Learning Tractable Statistical Relational Models

Aniruddh Nath NATH@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195

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

Abstract

Sum-product networks (SPNs; Poon & Domingos, 2011) are a recently-proposed deep architecture that guarantees tractable inference, even on certain high-treewidth models. SPNs are a propositional architecture, treating the instances as independent and identically distributed. In this paper, we introduce *Relational Sum-Product Networks* (RSPNs), a new tractable first-order probabilistic architecture. RSPNs generalize SPNs by modeling a set of instances jointly, allowing them to influence each other's probability distributions, as well as modeling the probabilities of relations between objects. We also present LearnRSPN, the first algorithm for learning tractable statistical relational models. LearnRSPN is a recursive top-down structure learning algorithm for RSPNs, similar to the LearnSPN (Gens & Domingos, 2013) algorithm for propositional SPN learning. In preliminary experiments, RSPNs outperform Markov Logic Networks (Richardson & Domingos, 2006) in both running time and predictive accuracy.

1. Introduction

Graphical probabilistic models compactly represent a joint probability distribution among a set of variables. Unfortunately, inference in graphical models is intractable. In practice, using graphical models for most real-world applications requires either using approximate algorithms, or restricting oneself to a subset of graphical models on which inference is tractable. A common restriction that ensures tractability is to use models with low treewidth (e.g. Bach & Jordan, 2001; Chechetka & Guestrin, 2007). However, not all tractable models have low treewidth. Poon & Domingos (2011) recently proposed *Sum-Product Networks (SPNs)*, a tractable probabilistic architecture that guarantees efficient exact inference. SPNs subsume most known tractable probabilistic models, and can compactly represent some high-treewidth distributions. SPNs can be seen as a deep architecture with alternating layers of sum nodes and product nodes. Since their introduction, several SPN learning algorithms have been proposed, and SPNs have been applied to a variety of problems (Delalleau & Bengio, 2011; Gens & Domingos, 2012; Dennis & Ventura, 2012; Amer & Todorovic, 2012; Peharz et al., 2013; Gens & Domingos, 2013).

Besides intractability, one other key restriction of most widely used graphical models is their reliance on the i.i.d. assumption. In many real-world applications, training instances are not truly independent, and can be better modeled if their interactions are taken into account. This is one of the key insights of the Statistical Relational Learning (SRL) community (Getoor & Taskar, 2007). In recent years, SRL techniques have been applied to a wide variety of tasks, including collective classification, link prediction, vision, natural language processing, etc. However, most SRL methods are built on top of graphical models (e.g. Markov logic networks; Richardson & Domingos, 2006), and suffer from the same computational difficulties; these are compounded by the additional problem of modeling interactions between instances.

In this paper, our goal is to combine these two lines of research: tractable probabilistic models (in this case, SPNs) and relational learning. One piece of related research is Flach & Lachiche's (2004) work on Naïve Bayes classification in structured domains. Their work was limited to classification, and was very limited in expressiveness (being a Naïve Bayes model). Koller et al.'s P-CLASSIC combines description logic with Bayesian networks, but is only tractable when the Bayesian networks are restricted to polytrees. The first non-trivially tractable probabilistic relational representation was Tractable Markov Logic (TML; Domingos & Webb, 2012), a subset of Markov logic

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32. Copyright 2014 by the author(s).

on which efficient inference could be guaranteed. TML is surprisingly expressive, subsuming most previous tractable models. However, a TML knowledge base determines the set of possible objects in the domain, and the relational structure among them. This limits the applicability of TML to learning; a TML knowledge base cannot be learned on one set of objects and applied to another mega-example with different size or structure.

To address this, we propose *Relational Sum-Product Networks* (RSPNs), a new tractable relational probabilistic architecture. RSPNs generalize SPNs by modeling a set of instances jointly, allowing them to influence each other's probability distributions, as well as modeling the probabilities of relations between objects. We also introduce *LearnRSPN*, the first algorithm for learning tractable statistical relational models. Intractable inference has historically been a major obstacle to the wider adoption of statistical relational methods; the development of learning and inference algorithms for tractable relational models could go a long way towards making SRL more widely applicable.

2. Background

2.1. Sum-Product Networks

A sum-product network (SPN) is a rooted directed acyclic graph with univariate distributions at the leaves; the internal nodes are (weighted) sums and (unweighted) products. In this paper, we use the simplified definition of Gens & Domingos (2013):

Definition 1. A sum-product network (SPN) is defined as follows.

1. A tractable univariate distribution is an SPN.
2. A product of SPNs with disjoint scopes[1] is an SPN.
3. A weighted sum of SPNs with the same scope is an SPN, provided all weights are positive.
4. Nothing else is an SPN.

A univariate distribution is tractable iff its partition function and mode can be computed in $O(1)$ time.

Intuitively, an SPN can be thought of as an alternating set of mixtures (sums) and decompositions (products) of the leaf variables (Fig. 1). If the values at the leaf nodes are set to the partition functions of the corresponding univariate distribution, then the value at the root is the partition function (i.e. the sum of the unnormalized probabilities of all possible assignments to the leaf variables). This allows the partition function to be computed in time linear in the size of the SPN.

Similarly, if some of the leaves are known, setting their values to their probabilities (according to the corresponding univariates) yields the unnormalized probability of the evidence. This can be divided by the partition function to obtain the normalized probability. MPE inference is also linear in the size of the SPN, replacing sum nodes by max nodes, which corresponds to viewing these nodes as hidden variables that we also maximize over.

2.2. Learning SPNs

The first learning algorithms for sum-product networks used a fixed network structure, and only optimized the weights (Poon & Domingos, 2011; Amer & Todorovic, 2012; Gens & Domingos, 2012). The network structure is domain-dependent; applying SPNs to a new problem required manually designing a suitable network structure.

More recently, several algorithms have been proposed that learn both the weights and the network structure, allowing SPNs to be applied out-of-the-box to new domains. Dennis & Ventura (2012) suggested an algorithm that builds an SPN based on a hierarchical clustering of variables. Gens & Domingos (2013) construct SPNs top-down, recursively partitioning instances and variables. Peharz et al. (2013) construct SPNs bottom-up, greedily merging SPNs into models of larger scope. These algorithms have been shown to perform well on a variety of domains, making more accurate predictions than conventional graphical models, while guaranteeing tractable inference.

2.3. Statistical Relational Learning

SPNs are a propositional representation, modeling instances as independent and identically distributed (i.i.d.). Although the i.i.d. assumption is widely used in statistical machine learning, it is often an unrealistic assumption. In practice, objects usually interact with each other; Statistical Relational Learning algorithms (Getoor & Taskar, 2007) can capture dependencies between objects, and make predictions about relationships between them.

Markov Logic Networks (MLNs; Richardson & Domingos, 2006) are a widely-used representation used for relational learning. An MLN is a first-order representation of the dependencies between objects in a domain. Given a *mega-example* (a set of related objects), an MLN can be *grounded* into a propositional graphical model representing a joint probability distribution over the attributes and relations among those objects. Unfortunately, the resulting graphical model is typically high treewidth, and inference is intractable. In practice, users of SRL methods typically resort to approximate inference algorithms based on

[1] The scope of an SPN is the set of variables that appear in it.

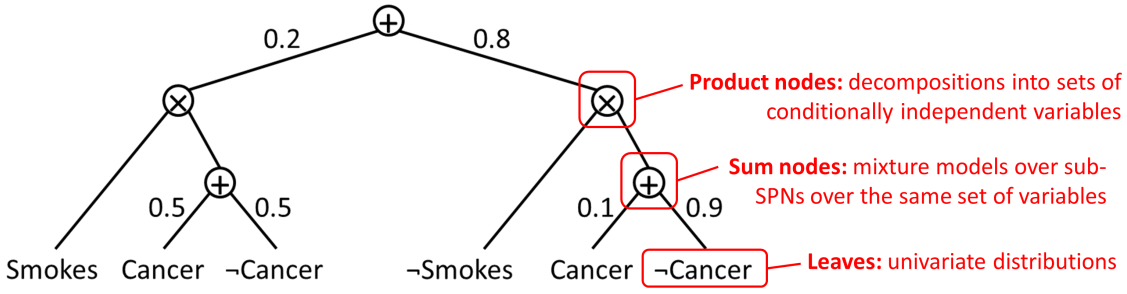

Figure 1. Example sum-product network. $Smokes$ and $Cancer$ are Bernoulli distributions with probability 1; $\neg Smokes$ and $\neg Cancer$ are distributions over the same two variables, with probability 0.

MCMC or loopy belief propagation, resulting in long runtimes and unreliable predictions. Like propositional graphical models, statistical relational models can be trivially restricted to the low-treewidth case, but this comes at great cost to the representational power of the model.

Tractable Markov Logic (TML; Domingos & Webb, 2012) is a subset of Markov Logic that guarantees polynomial-time inference, even in certain cases where the ground propositional model would be high in treewidth. TML is expressive enough to capture several cases of interest, including junction trees, non-recursive probabilistic context-free grammars, hierarchical mixture models etc. A TML knowledge base is a generative model that decomposes the domain into parts, with each part drawn probabilistically from a class hierarchy. Each part is further probabilistically decomposed into subparts (according to its class). Inference in TML is carried out using probabilistic theorem proving (Gogate & Domingos, 2011). The key limitation of TML is that the knowledge base fully specifies the set of possible objects in the domain, and their class and part structure. A TML knowledge base cannot be learned from one mega-example and used to model another example with different size or structure.

This paper uses some first-order logic terminology. For our purposes, '*predicate*' refers to a first-order logic predicate. (Our representation and algorithm support numeric attributes and relations as well, but for simplicity we focus on the Boolean case.) A *grounding* of a predicate (or a *ground atom*) is a replacement of all its variables by constants.

3. Relational Sum-Product Networks

3.1. Exchangeable Distribution Templates

Before we define RSPNs, we first define the notion of an *Exchangeable Distribution Template* (EDT).

Definition 2. Consider a finite set of variables $\{X_1, \ldots, X_n\}$ with joint probability distribution P. Let $S(n)$ be the set of all permutations on $\{1, \ldots, n\}$. $\{X_1, \ldots, X_n\}$ is a *finite exchangeable set* with respect to P if and only if $P(X_1, \ldots, X_n) = P(X_{\pi(1)}, \ldots, X_{\pi(n)})$ for all $\pi \in S(n)$. (Diaconis & Freedman, 1980).

Note that finite exchangeability does not require independence: a set of variables can be exchangeable despite having strong dependencies. (For example, consider binary variables $X_1, \ldots, X_n$, with a uniform probability distribution over value assignments with an even number of non-zero variables.)

Definition 3. An *Exchangeable Distribution Template* (EDT) is a function that takes a set of variables $\{X_1, \ldots, X_n\}$ as input (n is unknown a priori), and returns a joint probability distribution P with respect to which $\{X_1, \ldots, X_n\}$ are exchangeable. We refer to the probability distribution P returned by the EDT for a given set of variables as an *instantiation* of that EDT.

Example 1. The simplest family of EDTs simply returns a product of univariate distributions over each of $X_1, \ldots, X_n$. For example, if the variables are binary, then an EDT might model them as a product of Bernoulli distributions with some probability p.

Example 2. Consider an EDT over a set of binary variables $X_1, \ldots, X_n$ (with n unknown a priori), returning the following distribution:

$$P(X_1, \ldots, X_n) \propto \frac{\lambda^k}{k!} e^{-\lambda} \quad \text{where} \quad k = \sum_{X_i} \mathbf{1}[X_i]$$

λ is a parameter of the EDT. This is an EDT with a Poisson distribution over the number of variables in the set with value 1. (The probabilities must be renormalized, since the set of variables is finite.) Note that this EDT does not assume independence among variables.

Intuitively, EDTs can be thought of as probability distributions that depend only on aggregate statistics, and not on the values of individual variables in the set.

3.2. Relational Sum-Product Networks

Relational Sum-Product Networks (RSPNs) jointly model the attributes and relations among a set of objects. RSPNs inherit TML's notion of parts and classes. Unlike TML, an RSPN class's parts may be *unique* or *exchangeable*. An object's unique parts are those that play a special role, e.g. the commander of a platoon, the queen of a bee colony, or the hub of a social network. The exchangeable parts are those that behave interchangeably: soldiers in a platoon, worker bees in a colony, spokes in a network, and so on.

Definition 4. A *class definition* for RSPN class C consists of:

- A set of *attributes*: unary predicates of the form $A(x)$, where $x \in C$.
- A set of *unique parts*, each of which is associated with a class.
- A set of *exchangeable parts*, each of which is associated with a class.
- A set of *relations* between subparts: predicates of the form $R_1(P_1, P_2)$ or $R_2(C, P_1)$, where P_1 and P_2 are either unique or exchangeable parts of C. (Predicates may be of any arity.)
- A *class SPN* whose leaves are in one of three sets:
  - $U_A^{(C)}$, univariate distributions over unary predicates involving C (i.e. attributes of the form $A(C)$);
  - $U_R^{(C)}$, EDTs over groundings of binary (or higher-order) predicates involving C and/or its subpart types (e.g. relations of the form $R_1(P_1, P_2)$ or $R_2(C, P_1)$, where P_1 and P_2 are parts of C);
  - $U_P^{(C)}$, sub-SPNs for subparts.

Example 3. The following is a partial class specification for a simple political domain. A 'Region' consists of an arbitrary number of nations, and relationships between nations are modeled at this level. A 'Nation' has a unique government and an arbitrary number of people. National properties such as 'High GDP' are modeled here. The 'Supports' relation can capture a distribution over the number of people in the nation who support the government.

```
class Region:
  exchangeable part Nation
  relation Adjacent(Nation,Nation)
  relation Conflict(Nation,Nation)

class Nation:
  unique part Government
  exchangeable part Person
  attribute HighGDP
  relation Supports(Person,Government)

class Government:
  attribute Democratic

class Person:
  attribute AbovePovertyLine
  attribute Employed
```

See Fig. 2 for an example class SPN.

3.3. Grounding an RSPN

Like MLNs, RSPNs are templates for propositional models. To generate a ground SPN from an RSPN, we take as input a *part decomposition*, which is a tree of typed objects. A part decomposition D is consistent with RSPN class C if and only if:

- The root object of D is of class C;
- The root object has one child o_u for each unique part C_u of C, and the subtree rooted at o_u is consistent with C_u;
- Each remaining child object o_e is of a type matching some exchangeable part C_e of C, and the subtree rooted at o_e is consistent with C_e.

To *ground* a class SPN is to instantiate the template for a specific set of objects. Given a class C and a part decomposition D (rooted at object o), grounding C's SPN yields a propositional SPN whose leaf distributions are over attributes and relations involving the objects in D. This is done recursively as follows:

- For leaves in $U_A^{(C)}$: replace the univariate distribution over predicate $A(C)$ in the class SPN with a univariate distribution over $A(o)$ in the ground SPN.
- For leaves in $U_R^{(C)}$: replace the EDT over $R(X, Y)$ in the class SPN with an instantiation of that EDT over the groundings of R.
- For leaves in $U_P^{(C)}$: recursively ground P's class SPN over each type-P child of object o. Replace the sub-SPN in C's SPN with a product over groundings of P.

See Fig. 3 for an example.

3.4. Representational Power

The main limitation of the RSPN representation is that individual relational atoms are not modeled directly, but

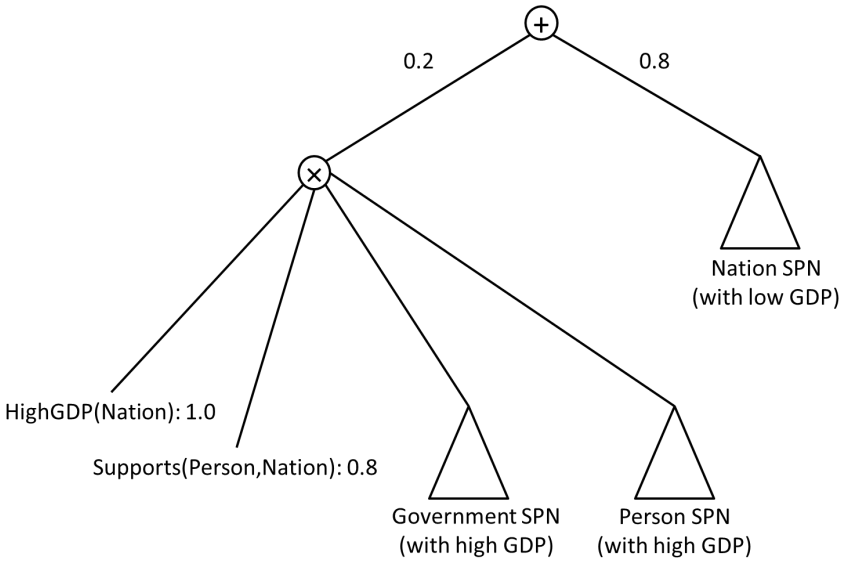

Figure 2. Partial SPN for the 'Nation' class (example 3), modeling all a nation's parts as independent conditioned on the 'HighGDP' attribute.

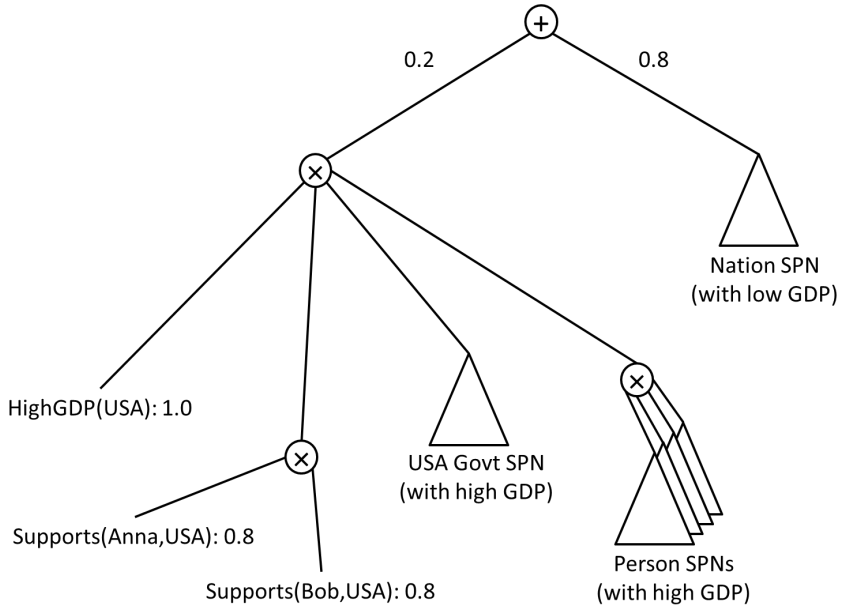

Figure 3. Example grounding of the 'Nation' class, with the class SPN from Fig. 2.

through aggregations. In this respect, it is similar to Probabilistic Relational Models (PRMs, Friedman et al., 1999)—though RSPNs guarantee tractable inference, unlike PRMs. Aggregations are extremely useful for capturing relational dependencies (Natarajan et al., 2012; Nath & Richardson, 2012). For example, a person's smoking habits may depend on the number of friends she has who smoke, and not the smoking habits of each individual friend. Nevertheless, aggregations are not well-suited to capturing relational patterns that depend on specific paths of influence, such as Ising models.

It is important to note than RSPNs (and SPNs) are not simply tree-structured graphical models. The graph of an RSPN is not a conditional dependency graph, but a graphical representation of the computation of the partition function. A ground RSPN can be converted into an equivalent graphical model, but the resulting model may be high-treewidth, and computationally intractable.

In effect, RSPNs are a way to compactly represent context-specific independences in relational domains: different children of a sum node may have different variable decompositions in their product nodes. These context-specific independences are what give RSPNs more expressiveness than low-treewidth tractable models.

4. Learning RSPNs

The learning task for RSPNs is to determine the structure and parameters of all the class SPNs in the domain. In this work, we assume that the part relationships among classes are known, i.e. the user determines what types of unique and exchangeable parts are allowed for each class. The input for the learning algorithm is a set of part decompositions (trees of objects), and an evidence database specifying the values of the attributes and relations among those objects.

Our learning algorithm is based on the top-down LearnSPN algorithm (Gens & Domingos, 2013). LearnSPN(T, V) is propositional SPN learning algorithm, and takes as input a set of training instances T and variables V. The algorithm attempts to decompose V into independent subsets $V_1, \ldots, V_k$ (using pairwise statistical independence tests); if such a decomposition exists, LearnSPN recurses over each set (calling LearnSPN(T, V_1),. . .,LearnSPN(T, V_k)), and returns a product node over the recursively learned sub-SPNs. If V does not decompose into independent subsets, LearnSPN instead clusters the instances T, recursively learns a sub-SPN over each subset $T_1, \ldots, T_k$, and returns a sum-node over the sub-SPNs weighted by the mixture proportions. Under certain assumptions, LearnSPN can be seen as a greedy search maximizing the likelihood of the learned SPN.

Algorithm 1 LearnRSPN(C, T, V)

input: C, a class
T, a set of instances of C
V, a set of class attributes, relation aggregates, and subparts
output: an RSPN for class C
if $|V| = 1$ **then**
  if $v \in V$ is an attribute **then**
    return univariate estimated from v's values in T
  else if v is a relation aggregate **then**
    return EDT estimated from v's values in T
  else
    //v is a subpart
    $C_{child} \leftarrow$ class of v
    $T_{child} \leftarrow$ subparts of $t \in T$ of type C_{child}
    $V_{child} \leftarrow$ attributes, relations and subparts of C_{child}
    return LearnRSPN$(C_{child}, T_{child}, V_{child})$
  end if
else
  partition V into approximately independent subsets V_j
  if success **then**
    return $\prod_j$ LearnRSPN(C, T, V_j)
  else
    partition T into sets of similar subsets T_i
    return $\sum_i \frac{|T_i|}{|T|}$.LearnRSPN(C, T_i, V)
  end if
end if

Given an RSPN class C, LearnSPN could be used directly to learn an SPN over C's attributes. However, C's exchangeable parts pose a problem for LearnSPN: the number of leaf variables in the ground SPN can differ from one training instance to another, and between training instances and test instances. To address this, we propose the LearnRSPN algorithm (Algorithm 1), a relational extension of LearnSPN. (For the purpose of this discussion, subparts are assumed to be exchangeable; attributes of unique subparts can be handled the same way as attributes of the parent part, and represented as separate leaves in the class SPN.)

LearnRSPN is a top-down algorithm similar to LearnSPN; it attempts to find independent subsets from among the object's set of attributes, relations and subparts; if multiple subsets exist, the algorithm learns a sub-SPN over each subset, and returns a product over the sub-SPNs. If independent subsets cannot be found, LearnRSPN instead clusters the instances, returning a sum node over the components, weighted by the mixture proportions.

LearnRSPN exploits the fact that predicates involving exchangeable parts are grounded into finite sets of exchangeable variables. Instead of treating each ground atom as a separate leaf in the SPN, LearnRSPN summarizes a set of

exchangeable variables with an aggregate statistic (in our experiments, we used the fraction of true variables in the set, though other statistics can be used). This summary statistic is treated as a single variable in the decomposition stage of RSPN. Thus, attributes that are highly predictive of the statistics of the groundings of a relation will be grouped with that relation. Subparts are similarly summarized by the statistics of their attribute and relation predicates.

Example 4. The following is an instance of the $Nation$ class described in example 3, as summarized by LearnRSPN (using the fraction of true groundings as the aggregation):

```
Nation AnkhMorpork:
  Government.Democratic: False
  Person.AbovePovertyLine: 0.4
  Person.Employed: 0.9
  HighGDP: False
  Supports(Person,Government): 0.1
```

The base case of LearnRSPN (when $|V| = 1$) varies depending on what v is. When v is an attribute, the RSPN to be returned is simply a univariate distribution over the attribute, as in the propositional version of LearnSPN. When v is a subpart of class C_{child}, LearnRSPN returns an SPN for class C_{child}. Crucially, different SPNs are learned for C_{child} in different children of a sum node in parent class C (since the recursive call is made with a different set of instances). The final base case is when v is an aggregate over an exchangeable relation. In this case, the RSPN to be returned is an EDT over the relation.

Like LearnSPN, LearnRSPN can be seen as an algorithm schema rather than a single learning algorithm; the user is free to choose a clustering algorithm for instances, a dependency test for variable splitting, an aggregate statistic, and a family of EDTs for exchangeable relations. The choices used for our experiments are explained in the section 5.1. Note that different families of EDTs may require different aggregate statistics for parameter estimation. The fraction of true groundings is sufficient for the two EDTs described in section 3.1.

5. Evaluation

5.1. Methodology

We compared a Python implementation of LearnRSPN to the MSL learning algorithm (Kok & Domingos, 2005) for Markov Logic Networks, as implemented in the widely-used ALCHEMY system[2] (Kok et al., 2008). We also compare to a baseline (BS) that simply predicts each atom according to the global marginal probability of the predicate being true. TML and propositional SPNs cannot be evaluated in this setting because the training and test examples have different structure.

To cluster instances in LearnRSPN, we used the EM implementation in SCIKIT-LEARN (Pedregosa et al., 2011), with two clusters. To test dependence between variables, we fit a Gaussian distribution (for aggregate variables) or Bernoulli distribution (for binary attributes), and computed the pairwise mutual information (MI) between the variables. The test statistic used was $G = 2N \times MI$, which in the discrete case is equivalent to the G-test used by (Gens & Domingos, 2013). We used a threshold of 0.5 for dependence. For EDTs, we used the independent Bernoulli form, as described in example 1. All Bernoulli distributions were smoothed with a pseudocount of 0.1.

For MLNs, we used the MSL learning algorithm and MC-SAT inference algorithm, the default choices in ALCHEMY 2.0, with the default parameters.

We report results in terms of area under the precision-recall curve (AUC; Davis & Goadrich, 2006) and the average conditional marginal likelihood (CMLL) of the test atoms. AUC is a measure of prediction quality that is insensitive to imbalance in the number of true and false atoms. CMLL directly measures the quality of the probability estimates. For LearnRSPN, we also report the test set likelihood normalized by the number of queries, as an alternate measure of prediction quality. (ALCHEMY does not compute this quantity.)

5.2. Experiment

The UW-CSE database (Richardson & Domingos, 2006) has been used to evaluate a variety of statistical relational learning algorithms. The dataset describes the University of Washington Compute Science & Engineering department, and includes advising relationships, paper authorships etc. The database is divided into five non-overlapping mega-examples, by research area.

To generate a part structure for this domain, we separated the people into one research group per faculty member, with students determined using the $AdvisedBy$ and $TempAdvisedBy$ predicates (breaking ties by number of coauthored papers). Publications are also divided among groups: each paper is assigned to the group of the professor who wrote it, voting by the number of student authors in the group in the event of a tie. The prediction tasks are to infer the roles of faculty (Professor, Associate Professor or Assistant Professor) and students (Pre-Quals, Post-Quals, Post-Generals), as well as paper authorships. The part structure is also made available to ALCHEMY in the form of predicates $Has(Area, Group)$, $Has(Group, Professor)$, $Has(Group, Student)$, $Has\_Group(Group, Paper)$,

[2] ALCHEMY 2.0, http://code.google.com/p/alchemy-2/

Table 1. UW-CSE results. The five areas are AI, Graphics, Languages, Systems and Theory.

Area	$\|Q\|$	Learning time (s)		Inference time (s)		AUC-PR			CMLL			LL/$\|Q\|$
		RSPN	MLN	RSPN	MLN	RSPN	MLN	BS	RSPN	MLN	BS	RSPN
AI	1414	8.85	12,137.74	0.10	5.88	0.73	0.25	0.25	-0.09	-0.19	-0.16	-0.09
Grph.	171	8.60	6,464.82	0.03	0.55	0.63	0.44	0.26	-0.25	-0.31	-0.30	-0.24
Lang.	17	8.46	20,294.36	0.01	0.04	1.00	0.42	0.43	-0.10	-1.35	-0.37	-0.07
Sys.	1120	10.90	9,106.62	0.09	6.99	0.54	0.30	0.13	-0.09	-0.17	-0.14	-0.09
Th.	308	6.93	17,435.36	0.03	1.11	0.75	0.30	0.33	-0.13	-0.37	-0.21	-0.12
Avg.	606	**8.75**	13,087.80	**0.05**	2.91	**0.73**	0.34	0.27	**-0.13**	-0.47	-0.23	-0.12

```
class Area:
  exchangeable part Group

class Group:
  unique part Professor
  exchangeable part Student
  exchangeable part GroupPaper
  exchangeable part NonGroupPaper
  relation Author(Professor, GroupPaper)
  relation Author(Professor, NonGroupPaper)
  relation Author(Student, GroupPaper)
  relation Author(Student, NonGroupPaper)

class Professor:
  attribute Position_Faculty
  attribute Position_Adjunct
  attribute Position_Affiliate

class Student:
  attribute InPhase_PreQuals
  attribute InPhase_PostQuals
  attribute InPhase_PostGenerals
```

Figure 4. Part structure for UW-CSE domain

and $Has\_NonGroup(Group, Paper)$.

We performed leave-one-out testing by area, testing on each area in turn using the model trained from the remaining four. 80% of the groundings of the query predicates were provided as evidence, and the task was to predict the remaining atoms. Table 1 shows the results on all five areas, and the average. LearnRSPN is orders of magnitude faster than ALCHEMY at both training and inference time, and considerably more accurate than either ALCHEMY or the baseline.

6. Discussion

Statistical relational learning algorithms have been successfully applied to several problems, but the difficulty, cost and unreliability of approximate inference has limited their wider adoption. In practice, applying statistical relational methods to a new domain requires substantial engineering effort in choosing and configuring the approximate learning and inference algorithms. The expressiveness of languages like Markov logic is both a boon and a curse: although these languages can compactly represent sophisticated probabilistic models, they also make it easy for practitioners to unintentionally design models too complex even for state-of-the-art inference algorithms.

In the propositional setting, several approaches have been recently proposed for learning high-treewidth tractable models (Lowd & Domingos, 2008; Roth & Samdani, 2009; Gogate et al., 2010; Poon & Domingos, 2011). To our knowledge, LearnRSPN is the first algorithm for learning high-treewidth tractable relational models. (TML is also a high-treewidth relational representation, but is unsuitable for learning, as explained in section 2.3.) In preliminary experiments, LearnRSPN outperforms conventional statistical relational methods in accuracy, inference time and training time.

A limitation of RSPNs is that they require a known, fixed part decomposition for all training and test mega-examples. Applying RSPNs to a new domain does require the user to specify the part decomposition; this is analogous to specifying the class structure in object-oriented programming. Many domains have a natural part structure that can be exploited (like the UW-CSE database); in other cases, part structure can be created using existing graph-cut or community detection algorithms. An important direction for future work is to develop an efficient, principled method of finding part structure in a database.

Acknowledgments

This research was partly funded by ARO grant W911NF-08-1-0242, ONR grants N00014-13-1-0720 and N00014-12-1-0312, and AFRL contract FA8750-13-2-0019. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, ONR, AFRL, or the United States Government.

References

Amer, M. R. and Todorovic, S. Sum-product networks for modeling activities with stochastic structure. In *Proceedings of CVPR*, 2012.

Bach, F. and Jordan, M. I. Thin junction trees. In *Proceedings of NIPS*, 2001.

Chechetka, A. and Guestrin, C. Efficient principled learning of thin junction trees. In *Proceedings of NIPS*, 2007.

Davis, J. and Goadrich, M. The relationship between precision-recall and ROC curves. In *Proceedings of ICML*, 2006.

Delalleau, O. and Bengio, Y. Shallow vs. deep sum-product networks. In *Proceedings of NIPS*, 2011.

Dennis, A. and Ventura, D. Learning the architecture of sum-product networks using clustering on variables. In *Proceedings of NIPS*, 2012.

Diaconis, P. and Freedman, D. De Finetti's generalizations of exchangeability. *In Studies in Inductive Logic and Probability*, 2:235–250, 1980.

Domingos, P. and Webb, A. A tractable first-order probabilistic logic. In *Proceedings of AAAI*, 2012.

Flach, P. and Lachiche, N. Naive Bayesian classification of structured data. *Machine Learning*, 57(3):233–269, 2004.

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. Learning probabilistic relational models. In *Proceedings of IJCAI*, 1999.

Gens, R. and Domingos, P. Discriminative learning of sum-product networks. In *Proceedings of NIPS*, 2012.

Gens, R. and Domingos, P. Learning the structure of sum-product networks. In *Proceedings of ICML*, 2013.

Getoor, L. and Taskar, B. (eds.). *Introduction to Statistical Relational Learning*. MIT Press, 2007.

Gogate, V. and Domingos, P. Probabilistic theorem proving. In *Proceedings of UAI*, 2011.

Gogate, V., Webb, W. A., and Domingos, P. Learning efficient Markov networks. In *Proceedings of NIPS*, 2010.

Kok, S. and Domingos, P. Learning the structure of Markov logic networks. In *Proceedings of ICML*, 2005.

Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., and Domingos, P. The Alchemy system for statistical relational AI. Technical report, University of Washington, 2008. http://alchemy.cs.washington.edu.

Koller, D., Levy, A., and Pfeffer, A. P-Classic: A tractable probabilistic description logic. In *AAAI*, 1997.

Lowd, D. and Domingos, P. Learning arithmetic circuits. In *Proceedings of UAI*, 2008.

Natarajan, S., Khot, T., Kersting, K., Gutmann, B., and Shavlik, J. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1):25–56, 2012.

Nath, A. and Richardson, M. Counting-MLNs: Learning relational structure for decision making. In *Proceedings of AAAI*, 2012.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Peharz, R., Geiger, B. C., and Pernkopf, F. Greedy partwise learning of sum-product networks. In *Proceedings of ECML-PKDD*, 2013.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *Proceedings of UAI*, 2011.

Richardson, M. and Domingos, P. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

Roth, D. and Samdani, R. Learning multi-linear representations of distributions for efficient inference. *Machine Learning*, 76:107–136, 2009.