground atom.*) A path of hyperedges can be generalized into a first-order clause by variabilizing their arguments. To avoid tracing the exponential number of paths in the hypergraph, LHL first jointly clusters the nodes into higher-level concepts, and by doing so it also clusters the hyperedges (i.e., the ground atoms containing the clustered nodes). The 'lifted' hypergraph has fewer nodes and hyperedges, and therefore fewer paths, reducing the cost of finding them.

LHL begins by lifting a hypergraph. Then it finds paths in the lifted hypergraph. Finally it creates candidate clauses from the paths, and learn their weights to create an MLN.

Presented at *ILP-MLG-SRL*, Leuven, Belgium, 2009.

2.1. Hypergraph Lifting

We call our hypergraph lifting approach LiftGraph. It is defined using Markov logic rules similar to those in SNE (Kok & Domingos, 2008). LiftGraph differs from SNE in the following ways. LiftGraph can handle relations of arbitrary arity whereas SNE can only handle binary relations. While SNE can cluster relation symbols, in this paper, for simplicity, we do not cluster relations. (However, it is straightforward to extend LiftGraph to do so.)

LiftGraph works by jointly clustering the constants in a hypergraph in a bottom-up agglomerative manner, allowing information to propagate from one cluster to another as they are formed. The number of clusters need not be pre-specified. As a consequence of clustering the constants, the ground atoms in which the constants appear are also clustered. Each hyperedge in the lifted hypergraph contains at least one true ground atom.

LiftGraph simplifies the learning problem by performing hard assignments of constant symbols to clusters (i.e., instead of computing probabilities of cluster membership, a symbol is simply assigned to its most likely cluster). The weights and the log-posterior can now be computed in closed form. LiftGraph thus simply searches over cluster assignments, evaluating each one by its posterior probability.

2.2. Path Finding

FindPaths constructs paths by starting from each hyperedge in a hypergraph. It begins by adding a hyperedge to an empty path, and then recursively adds hyperedges linked to nodes already present in the path (hyperedges already in the path are not re-added). Its search terminates when the path reaches a maximum length, when no new hyperedge can be added, or when a resource bound is reached. Each time a hyperedge is added to the path, FindPaths stores the resulting path as a new one. All the paths are passed on to the next step to create clauses.

2.3. Clause Creation and Pruning

A path in the hypergraph corresponds to a conjunction of hyperedges, and it guarantees that the conjunction has at least one support in the hypergraph. We replace each cluster in a path with a variable, thereby creating a variabilized atom for each hyperedge. We convert the conjunction of positive literals to a clause because that is the form that is typically used by ILP and MLN structure learning and inference algorithms. (In Markov logic, a conjunction of positive literals with weight w is equivalent to a clause of negative literals with weight $-w$). In addition, we add clauses with the signs of up to n literals flipped (where n is a user-defined parameter), since the resulting clauses may also be useful. (Notice that if all but one of the literals are negative, this is a definite clause whose antecedent is supported by a path in the hypergraph.)

Table 1. Information on datasets.

Dataset	Types	Constants	Predicates	True Atoms	Total Atoms
IMDB	4	316	6	1224	17,793
UW-CSE	9	929	12	2112	260,254
Cora	5	3079	10	42,558	687,422

We score each clause using weighted pseudo-log-likelihood (WPLL) (Kok & Domingos, 2005), and penalize the WPLL with a length penalty $-\pi d$, where d is the number of atoms in a clause. We iterate over the clauses from shortest to longest. For each clause, we compare its scores against those of its sub-clauses (considered separately) that have already been retained. If the clause scores higher than all of these sub-clauses, it is retained; otherwise, it is discarded. In this manner, we discard clauses which are unlikely to be useful. Note that this process is efficient because the score of a clause only needs to be computed once, and can be cached for future comparisons. Finally we greedily add the clauses one at a time in order of decreasing score to an MLN (initially empty). After adding each clause, we relearn the weights, and keep the clause in the MLN if it improves the overall WPLL. Optionally, we discard clauses containing 'dangling' variables (i.e., variables which only appear once in a clause), since these are unlikely to be useful.

3. Experiments

We carried out experiments to investigate whether LHL performs better than previous approaches, and to evaluate the contributions of its components. We used three datasets publicly available at http://alchemy.cs.washington.edu. Their details are shown in Table 1. The IMDB dataset, created by Mihalkova and Mooney (2007) from the IMDB.com database, describes a movie domain. The UW-CSE dataset, prepared by Richardson and Domingos (2006), describes an academic department. The Cora dataset is a collection of citations to computer science papers, created by Andrew McCallum, and later processed by Singla and Domingos (2006). Each dataset is divided into 5 folds.

We compared LHL to two state-of-the-art systems: BUSL (Mihalkova & Mooney, 2007) and MSL (Kok & Domingos, 2005). Both systems are implemented in the Alchemy software package (Kok et al., 2009). To investigate the importance of hypergraph lifting, we removed the LiftGraph component from LHL, and let

Table 2. Experimental results.

	IMDB			UW-CSE			Cora		
System	AUC	CLL	Time (min)	AUC	CLL	Time (hr)	AUC	CLL	Time (hr)
LHL	0.69±0.01	−0.13±0.00	15.63±1.88	0.22±0.01	−0.04±0.00	7.55±1.53	0.87±0.00	−0.26±0.00	14.82±1.78
LHL-FindPaths	0.69±0.01	−0.13±0.00	242.41±30.31	0.19±0.01	−0.04±0.00	56.69±19.98	0.91±0.00	−0.17±0.00	5935.50±39.21
LHL-LiftGraph	0.45±0.01	−0.27±0.01	0.18±0.01	0.14±0.01	−0.06±0.00	0.001±0.000	-	-	0.01±0.01
BUSL	0.47±0.01	−0.14±0.00	4.69±1.02	0.21±0.01	−0.05±0.00	12.97±9.80	0.17±0.00	−0.37±0.00	18.65±9.52
MSL	0.41±0.01	−0.17±0.00	0.17±0.10	0.18±0.01	−0.57±0.00	2.13±0.38	0.17±0.00	−0.37±0.00	65.60±1.82

FindPaths run on the unlifted hypergraph. We call this system LHL-FindPaths. We also investigated the contribution of hypergraph lifting alone by applying LiftGraph's MLN on the test sets. We call this system LHL-LiftGraph. Altogether we compared five systems: LHL, LHL-FindPaths, LHL-LiftGraph, BUSL and MSL. All systems are implemented in C++.

For each dataset, we performed cross-validation using the five previously defined folds. For IMDB and UW-CSE, we performed inference over the groundings of each predicate to compute their probabilities of being true, using the groundings of all other predicates as evidence. Exceptions are the predicates `Actor` and `Director` (IMDB), and `Student` and `Professor` (UW-CSE). We evaluated groundings for those predicates together, using all other predicates as evidence. This is because groundings of those predicates for the same constant are mutually exclusive and exhaustive (e.g., `Actor(Bob)` and `Director(Bob)`). Knowing one determines the value of the other. For Cora, we ran inference over each of the four predicates `SameCitation`, `SameTitle`, `SameAuthor`, and `SameVenue` in turn, using the groundings of all other predicates as evidence. To evaluate the performance of the systems, we measured the average conditional log-likelihood of the test atoms (CLL), and the area under the precision-recall curve (AUC). Table 2 reports the AUCs, CLLs and runtimes (with their standard deviations) of the various systems. The AUC and CLL results are averages over all atoms in the test sets. Runtimes are averages over the five folds.

We first compare LHL to BUSL and MSL.[1] In both AUC and CLL, LHL outperforms BUSL and MSL on all datasets. The differences between LHL and BUSL on UW-CSE are statistically significant according to one-tailed paired t-tests (p-values ≤ 0.01 for both AUC and CLL). LHL is slower than BUSL and MSL on the smallest dataset (IMDB), mixed on the medium one (UW-CSE), and faster on the largest one (Cora). This suggests that LHL scales better than BUSL and MSL. Next we compare LHL to its components LHL-LiftGraph and LHL-FindPaths. Comparing the runtimes of LHL and LHL-FindPaths, we see that LHL is much faster than LHL-FindPaths. LHL's AUC and CLL are similar to or better than LHL-FindPaths's on IMDB and UW-CSE, but are worse on Cora. These results suggest that: LHL is a lot faster than LHL-FindPaths without any loss in accuracy on some datasets; and when LHL-FindPaths does better, it does so at a huge computational cost (e.g., it took about 247 days to run on Cora[2]). LHL also outperforms LHL-LiftGraph on both AUC and CLL on the IMDB and UW-CSE datasets.[3] This suggests that LHL's ability to learn clauses that capture complex dependencies among predicates is an advantage over the simple rules in LHL-LiftGraphs.

Acknowledgments: This research was partly funded by ARO grant W911NF-08-1-0242, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881 and IIS-0803481, and ONR grant N00014-08-1-0670. The views and conclusions contained in this document are the authors' and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, DARPA, NSF, ONR, or the US Government.

References

Biba, M., Ferilli, S., & Esposito, F. (2008). Structure learning of Markov logic networks through iterated local search. *Proc. ECAI'08*.

Genesereth, M. R., & Nilsson, N. J. (1987). *Logical foundations of artificial intelligence*.

Getoor, L., & Taskar, B. (Eds.). (2007). *Introduction to statistical relational learning*.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. *Proc. ICML'05*.

Kok, S., & Domingos, P. (2008). Extracting semantic networks from text via relational clustering. *Proc. ECML'08*.

Kok, S., & Domingos, P. (2009). Learning Markov logic network structure via hypergraph lifting. *Proc. ICML'09*.

Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., & Domingos, P. (2009). *The Alchemy system for statistical relational AI* (Technical Report). Dept. of Comp. Sci. and Eng., Univ. of Washington. http://alchemy.cs.washington.edu.

Mihalkova, L., & Mooney, R. J. (2007). Bottom-up learning of Markov logic network structure. *ICML'07*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*.

Richards, B. L., & Mooney, R. J. (1992). Learning relations by pathfinding. *Proc. AAAI'92*.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning, 62*.

Singla, P., & Domingos, P. (2006). Entity resolution with Markov logic. *Proc. ICDM'06*.

[1]The results for MSL on UW-CSE and Cora are much worse than those reported by Kok and Domingos (2005). They evaluated MSL by computing the probability that a ground atom is true given all other ground atoms as evidence, a much easier task than ours. We also did not use their domain-specific declarative bias to guide clause construction. The results for BUSL on IMDB and UW-CSE are also worse than that reported by Mihalkova and Mooney (2007). Unlike them, we omitted equality predicates (e.g., `SameMovie(movie, movie)`) because they are superfluous and can be easily predicted with a single unit clause. We also infer the groundings of `Actor/Director` and `Professor/Student` simultaneously, which is a harder task than theirs. The last two reasons also contribute to the poor performance of MSL.

[2]For each test fold, we ran FindPaths in parallel on all training folds, and added the runtimes.

[3]LHL-LiftGraph on Cora crashed by running out of memory. Alchemy automatically converts rules into clausal form and represents each clause separately, causing a blow-up in the number of clauses.