The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Learning Mixtures of MLNs

Mohammad Maminur Islam
Department of Computer Science
The University of Memphis
mislam3@memphis.edu

Somdeb Sarkhel
Adobe Research
sarkhel@adobe.com

Deepak Venugopal
Department of Computer Science
The University of Memphis
dvngopal@memphis.edu

Abstract

Weight learning is a challenging problem in Markov Logic Networks (MLNs) due to the large size of the ground propositional probabilistic graphical model that underlies the first-order representation of MLNs. Though more sophisticated weight learning methods that use lifted inference have been proposed, such methods can typically scale up only in the absence of evidence, namely in generative weight learning. In discriminative learning, where the evidence typically destroys symmetries, existing approaches are lacking in scalability. In this paper, we propose a novel, intuitive approach for learning MLNs discriminatively by utilizing approximate symmetries. Specifically, we reduce the size of the training database by clustering approximately symmetric atoms together and selecting a representative atom from each cluster. However, each choice made from the clusters induces a different distribution, increasing the uncertainty in our learned model. To reduce this uncertainty, we learn a finite mixture model by stacking the different distributions, where the parameters of the model are learned using an EM approach. Our results on several benchmarks show that our approach is much more scalable and accurate as compared to existing state-of-the-art MLN learning methods.

Introduction

Markov Logic Networks (MLNs) (Domingos and Lowd 2009) are templates that can represent large probabilistic graphical models compactly using first-order logic formulas. Specifically, MLNs represent uncertain knowledge as weighted first-order logic formulas, where the weight signifies uncertainty associated with that formula. In a typical use-case for MLNs, application designers encode domain knowledge manually through first-order logic formulas, and then use automated tools such as Alchemy (Kok et al. 2006) and Tuffy (Niu et al. 2011) to learn the weights of the formulas from data. Unfortunately, both weight learning and inference (which is a sub-step in weight learning) in MLNs are highly challenging problems. Typical learning and inference methods developed for probabilistic graphical models, cannot be directly applied to MLNs. Specifically, MLNs represent extremely large probabilistic graphical models typically containing millions of variables and factors, and running even approximate inference methods or approximate gradient-based methods for learning such as voted perceptron (Collins 2002) or contrastive divergence (Hinton 2002) is often infeasible in MLNs designed for practical applications. To circumvent the non-scalability of learning algorithms, application designers typically use domain-specific 'tricks' to utilize MLNs for practical problems. For example, in their entity resolution application, Singla and Domingos (Singla and Domingos 2006) remove some evidence from the training data that are likely to be false based on heuristics. Similarly, Venugopal et al. (Venugopal et al. 2014) use SVM-learned weights for MLNs. However, such domain-specific heuristics are problematic since they require a deep understanding of the domain, and are typically not transferable across different MLN applications.

Over the last several years, *lifted inference* has become the predominant method to scale up inference, and several algorithms have been developed using this strategy (Poole 2003; Gogate and Domingos 2011; Van den Broeck et al. 2011). The main idea in these algorithms is to use symmetries in the MLN to scale up inference. More recently, lifted inference algorithms have been used in generative weight learning algorithms (Haaren et al. 2016). Unfortunately, the same techniques do not work well for discriminative learning methods. Specifically, when learning discriminatively, we need to perform inference on MLNs that are conditioned on certain pre-specified query atoms (Domingos and Lowd 2009), and most lifted inference methods, even the approximate ones, fail to scale up when the MLN is conditioned on arbitrary evidence that breaks symmetries (van den Broeck and Darwiche 2013). This is highly problematic because discriminative learning is quite often preferred since it converges faster than generative learning (Singla and Domingos 2005), for many applications. Therefore, we need advanced methods to perform discriminative weight learning in MLNs. Recently Sarkhel et al. (Sarkhel et al. 2016) proposed an approach that scales up specific learning algorithms that use Gibbs sampling or MaxWalkSAT as a substep, by utilizing fast methods to solve a counting problem that repeatedly occurs with Gibbs sampling and MaxWalkSAT. However, a generic approach for scalable discriminative learning is still elusive, which is our main contribution in this paper.

Specifically, in this paper, we propose a discriminative learning method which leverages recent advancements in 'approximate' lifted inference techniques. These techniques

utilize approximate symmetries (Venugopal and Gogate 2014; van den Broeck and Darwiche 2013) in the MLN in the absence of exact symmetries, to improve the scalability of inference procedures in the presence of arbitrary evidence. The main idea behind these approaches is to create a smaller, compressed database, where inference can be performed efficiently. Our approach in this paper is to learn the weights from such a compressed database. Specifically, we cluster atoms in the given training database and sample a representative atom from each cluster. However, this approximation increases the uncertainty in our learned model, since, depending on which representative atoms were sampled from a cluster, the underlying distributions could be entirely different. To reduce this uncertainty, we learn multiple models from the same training database and combine them into a mixture model. However, learning such a mixture model turns out to be a complex task since algorithms such as EM are computationally expensive for MLNs (Domingos and Lowd 2009). To scale up the learning procedure, we learn the model based on *stacked density estimation* (Smyth and Wolpert 1999), where we fix the parameters of the mixture components and then use EM to estimate the component coefficients.

Our experiments on several benchmarks taken from Alchemy (Kok et al. 2006) show that our approach is more scalable and accurate than Tuffy (Niu et al. 2011) and other state-of-the-art systems for MLNs.

Background

Markov Logic Networks (MLNs). An issue with first-order logic is that it cannot represent uncertainty: all worlds that violate even one ground formula are considered inconsistent. MLNs soften the constraint expressed by each formula, by attaching a weight to it. Higher the weight, higher the probability of the clause being satisfied, all other things being equal. MLNs can also be seen as a first-order template for generating large Markov networks. Formally, an MLN is a set of pairs (f_i, θ_i) where f_i is a formula in first-order logic and θ_i is a real number. Given a set of constants, an MLN represents a ground Markov network which has one random variable for each grounding of each predicate and one propositional feature for each grounding of each formula. The weight associated with the feature is the weight attached to the corresponding formula. The ground Markov network represents the following probability distribution:

$$P_{\mathbf{\Theta}}(\omega) = \frac{1}{Z_{\mathbf{\Theta}}} \exp\left(\sum_i \theta_i N_i(\omega)\right) \quad (1)$$

where $N_i(\omega)$ is the number of groundings of f_i that evaluate to TRUE given ω and $Z_{\mathbf{\Theta}}$ is the normalization constant known as the partition function.

Important inference queries in MLNs are computing the partition function, finding the marginal probability of an atom given evidence (an assignment to a subset of variables) and finding the most probable assignment to all atoms given evidence (MAP inference). All these problems are computationally hard. Weight-learning, which is computing the optimal weights for a given MLN structure, and observed data, is typically done through Max-likelihood estimation. Weight-learning is also computationally challenging since inference procedures are used as a substep within each iteration of weight-learning.

Formulas:
$\mathtt{R}(x) \vee \mathtt{S}(x, y), w$
Original Domains:
$\Delta_x = \{A_1, B_1, C_1, D_1\}$
$\Delta_y = \{A_2, B_2, C_2, D_2\}$
Domain Approximation:
$\Delta'_x = \{\mu_1, \mu_2\}$
$\Delta'_y = \{\mu_3, \mu_\}$
(a)

Meta-Objects:
$\mu_1 = \{A_1, B_1\}$;
$\mu_2 = \{C_1, D_1\}$
$\mu_3 = \{A_2, B_2\}$; and
$\mu_4 = \{C_2, D_2\}$
(b)

Meta-Atoms:
$\mathtt{R}_1(\mu_1) = \{\mathtt{R}(A_1), \mathtt{R}(B_1)\}$
$\mathtt{R}_2(\mu_2) = \{\mathtt{R}(C_1), \mathtt{R}(D_1)\}$
$\mathtt{S}_1(\mu_1, \mu_2) =$
$\{\mathtt{S}(A_1, C_1), \mathtt{S}(A_1, D_1),$
$\mathtt{S}(B_1, C_1), \mathtt{S}(B_1, D_1)\}$
...
(c)

Figure 1: (a) an example MLN $\mathcal{M}$ and a possible domain approximation for the original domain of $\mathcal{M}$. $\mathcal{M}'$ contains meta-objects and meta-atoms, i.e., objects that represent multiple objects in the original domain and atoms that represent multiple atoms in $\mathcal{M}$ as shown in (b) and (c)

Evidence-based Clustering. Inference in MLNs is in general intractable. One recent technique for scaling up inference in MLNs is lifted inference. At a very high level, lifted inference techniques exploit symmetries present in the first-order MLN to scale up inference. However, in most practical MLNs, either the complexity of the MLN structure or the presence of evidence destroys symmetries and lifted inference algorithms that only try to exploit exact symmetries are as scalable as ground inference (van den Broeck and Darwiche 2013). To alleviate this problem, among others, Broeck and Darwiche (van den Broeck and Darwiche 2013) and Venugopal and Gogate (Venugopal and Gogate 2014) proposed approximate lifted inference algorithms that use clustering methods to group together approximately symmetrical objects. That is, given a set of domains $\mathcal{D} = \{D_1, \ldots D_M\}$, where each D_j is a set of real-world objects that can be instantiated in MLN $\mathcal{M}$, and evidence $\mathbf{E}$, we cluster each domain in $\mathcal{D}$ independently and replace the set of objects with *meta-objects*, i.e., the set of cluster-centers, to generate a new domain $\mathcal{D}' = \{D'_1, \ldots D'_k\}$, where each $|D'_j| << |D_j|$. Replacing the domain in $\mathcal{M}$ with $\mathcal{D}'$ yields a new MLN $\mathcal{M}'$ which we refer to as the *domain-approximated* version of $\mathcal{M}$. In $\mathcal{M}'$, each ground atom is now a *meta-atom* since it implicitly represents a set of ground atoms in $\mathcal{M}$. An example MLN and its domain approximation is shown in Fig. 1. Therefore, a threshold can be used to determine whether this meta-atom is to be considered observed or not, depending upon how many of the original atoms that it represents were observed evidence atoms. Thus, the original evidence is in a way represented in a "compressed" form by the new evidence.

Learning a Mixture Model for MLNs

We consider the discriminative weight learning task in MLNs with no hidden variables. Here, we pre-specify the structure of the MLN and the query atoms, and compute the weights of the MLN that maximizes the conditional log-likelihood (CLL) of a training relational database. Specifically, given a training relational database or world, ω (assignment to all ground atoms), and query atoms Q, the optimization problem is given by,

$$\max_{\Theta} P_{\Theta}(\omega|Q) \tag{2}$$

It is important to note that typically, for MLN learning, we only have a single instance to learn from unlike traditional machine learning algorithms, where we have multiple i.i.d instances. Further, to solve Eq. (2) using gradient ascent, it turns out that we need to run an inference algorithm to compute the gradient. Specifically, we need to compute $\mathbb{E}_{\Theta}[N_i(\omega)]$, which is the expected number of satisfied groundings of the i-th formula in the MLN (See proof in (Domingos and Lowd 2009)). Computing the gradient exactly is typically infeasible. Therefore, approximate inference methods are used to estimate this expectation, yielding approximate learning algorithms such as contrastive divergence (Lowd and Domingos 2007) and voted perceptron (Singla and Domingos 2005). However, it turns out that even these algorithms fail to scale up in practical cases, where the ground Markov network induced by ω is very large. An alternative approach is to use lifted inference such as lifted MCMC (Venugopal and Gogate 2012) or lifted MAP (Sarkhel et al. 2014) within learning which takes advantage of symmetries in the MLN. Unfortunately, most lifted inference methods fail when we present evidence to the MLN which destroys symmetries (van den Broeck and Darwiche 2013). In the case of discriminative learning, since we need to perform inference on the MLN conditioned on the query variables, this is problematic for lifted inference methods. Next, we describe our approach, where we reduce ω by exploiting approximate symmetries, and then learn a mixture model over multiple such reduced datasets.

Generating Reduced Datasets

We first generate a new training database ω' from ω, such that each domain in the MLN has a bounded number of objects, thus making learning from ω' scalable with any existing ground/lifted inference technique. Ideally, we would want the CLL function corresponding to ω' to be quite close to that corresponding ω. This will ensure that we learn weights from ω' that are similar to those we would have learned from ω. The naive approach is to randomly sample a subset of ω. However, this is problematic since the training database is relational, i.e., an atom is related to the other atoms in the training data. Therefore, random sampling is unlikely to preserve the relational structure in the database, and would therefore yield poor learning results. For example, Fig. 2 shows the ground network for an MLN with a single formula, $\texttt{Strong}(x) \Rightarrow \texttt{Wins}(\texttt{x},\texttt{y})$. Here, there are 6 atoms specified in the training database (shown as gray nodes in the figure). As seen in Fig. 2, there

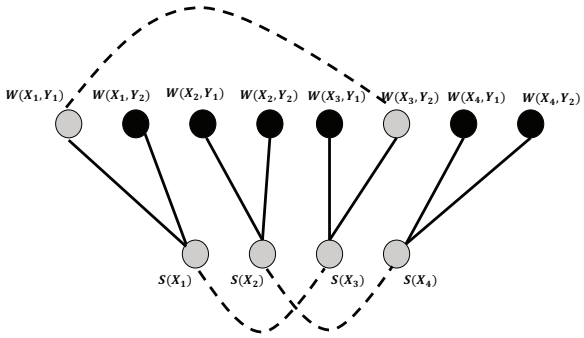


Figure 2: Example to illustrate sampling of approximately symmetric evidences for the MLN $\texttt{Strong}(x) \Rightarrow \texttt{Wins}(\texttt{x},\texttt{y})$ (abbreviated as S and W). Observed evidence variables are shown in gray, the black nodes are assumed to be false. Dashed curves show approximately symmetric evidences.

are approximately 2 distinct sub-structures in the MLN. Suppose we add a constraint that we can only choose 2 atoms corresponding to the `Strong` predicate and 1 atom corresponding to `Wins` predicate, we would want to choose atoms from varied sub-structures in order to better represent a distribution that is close to the original distribution. That is, suppose we choose $\texttt{Strong}(X_1)$, $\texttt{Strong}(X_3)$ and $\texttt{Wins}(X_1, Y_1)$, we completely loose the sub-structure that corresponds to $\texttt{Strong}(X_2)$, $\texttt{Strong}(X_4)$. A better choice is to pick $\texttt{Strong}(X_1)$, $\texttt{Strong}(X_2)$ and $\texttt{Wins}(X_1, Y_1)$ to represent both sub-structures that are present in the model.

In order to choose a subset of atoms from ω that best represent the relational dependencies in the MLN, we base our approach on the clustering method proposed by Venugopal and Gogate (Venugopal and Gogate 2014) (see the preliminary section). Specifically, we cluster each domain in the MLN using features that encode the approximate number of ground formulas that are true/false/unknown. Let $\mu_{i,j}$ represent the j-th cluster for the i-th domain of the MLN. Suppose we have a relation that joins the i_1-th $\ldots$ i_p-th domains, we sample from each of the sets $\{\mu_{i_1,1} \times \mu_{i_2,1} \ldots \mu_{i_p,1}\} \cap \omega$, $\{\mu_{i_1,1} \times \mu_{i_2,1} \ldots \mu_{i_p,2}\} \cap \omega$, $\ldots$ $\{\mu_{i_1,n_1} \times \mu_{i_2,n_2} \ldots \mu_{i_p,n_p}\} \cap \omega$, where $n_1 \ldots n_p$ are the number of clusters in the i_1-th $\ldots$ i_p-th domain respectively. We generate a sampling distribution over the atoms in a set as follows.

$$\hat{P}(X) = \begin{cases} 1/\sum_{k=1}^{p} n_k, & \text{if}(N_k + n_k) \leq \alpha, \forall i \\ 1, & \text{if}(\sum_{k=1}^{p} n_k = 0) \\ 0, & \text{Otherwise} \end{cases} \tag{3}$$

where X is an atom, $\hat{P}(X)$ is its un-normalized probability of being sampled, p is the arity of the relation, n_k is the number of objects that would be added to the k-th domain of the MLN if X were to be sampled, and N_k is the total number of objects that have already been added to the k-th domain. Intuitively, if sampling an atom does not increment the domains in the MLN (since it refers to objects that were already added to the domain from a previously sampled atom), we would sample it with high probability. However, if

any of the domains that would be incremented by sampling X have already reached their limit, α, then we do not sample X. Note that depending on the order in which we sample the sets, the sampling distributions for each set can change, and thus, we can generate different reduced databases ω' from the same training database ω.

Mixture Model

Note that each different sampled training database generated from ω using Eq. (3) will induce a different underlying graphical model structure. Thus, if we use Eq. (2) to learn the weights from each database, we will learn different sets of weights for the same MLN. We now argue why learning using a single (or even best) reduced training database in this context may not be the best approach. Consider a simple MLN, $\mathtt{R}(x) \wedge \mathtt{S}(x)$. Recall that the weights learned for this MLN depend upon the true groundings for this MLN in the training dataset. Specifically, the learning algorithms try to match the expected number of true groundings (using the weights) to the observed number of true groundings (in the data). Fig 3 illustrates the situation where for a given dataset corresponding to the example MLN, we sample the dataset and measure the number of true groundings in the dataset. Specifically, we sample 25%, 50% and 75% of the dataset 25 times each, and construct a histogram of the results obtained. As we can see, there are several peaks in the distribution. If we only take a single sample, we would probably end up sampling a single peak, and the weights may be quite biased. Using a mixture model, we propose to hit several points in the distribution and average the results, leading to a better-learned model.

More formally, we can make arguments from a Bayesian perspective. This argument is similar to the arguments made by Draper (Draper 1995) and Smyth and Wolpert (Smyth and Wolpert 1999), where they argue that from a Bayesian sense, combining different learned distributions is beneficial when there is uncertainty in the learned model. For this, let us assume that we can, in fact, represent the original distribution using a model learned from the reduced training datasets ((Wolpert 1996) discusses such assumptions). Note that during learning, there is uncertainty over both the structure of the MLN as well as the weights learned for that MLN. Thus, we would integrate over all possible structures that can be generated from ω, as well as all the weights that can be learned for a specific structure, to obtain the posterior distribution of the model. Specifically, the probability that the model $(\mathcal{M}; \Theta_{\mathcal{M}})$ generated the database ω is given by,

$$P((\mathcal{M}; \Theta_{\mathcal{M}})|\omega) = \sum_{\mathcal{M}} \int_{\Theta_{\mathcal{M}}} P((\mathcal{M}; \Theta_{\mathcal{M}})|\omega) d\Theta_{\mathcal{M}} \quad (4)$$

where $\mathcal{M}$ is the structure of the MLN, $\Theta_{\mathcal{M}}$ are the weights corresponding to $\mathcal{M}$. The above expression can be written as,

$$P((\mathcal{M}; \Theta_{\mathcal{M}})|\omega) = \sum_{\mathcal{M}} \int_{\Theta_{\mathcal{M}}} P(\Theta_{\mathcal{M}}|\omega, \mathcal{M}) d\Theta_{\mathcal{M}} \quad (5)$$

$$\times P(\mathcal{M}|\omega) \quad (6)$$

Eq. (6) suggests that it is better to compute a weighted average over different models rather than use a single model. Each model is weighted by $P(\mathcal{M}|\omega)$. However, computing these weights is not trivial, and requires us to compute the distribution over all possible structures. Therefore, we estimate these weights empirically using *stacking*.

Stacked MLNs Let $P_{\Theta_1}, P_{\Theta_2}, \ldots P_{\Theta_k}$ be k distributions corresponding to the k reduced databases, $\omega_1 \ldots \omega_k$. That is, we perform weight learning on $\omega_1 \ldots \omega_k$ to obtain the weights $\Theta_1 \ldots \Theta_k$ respectively. We now define a mixture distribution that approximates $P_{\Theta}(\omega|Q)$ as,

$$\sum_{j=1}^{k} \phi_j P_{\Theta_j}(\underline{\omega}|Q) \quad (7)$$

where ϕ_j is the mixture-coefficient corresponding to j-th distribution in the mixture, $\underline{\omega}$ represents any possible world, and Q is the set of variables designated as query atoms. To learn the mixture model discriminatively, given ω, we maximize the overall conditional log-likelihood (CLL) as,

$$\max_{\phi_1 \ldots \phi_k, \Theta_1 \ldots \Theta_k} log \sum_{j=1}^{k} \phi_j P_{\Theta_j}(\omega|Q) \quad (8)$$

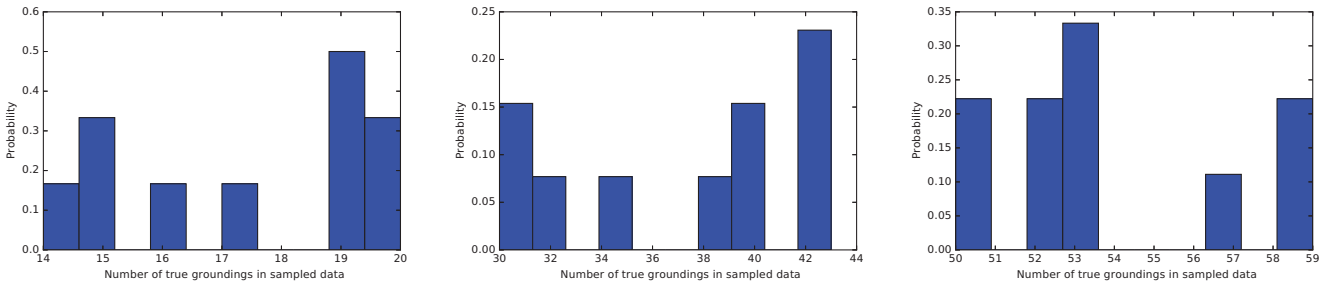
One way to solve the optimization problem in Eq. (8) is to use the standard EM algorithm. Specifically, we treat the coefficients as hidden variables in the model. In the E-step, we fill the hidden variables, and in the M-step we perform a joint max-likelihood optimization over all the components in the mixture. Typically, the M-step is easy in most applications of EM such as Gaussian Mixture Models. Unfortunately, this step is costly for MLNs. Specifically, since there is no closed form solution for the max-likelihood, we need to run an expensive inference procedure to perform the optimization in each iteration of EM. This approach is not scalable for real-world MLN applications. Therefore, alternative techniques to the standard EM method need to be developed for MLNs (Domingos and Lowd 2009). Here, we propose to use a technique called stacked density estimation (Smyth and Wolpert 1999) to learn the model hierarchically.

To learn our model, we define a modified function that maximizes the "out-of-sample" CLL of the dataset. This is needed to ensure that our model generalizes well to unseen cases. Specifically,

$$\max_{\phi_1 \ldots \phi_k, \Theta_1 \ldots \Theta_k} log \sum_{j=1}^{k} \phi_j \hat{P}_{\Theta_j}(\omega_j|Q) \quad (9)$$

where $\hat{P}_{\Theta_j}(\omega_j|Q)$ is the CLL assuming that ω_j is not used when computing Θ_j.

To solve Eq. (9), we perform a procedure similar to cross-validation in ML methods. Specifically, we divide the input data into v folds, $\omega_1 \ldots \omega_v$. We learn all the weight-vectors $\Theta_1 \ldots \Theta_k$ from $v-1$ folds and estimate the CLL on the remaining fold, and repeat this over all folds. We then compute the optimal mixture coefficients that maximize the test CLL scores given the optimal weight-vectors. However, note that the out-of-sample CLL cannot be computed exactly



(a) 25% (b) 50% (c) 75%

Figure 3: Example to illustrate the distribution of true groundings in a sampled dataset for $\mathtt{R}(x) \wedge \mathtt{S}(x)$. The sampling is repeated 25 times at sampling percentages of 25%, 50% and 75%, to construct the histograms in (a), (b) and (c) respectively.

for MLNs. Therefore, we calculate a rough estimate of the CLL as $P_\Theta(\omega) = \sum_{q \in Q} log P_\Theta(q)$. That is, we compute the product of the marginal probabilities of the query atoms in the test fold, and use it in place of the true CLL (Lowd and Domingos 2007). Each marginal probability is estimated using approximate inference procedures. Next, we describe our approach to learn the mixture coefficients.

Let $\hat{P}_{\Theta_1}(\omega_i|Q)$, $\hat{P}_{\Theta_2}(\omega_i|Q)$ ... $\hat{P}_{\Theta_k}(\omega_i|Q)$ be the estimated out-of-sample CLLs for the i-th fold. That is, we compute k reduced evidence databases from the remaining folds, learn the weights for the given MLN structure from each of these databases, and compute the test CLL on the i-th fold for each learned MLN. The optimization problem that we now need to solve is given by,

$$\max_{\phi_1 \ldots \phi_k} \sum_{i=1}^{v} log \sum_{j=1}^{k} \phi_j \hat{P}_{\Theta_j}(\omega_i|Q) \tag{10}$$

Note that the optimization problem specified in Eq. (10) is non-convex, and therefore hard to optimize analytically. Instead, we solve the optimization problem in Eq. (10) using the EM algorithm for learning finite mixture models. However, unlike the traditional EM algorithm for learning finite mixtures, here, the parameters of the component distributions are assumed to be fixed, i.e., the MLN weights remain unchanged for all k components, and only the mixture coefficients are modified in each step.

Specifically, let W be a $v \times k$ matrix of weights that determine the relative importance of each mixture component. The (i, j)-th entry in W is given by,

$$w_{ij} = \frac{\phi_j \hat{P}_{\Theta_j}(\omega_i)}{\sum_{m=1}^{k} \phi_m \hat{P}_{\Theta_m}(\omega_i)} \tag{11}$$

We start by initializing the mixture coefficients to random initial values, $\phi_1^{(0)} \ldots \phi_k^{(0)}$. In each subsequent step, we recompute each mixture coefficient as,

$$\phi_j^{(t+1)} = \frac{\sum_{i=1}^{v} w_{ij}}{k} \tag{12}$$

We terminate the algorithm once the coefficients converge to a fixed point. In our experiments we used the stopping criterion, $\max_i |\phi_i^{(t)} - \phi_i^{(t-1)}| \leq 0.0001$. To avoid local minima, we run the algorithm from several initial random starting states $\phi_1^{(0)} \ldots \phi_k^{(0)}$, and average the converged coefficient values across these runs.

Algorithm 1: Learning the MLN Mixture

Input: Training database ω, number of components k, MLN structure μ, Query Q
Output: Mixture distribution of MLNs

Divide ω into v folds, $\omega_1 \ldots \omega_v$
for *each fold* ω_i **do**
  `// Learn the MLN weights using` $\omega \setminus \omega_i$ `as the training database`
  for *j = 1 to k* **do**
    Reduce $\omega \setminus \omega_i$ to $\hat{\omega}$ by sampling from approximately symmetric clusters
    Learn the MLN weights discriminatively from $\hat{\omega}$ to obtain Θ_i
    Compute the CLL, $\hat{P}_{\Theta_j}(\omega_j)$ by running inference on Q using ω_i as evidence
initialize the W matrix as in Eq. (11)
`// Learn the mixture coefficients`
Initialize $\phi_1^{(0)} \ldots \phi_k^{(0)}$
while *Not converged* **do**
  Update $\phi_1^{(t)} \ldots \phi_k^{(t)}$ using Eq. (12)
  Update W using Eq. (11)
return $\sum_{j=1}^{k} \phi_j^{(t)} \hat{P}_{\Theta_j}$

Algorithm 1 summarizes our approach for learning the mixture of MLNs. We first divide the data into v folds, learn k MLNs from $v-1$ folds, and compute the test likelihood in the v-th fold. For learning each of he k MLNs, we cluster approximately symmetrical evidences, and generate a reduced representation of the input training database. Once we learn all the MLNs, we learn the mixture coefficients that maximizes Eq. (10) by alternating between updating the coefficients and updating the W matrix.

Note that Algorithm 1 can be efficiently implemented since each MLN in the mixture can be learned independently of the other. Specifically, we learn each of the k models in parallel for each of the v folds. Similarly, the W matrix can

be computed in an embarrassingly parallel manner. That is, we compute the out-of-sample CLL for each of the v folds, and for each of the k models in parallel.

Since Algorithm 1 is essentially the EM algorithm for finite mixtures, with fixed distribution parameters, the following result holds from the convergence proof of the EM algorithm (cf. (Dempster, Laird, and Rubin 1977; Meila and Jordan 2000))

Proposition 1. *Algorithm 1 converges to a locally optimal solution for Eq.* (10)*.*

Related Work

Several prior studies have focused on weight learning in MLNs. Singla and Domingos (Singla and Domingos 2005) proposed to use approximate MAP inference to perform learning efficiently. Similarly, Lowd and Domingos (Lowd and Domingos 2007) developed discriminative learning methods that use MCSAT sampling, diagonal Newton and scaled conjugate gradient. Haaren et al. (Haaren et al. 2016) recently developed a generative weight learning method that uses lifted inference. Since generative learning does not condition on evidence, lifted inference is somewhat easier to apply in such learning algorithms. However, even for generative learning, for practical MLNs, we need to approximate the gradient since they are typically not liftablem, even in the absence of evidence. To scale up discriminative learning, Sarkhel et al. (Sarkhel et al. 2016) used efficient counting within contrastive divergence, voted perceptron and pseudo-likelihood learning. The work by Ahmadi et al. (Ahmadi et al. 2013) is related to our work in the sense that Ahmadi et al. propose to use mini-batches of training data, and learn the MLN weights in an online manner. However, they do not explore mixing several models, as we do in this work. Khot et al. (Khot et al. 2015) developed methods to learn MLNs with missing data using a EM-based approach. However, as far as our knowledge goes, ours is the first work to explore the use of mixture models to improve MLN weight-learning.

Experiments

Setup

We used three benchmarks from Alchemy, namely WebKB, Protein, and ER, to evaluate our approach. We compared our approach with Tuffy (Niu et al. 2011), the current state-of-the-art MLN system, and also Magician (Venugopal, S.Sarkhel, and Gogate 2016), which implements scalable versions of contrastive divergence (CD), voted perceptron (VP) and pseudo-log-likelihood maximisation (PLL), using approximate counting oracles (Sarkhel et al. 2016). Both these systems are available as open-source. We also tried to use Alchemy, an older MLN learning, and inference system, but it did not work with any of our datasets in our experiments since it ran out of memory during the grounding process. Further, we created another baseline method where we used Venugopal and Gogate's (Venugopal and Gogate 2014) approach (available in the Magician open-source code) to compress the distribution. We then used the compressed distribution generated by this method to learn the MLN weights using Tuffy, and we refer to this approach as VG. Finally, we created a baseline where we randomly sample from the training dataset and create a reduced dataset on which we run weight learning using Tuffy. We refer this simple baseline as Random.

We implemented the mixture model by learning the components of the mixture in parallel. Specifically, we used a cluster of k 8GB quad-core machines for k components of the mixture, where we performed the learning using Tuffy, and computed the out-of-sample CLL using MCSAT.

Results

We evaluated our approach on three key aspects, (i) Solution Quality, (ii) Stability, and, (iii) Scalability. For evaluating solution quality, we have reported the cross validated test CLL score. For stability, we have reported the variance in weight as well as the variance in CLL and finally, to measure scalability we have reported the running time of competing approaches.

Solution Quality Table 1 shows our results where we compute CLL through 5-fold cross validation. The CLL is approximated using the marginal probabilities computed using MCSAT. For the mixture model, we set the number of clusters as 5% of the original domain-size and used the K-Means algorithm for clustering. We used five components in our mixture model. To compute the W matrix, we divided the training data into 5 folds.

As we see in the results shown in Table 1, our approach using the Mixture MLN substantially outperforms both Tuffy and all algorithms in Magician, in terms of the CLL scores. Further, note that Tuffy could only work on the WebKB MLN, and even on this MLN, it took an extremely long time ($>$ 10 hours) for the weights to converge. On the other original MLNs, it failed to work even after running it for a day. In contrast, Magician could learn weights on all the benchmarks. However, its CLL scores were considerably lower than our mixture model. This phenomenon was observed for all algorithms in Magician, i.e., CD, VP and PLL; and suggests that approximating the dataset yields more accurate results than approximating the learning methods (as in Magician)

Variance in Weights To understand the importance of using the mixture model, we performed an experiment where we compare how far off the different weights learned in a mixture component are, as compared to the true weights. Specifically, we designed a very simple MLN consisting purely of singleton atoms where learning and inference are well-known to be easy. We used eight predicates distributed between 4 randomly generated formulas. Since the MLN is easy to learn, we consider Tuffy's weights learned from the full dataset as the accurate or true weights for the dataset. We then used our clustering approach and generated 25 reduced datasets, and learned the weights for each of them. Fig. 4 shows our results in terms of the average absolute error between the true weights and the weights learned from the reduced dataset. The variance in error illustrates why a mixture model is more reliable to approximate the true weights as compared to learning from a single reduced database.

Dataset	Magician			Tuffy	VG	Random	Mixture MLN
	CD	VP	PLL				
WEBKB	-0.66	-0.91	-0.72	-0.89	-0.35	-1.82	**-0.13**
PROTEIN	-0.779	-0.78	-0.74	X	-0.67	-1.69	**-0.16**
ER	-0.694	-0.693	-0.693	X	-0.56	-0.85	**-0.148**

Table 1: Comparison of results in terms of estimated average CLL scores using 5-fold cross validation (larger is better).

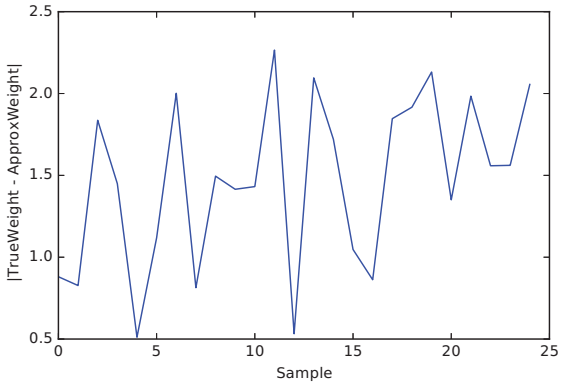


Figure 4: Illustrating the variance in error for learned weights from reduced datasets.

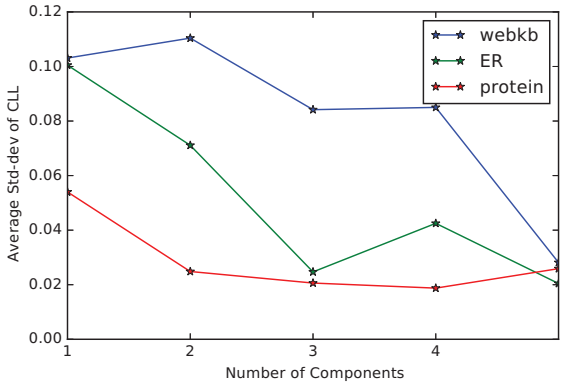


Figure 5: Illustrating the variance in CLL for varying mixture components.

Variance in CLL Fig. 5 illustrates the variance in CLL as we vary the number of mixture components. Specifically, we learned 12 different MLNs based on reduced databases. We then sampled k mixture components from these MLNs, and performed stacked learning. We then measured the CLL on the test dataset and recorded the standard deviation of the CLL as we sample different sets of k mixture components. As seen in the figure, the standard deviation reduces as we increase the mixture components. However, knowing the ideal number of mixture components is a hard problem (just as in other mixture models). In future research, we will investigate non-parametric approaches to compute the right number of mixture components needed for a dataset.

Dataset	Tuffy	Magician	VG	Random	Mixture
WEBKB	10 hours	24 hours	15 mins	12 mins	22 mins
PROTEIN	X	3 mins	25 mins	25 mins	45 mins
ER	X	10mins	28 mins	20 mins	40 mins

Table 2: Comparison of time required for learning.

Scalability Table 2 illustrates the amount of time required for learning using the various algorithms. As shown here, Tuffy and Alchemy are the slowest systems, and while Tuffy runs on one benchmark but takes a long time, Alchemy fails on all benchmarks (therefore not shown in table). Magician (with the lowest possible ibound for the approximate counting oracle) is very fast for certain benchmarks but extremely slow for others (such as webkb). The random sampling and learning using VG run in quite similar times. For our approach, since it is parallelizable, we present the longest time that it took to learn a single component. Note that, this includes the learning as well as inference time required in our stacking procedure. As shown here, our system performs favorably in terms of time while being much more accurate than other systems.

Conclusion

In this paper, we proposed a novel, scalable approach to discriminative weight learning in MLNs. Specifically, we first generated smaller sized training databases from the original data, by clustering based on approximate symmetries and then sampling a representative atom from each cluster. However, doing so results in more uncertainty in the learned model. To reduce this, we learned multiple MLNs and combined them through a mixture model. However, learning a mixture model through traditional EM is expensive. Therefore, we use a stacking approach, where we fix the MLN weights and optimize the mixture coefficients by maximizing the out-of-sample CLL. Our experiments on different benchmarks and comparisons with state-of-the-art MLN learning systems illustrated accuracy and scalability of our method.

Future work includes incorporating nonparametrics (Venugopal, Sarkhel, and Cherry 2016) to compute the optimal number of components, boosting methods for MLNs, etc.

Acknowledgments

This work was supported in part by an Adobe Research Gift Award.

References

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

Collins, M. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 1–8. Philadelphia, PA: ACL.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society, Series B* 39:1–38.

Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. San Rafael, CA: Morgan & Claypool.

Draper, D. 1995. Assessment and propagation of model uncertainty (disc: p71-97). *Journal of the Royal Statistical Society, Series B: Methodological* 57:45–70.

Gogate, V., and Domingos, P. 2011. Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, 256–265. AUAI Press.

Haaren, J. V.; den Broeck, G. V.; Meert, W.; and Davis, J. 2016. Lifted generative learning of markov logic networks. *Machine Learning* 103(1):27–55.

Hinton, G. E. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation* 14(8):1771–1800.

Khot, T.; Natarajan, S.; Kersting, K.; and Shavlik, J. W. 2015. Gradient-based boosting for statistical relational learning: the markov logic network and missing data cases. *Machine Learning* 100(1):75–100.

Kok, S.; Sumner, M.; Richardson, M.; Singla, P.; Poon, H.; and Domingos, P. 2006. The Alchemy System for Statistical Relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu.

Lowd, D., and Domingos, P. 2007. Recursive random fields. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 950–955. Hyderabad, India: AAAI Press.

Meila, M., and Jordan, M. I. 2000. Learning with mixtures of trees. *Journal of Machine Learning Research* 1:1–48.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. W. 2011. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB* 4(6):373–384.

Poole, D. 2003. First-Order Probabilistic Inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 985–991. Acapulco, Mexico: Morgan Kaufmann.

Sarkhel, S.; Venugopal, D.; Singla, P.; and Gogate, V. 2014. Lifted MAP inference for markov logic networks. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS*, 859–867.

Sarkhel, S.; Venugopal, D.; Pham, T. A.; Singla, P.; and Gogate, V. 2016. Scalable training of markov logic networks using approximate counting. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 1067–1073.

Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 868–873. Pittsburgh, PA: AAAI Press.

Singla, P., and Domingos, P. 2006. Entity Resolution with Markov Logic. In *Proceedings of the Sixth IEEE International Conference on Data Mining*, 572–582. IEEE Computer Society Press.

Smyth, P., and Wolpert, D. 1999. Linearly combining density estimators via stacking. *Machine Learning* 36(1):59–83.

van den Broeck, G., and Darwiche, A. 2013. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems 26*, 2868–2876.

Van den Broeck, G.; Taghipour, N.; Meert, W.; Davis, J.; and De Raedt, L. 2011. Lifted Probabilistic Inference by First-Order Knowledge Compilation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2178–2185.

Venugopal, D., and Gogate, V. 2012. On lifting the gibbs sampling algorithm. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, 1664–1672.

Venugopal, D., and Gogate, V. 2014. Evidence-based clustering for scalable inference in markov logic. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 258–273.

Venugopal, D.; Chen, C.; Gogate, V.; and Ng, V. 2014. Relieving the computational bottleneck: Joint inference for event extraction with high-dimensional features. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 831–843. ACL.

Venugopal, D.; Sarkhel, S.; and Cherry, K. 2016. Non-parametric domain approximation for scalable gibbs sampling in mlns. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence*, 745–755. AUAI Press.

Venugopal, D.; S.Sarkhel; and Gogate, V. 2016. Magician: Scalable Inference and Learning in Markov logic using Approximate Symmetries. Technical report, Department of Computer Science, The University of Memphis. https://github.com/dvngp/CD-Learn.

Wolpert, D. H. 1996. Reconciling bayesian and non-bayesian analysis. In *Maximum Entropy and Bayesian Methods*. Springer. 79–86.