Efficient Weight Learning in High-Dimensional Untied MLNs

Khan Mohammad Al Farabi
The University of Memphis
kfarabi@memphis.edu

Somdeb Sarkhel
Adobe Research
sarkhel@adobe.com

Deepak Venugopal
The University of Memphis
dvngopal@memphis.edu

Abstract

Existing techniques for improving scalability of weight learning in Markov Logic Networks (MLNs) are typically effective when the parameters of the MLN are tied, i.e., several ground formulas in the MLN share the same weight. However, to improve accuracy in real-world problems, we typically need to learn separate weights for different groundings of the MLN. In this paper, we present an approach to perform efficient weight learning in MLNs containing high-dimensional, untied formulas. The fundamental idea in our approach is to help the learning algorithm navigate the parameter search-space more efficiently by a) tying together groundings of untied formulas that are likely to have similar weights, and b) setting good initial values for the parameters. To do this, we follow a hierarchical approach, where we first learn the parameters that are to be tied using a non-relational learner. We then use a relational learner to learn the tied-parameter MLN with initial values derived from parameters learned by the non-relational learner. We illustrate the promise of our approach on three different real-world problems and show that our approach yields much more scalable and accurate results compared to existing state-of-the-art relational learning systems.

1 Introduction

Markov Logic Networks (MLNs) [6] repesent uncertain knowledge using weighted first-order logic formulas, where weights (that encode uncertainty) are shared between all groundings of a formula. However, learning MLN weights is quite challenging since the probabilistic graphical model underlying the MLN is extremely large even though the number of weights to be learned is quite small. A popular approach is to use *lifted* inference [17] methods that exploit symmetries, to scale up weight learning. However, doing so presents two significant difficulties. First, *discriminative* learning methods break symmetries due to conditioning over non-query variables, and second, when encoding features in real-world applications, having too many symmetries does not yield an accurate or expressive-enough model. For example, suppose we are developing a classifier for topics in a page, a feature commonly used is the words in the page. Encoding this into a formula `Word`(w, p) $\Rightarrow$ `Topic`(p, t), i.e., having all instantiations of this formula share the same weight is not useful. Therefore, typically such formulas are modeled using "+" variables, i.e., `Word`$(+w, p)$ $\Rightarrow$ `Topic`$(p, +t)$, which signifies that each grounding of the "+" variables, namely each word and topic-class, has a different weight. We call such formulas as *untied* formulas. Naturally, for MLNs that encode high-dimensional features, the increased number of weights, and a loss of symmetry due to untied formulas makes learning even more difficult.

Our main contribution in this paper is an approach to perform efficient learning in MLNs containing high-dimensional, untied formulas. Specifically, as shown in prior applications [22], encoding high-dimensional features as MLN formulas makes learning infeasible using relational learning, since we need to learn a large number of parameters from essentially a single instance (the training data is a single relational database). To address this, we a) reduce the parameter search-space by *tying* together groundings of untied formulas that are likely to have similar weights, and b) perform weight initialization for untied formulas for faster convergence. Naturally, to begin with, we do not know which groundings of untied formulas will turn out to have the same weights. To predict similarity in learned weights w.r.t to a given problem-of-interest, we encode groundings of the untied formula as features in a non-relational model that learns a classifier for this problem. We then tie together groundings with similar parameter values

Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

in the non-relational learner and initialize a tied-weight with the average parameter value. Our underlying assumption being that groundings that exhibit similarity in the non-relational learner will also exhibit similarity in the relational learner. Further, tying the parameters also in a sense regularizes the model, leading to better generalization. On the tied-parameter MLN, we use a relational learner to learn far fewer parameters, and with stronger initial values, thus making it more efficient.

We perform experiments on three different real-world problems, namely collective classification of web-page topics, entity resolution, and information extraction using datasets available in Alchemy [12]. For our experiments, we integrate SVMs with relational learning, and show that our approach yields much more scalable and accurate results, as compared to using state-of-the-art relational learning systems such as Tuffy [16].

2 Background

2.1 Markov Logic Networks

Markov logic networks (MLNs) are template models that define uncertain, relational knowledge as first-order formulas with weights. The larger the weight of a formula, the more likely is that formula to be true. ∞ weight formulas are hard constraints which should always be true. Similarly formulas with $-\infty$ weights are always false. Thus, MLNs offer a flexible framework to mix hard and soft rules. Given a set of constants that represent the domains of variables in the MLN, an MLN represents a factored probability distribution over the possible worlds, in the form of a Markov network. A world in an MLN is an assignment of 0/1 to all ground atoms of the MLN (first order predicates in the MLN whose variables have been substituted with a constant). Specifically, the distribution is given by,

$$\Pr(\omega) = \frac{1}{Z} \exp\left(\sum_i w_i N_i(\omega)\right) \tag{1}$$

where w_i is the weight of formula f_i, $N_i(\omega)$ is the number of groundings of formula f_i that evaluate to True given a world ω, and Z is the normalization constant.

As a simple example of an MLN, suppose we want to encode the fact that smokers and asthmatics are not friends. We would design an MLN with a formula such as $\texttt{Smokes}(x) \wedge \texttt{Friends}(x,y) \Rightarrow \neg\texttt{Asthma}(y)$. Given constants corresponding to the variables, x and y, the MLN represents a joint distribution over all ground atoms of `Smokes`, `Friends`, and `Asthma`. The two key tasks in MLNs are *weight learning*, which is the task of learning the weights attached to the formulas from a relational database, and *inference* (prediction), which is the task of answering queries posed over the learned model given observations (evidence database). The two main inference problems over MLNs are (1) posterior or marginal estimation which involves finding the marginal probability distribution of a random variable (ground atom) in the ground Markov network given an evidence database. For example computing the probability that $\texttt{Smokes}(Ana)$ is true given $\texttt{Smokes}(Bob)$ is true and $\texttt{Friends}(Ana, Bob)$ is true. (2) Maximum-a-posteriori (MAP) estimation which involves finding an assignment of values to all non-evidence variables such that the assignment has the maximum probability. For example, given that $\texttt{Smokes}(Ana)$, $\texttt{Smokes}(Bob)$ and $\texttt{Friends}(Ana, Bob)$ is true, what is the assignment to all other ground atoms in the MLN such that their joint probability is maximum.

Weights attached to first-order formulas can be learned either *generatively*, in which the goal is to find weights that maximize the log-likelihood of the data, or *discriminatively*, in which the goal is to maximize the conditional log-likelihood of the data given evidence. In principle, both tasks can be solved using a standard gradient ascent procedure with appropriate regularization constraints. However, it turns out that computing the gradient is computationally intractable, since the problem of computing the gradient can be reduced to the posterior marginal estimation task. Therefore, practitioners often use alternate objective criteria such as contrastive divergence [9], voted perceptron [4], and pseudo log-likelihood [2] since the gradient of these functions is easier to compute.

2.2 Related Work

In many practical applications of MLNs, researchers have used various strategies to reduce the complexity of learning and inference. Especially when the MLN contains formulas that are hard to learn, alternate techinques have been used to set the weights of such formulas. For example, Venugopal et al. [22] learn high-dimensional linguistic features using SVMs, Khot et al. [11] set some weights manually for hard-to-learn formulas in question answering, etc. Previously, Craven and Slattery [5] proposed an approach to utilize Naive Bayes within propositional rule-based learning through FOIL. Our approach can be seen as an analogous approach but for learning first-order relational models.

On the other hand, there have been a few approaches to perform weight learning in a more efficient manner. Haaren et al. [8] used lifted inference in generative learning where symmetries are better preserved. Recently, Mittal et al. [15] proposed a new approach to learn more fine-grained weights. This approach is similar to ours, but they learn the grouping through hidden variables, and therefore, they need to sum it out using the

EM algorithm which is expensive. Particularly, in the presence of high-dimensional, untied formulas, which is our main focus here, their approach is computationally much more expensive, and less scalable as compared to our method. Chou et al [3]proposed a similar idea, where they quantized Bayesian network parameters based on similar values in the CPT. However, in the case of MLNs, learning the original parameters is infeasible, which makes the quantization hard. Sarkhel et al. [19] proposed an efficient discriminative learning approach to learn more efficiently by using approximate counting oracles. But in this case, they assume that the weights of all groundings of a formula are shared. Ahmadi et al. [1] proposed an approach to perform learning in an online manner by using mini-batches of data. Specifically, they partially ground the MLN using a part of the dataset each time, and update the weights incrementally. However, even with these advancements, applying MLNs to practical applications remains challenging, where we need to encode features that may possibly result in hundreds and thousands of ground formulas, and we would need to perform relational weight learning in an extremely large parameter space. The approach that we propose in this paper is an approach to make relational learning applicable to large problems, by integrating it with scalable non-relational learners.

3 Weight Learning for Untied MLNs

Given a relational, closed-world training database $\mathcal{D}$, MLN structure $\mathcal{M}$ that encodes high-dimensional untied formulas, and query predicates $\mathbf{Q}$, our task is to learn weights $\mathbf{w}$ for $\mathcal{M}$ that maximizes the conditional log-likelihood (CLL) $P_{\mathcal{M}}(\mathcal{D}|\mathbf{Q})$. Instead of learning a separate weight for each grounding of the untied formulas, we learn weights for a smaller set of formulas by tying together groups of untied formulas. To tie the untied formulas together optimally, we learn which of the untied formulas are likely to have similar weights, using a supervised non-relational learner. We then use weights derived from the non-relational learner as initialization weights when we re-learn the model using a relational learner.

3.1 Encoding Untied Formulas

We make the following assumptions about the structure of the untied formulas. All the assumptions are reasonable assumptions, and are typically satisfied when we consider the design of MLNs with untied formulas.

- Each formula is a universally quantified clause.
- + variables are a part of the predicate definition. That is, if an argument of a predicate is defined with the + symbol, then all atoms that occur in the MLN corresponding to that predicate are assumed to have + variables in that argument position.
- Each untied formula contains at least one query atom, and each atom in an untied formula contains at least one + variable. The + variables in query predicates define classes that the MLN is designed to predict. Note that for a binary classification problem + variables are strictly not needed, however for ease of explanation, we add a + variable with domains-size of 2. For the query predicates, there is a mutual exclusion constraint imposed on the arguments defined with the + symbol. Specifically, for every grounding to the remaining variables, one and only one ground atom is true among all the ground atoms obtained by grounding the + variables.
- If an untied formula contains multiple query atoms, one of query atoms is designated as the target atom that we are trying to predict. Each query atom must be the target of exactly one untied formula.

Note that with the last two assumptions mean that, we assume untied formulas ti typically model "features" for a classification task. Note that, if none of the variables in the untied formula are query variables, then, learning separate weights will not make sense, since we assume that the training database contains an assignment to all non-query variables (closed world assumption).

Let f be an untied formula. Let $\mathcal{X}_-$ denote the lifted variables in f, i.e., variables that are not associated will have a + symbol. We divide f into two parts, the target atom of f whose class value we are trying to predict, and the rest of the formula in f denoted by $f^{\bar{Q}}$. Let $\mathcal{X}_+$ be the + variables in $f^{\bar{Q}}$.

Case 1. $f^{\bar{Q}}$ has no query atoms. In this case, all atoms in $f^{\bar{Q}}$ are observable, therefore we can use them directly to learn to classify the target atom. To do this, we consider each grounding to $\mathcal{X}_-$ as an i.i.d instance for our non-relational learner. The features are defined by projecting $\mathcal{D}$ on $f^{\bar{Q}}$ independently for every instance. Specifically, let $\bar{x}_-^j$ be the j-th grounding to $\mathcal{X}_-$, $\bar{x}_+^k$ be the k-th grounding to $\mathcal{X}_+$, and $\bar{f}_{jk}^{\bar{Q}}$ be the formula obtained by grounding $f^{\bar{Q}}$ with $\bar{x}_-^j$ and $\bar{x}_+^k$, we derive the feature vector for the j-th instance, $\mathbf{X}_j$ as,

$$X_{jk} = \begin{cases} 1 & \text{if } \bar{f}_{jk}^{\bar{Q}} \text{ is true in } \mathcal{D} \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

Case 2. $f^{\bar{Q}}$ has one or more query atoms. In this case, since query atoms are considered unknown, we cannot use the query atoms in $\mathcal{D}$ directly to learn to classify the target of f. Therefore, we define a pipeline processing order for query atoms. That is, we will have predictions for all query atoms in $f^{\bar{Q}}$ before we can classify the target atom of f. Thus, we compute an ordering over the untied formulas $f_1 \ldots f_n$ such that, the predictions for all query atoms in f_k (other than the target of f_k) are available from $f_1 \ldots f_{k-1}$. In the event that such an ordering is impossible for a formula f_i, we drop the query atoms that cannot be predicted from formulas $f_1 \ldots f_{i-1}$, when computing the features for f_i. The feature matrix is computed as in case 1, except that, we do not use the query atoms in $\mathcal{D}$ but instead use the predicted values for the query atoms when computing the feature vector from Eq. (2).

Given the feature vector for all instances of f note that we can use any classifier to learn to classify the target of f. The only requirement from the classifier is that it needs to assign a weight for each dimension of the feature vector which can be used to interpret the contribution of a dimension w.r.t the classification of the target. In our experiments, we used multiclass SVMs as our non-relational classifier. However, our approach can plug-in several other classifiers, and is thus a general method to integrate relational and non-relational classifiers. We utilize the scalability of non-relational classifiers for handling high-dimensional formulas, and relational learning for learning dependencies across instances, thus yielding the best of both worlds.

In the case of SVMs, we use the (normalized) coefficients of the learned hyper-planes of the SVM as the initialization weights for our formulas. Specifically, if the target of f has m classes, for i-th feature dimension, we have a vector of weights Θ_i, where the j-th component of the vector, θ_{ij}, is the j-th coefficient of the hyper-plane that distinguishes class i from the rest of the classes of the target. Thus, the i-th grounding to $\mathcal{X}_+$, and the grounding corresponding to the k-th class of the target atom will get the initialization weight θ_{ij}.

3.2 Clustering

Once we obtain the initialization weights for each grounding of the untied formulas, we reduce the number of formulas by clustering those with similar weights together. Specifically, for every cluster, we replace all the formulas in that cluster with a single formula, with initialization weight equal to the average weight of all formulas in that cluster. However, the clustering must be chosen carefully such that the grouping of formulas is consistent.

For example, consider the formula $\texttt{Feature}_1(x, +y_1) \wedge \texttt{Feature}_2(x, +y_2) \Rightarrow \texttt{Class}(x, +c)$. Let us assume that the domain of y_1 denoted as Δy_1 is equal to $\{F_{11}, F_{12}, F_{13}, F_{14}\}$ and $\Delta y_2 = \{F_{21}, F_{22}, F_{23}, F_{24}\}$.

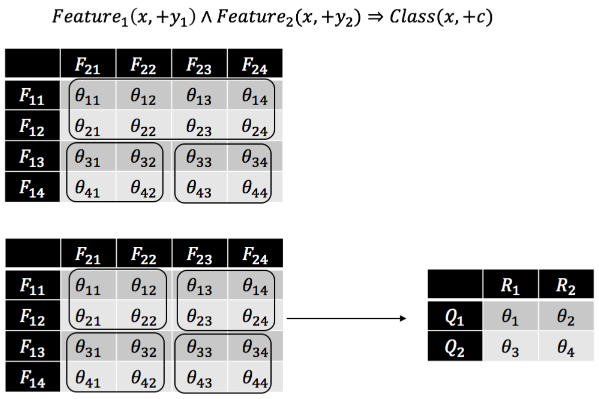
$Feature_1(x, +y_1) \wedge Feature_2(x, +y_2) \Rightarrow Class(x, +c)$
F_{21} F_{22} F_{23} F_{24}
F_{11} θ_{11} θ_{12} θ_{13} θ_{14}
F_{12} θ_{21} θ_{22} θ_{23} θ_{24}
F_{13} θ_{31} θ_{32} θ_{33} θ_{34}
F_{14} θ_{41} θ_{42} θ_{43} θ_{44}
R_1 R_2
Q_1 θ_1 θ_2
Q_2 θ_3 θ_4

Figure 1: Clustering a + variable formula. The first clustering is inconsistent since the atoms of a specific predicate will be clustered with distinct atoms of that predicate for two different clusters. The second clustering is consistent. The θ values specify the initialization weights for the formulas. in the clustered formula, the weight is the average over all weights in that cluster. e.g. $\theta_1 = \frac{\theta_{11}+\theta_{12}+\theta_{21}+\theta_{22}}{4}$.

Thus, we have $16 \times |\Delta C|$ formulas. Two different clusterings for this formula are shown in Fig. 1. The first clustering is inconsistent, since atoms in two different clustered formulas will overlap. For example, $\texttt{Feature}_1(x, F_{21})$ will be clustered along with $\texttt{Feature}_1(x, F_{22})$, $\texttt{Feature}_1(x, F_{23})$ and $\texttt{Feature}_1(x, F_{24})$ in one cluster, and with only $\texttt{Feature}_1(x, F_{22})$ in a separate cluster. In contrast, the second clustering is consistent. Specifically, along every dimension, the clustered atoms can mapped to a unique cluster.

To achieve a consistent clustering, we model the problem of learning a consistent clustering as a joint clustering problem over all the + variables in the untied formula. Note that we do not consider clustering the + variables corresponding to query predicates. This is because, the + variables in query predicates correspond to classes, and we want our final model to be able to discriminate between all classes, which is not possible if we cluster the classes.

The joint clustering problem is modeled as follows. Given formula f, let $\mathcal{X}_+$ be the + variables to be jointly clustered. Let $\bar{x}_+$ be a specific assignment to $\mathcal{X}_+$, and let $\theta_{\bar{x}_+}$ be the average initialization weight of groundings of f consistent with $\bar{x}_+$. We arrange $\theta_{\bar{x}_+}$ as a tensor, where each of the + variables represent a single order or dimension of the tensor. Our task is to now find a quantizer that reduces the order of this tensor across all dimensions. Specifically, the quantizer partitions or clusters each dimension of the tensor, to obtain a lower-order tensor. Thus, given the desired number of clusters for each domain corresponding to the + variables, $k_1, \ldots k_m$, the quantizer will jointly partition the domains in $\mathcal{X}_+$.

Formally, let $\mathbf{A}$ be a m-order tensor $\mathbb{R}^{n_1 \times n_2 \times \ldots \times n_m}$ that represents the weights to each possible grounding of $\mathcal{X}_+$. We want to find a quantizer $\mathcal{Q}$ that maps $\mathbf{A}$ to a reduced tensor $\mathbf{B}$, $\mathbb{R}^{k_1 \times k_2 \times \ldots \times k_m}$, that minimizes the Euclidean distance between the cluster-centers in $\mathbf{B}$, and the weights of their corresponding cluster elements. Specifically,

$$\min_{\mathcal{Q}} \sum_{a \in \mathbf{A}} ||a - M_{\mathcal{Q}}(a)||_2^2 \tag{3}$$

where $M_{\mathcal{Q}}(a)$ is the cluster center of a under the mapping $\mathcal{Q}$. Note that directly optimizing Eq. (3) is a hard problem. Therefore, we use approximate methods that use a co-ordinate descent like procedure and perform dimension-wise clustering as described in [20, 10].

From the reduced tensor for an untied formula f, we reduce the number of possible groundings of f, by utilizing the clusters along each dimension of the tensor. Specifically, given the original tensor $\mathbb{R}^{n_1 \times n_2 \times \ldots \times n_m}$ and the reduced tensor $\mathbb{R}^{k_1 \times k_2 \times \ldots \times k_m}$ induced by the quantizer $\mathcal{Q}$, note that we have two choices. We can generate k_i variables for each dimension, such that the domain of each variable is equal to the objects in the partition specified by $\mathcal{Q}$, and the weight is given by the cluster center. However, though this reduces the number of weights from $n_1 \times n_2 \times \ldots \times n_m$ to $k_1 \times k_2 \times \ldots \times k_m$, the total number of possible groundings of f on the + variables remains $n_1 \times n_2 \times \ldots \times n_m$. This is problematic especially, since we are considering high-dimensional formulas, the large number of possible groundings makes relational learning infeasible. Therefore, we reduce the number of weights and number of ground formulas at the same time. Specifically, we replace each partition along a dimension with a constant instead of a variable. It then follows that the total number of weights and the number of possible groundings of the + variables, reduces from $n_1 \times n_2 \times \ldots \times n_m$ to $k_1 \times k_2 \times \ldots \times k_m$.

Once we obtain the modified MLN, we perform discriminative relational learning using existing approaches such as voted perceptron and contrastive divergence on the modified MLN to re-learn the weights of all the formulas in the MLN. We can think of this re-learning process as jointly optimizing the weights learned for i.i.d instances. For example, assume that for a given dataset, our non-relational learner gives us ideal weights for $p\%$ of the instances, and imperfect weights for the remaining $(n-p)\%$ instances. Joint learning will now relate the i.i.d instances through the MLN formulas, and jointly optimize their weights to maximize to CLL. Naturally, as p increases, the non-relational learner will influence the relational learner to correct the $n-p$ remaining weights, and as p reduces, the relational learner will use dependencies to correct the weights given by the non-relational learner. Note that in our final learned model, clusters of objects in the domain of + variables will be replaced by constants. Therefore, at testing time, i.e., when we perform inference on this MLN using a test database as evidence, we replace + domain objects in the evidence atoms with symbols that represent their clusters.

3.3 Tying Related Formulas

So far, we considered untied formulas independently. However, when atoms are shared across untied formulas, the clustering over one formula affects the other. Specifically, let f and f' be two untied formulas with shared atoms. Clustering the groundings of f affects the formulas in f'. This is similar to the *shattering* process seen frequently in lifted inference [17, 7]. In shattering, grounding the domain of a variable in one formula needs to be propagated to other formulas with a shared domain. In our case, the tying of formulas needs to be propagated to other formulas with one or more shared atoms. Thus, once we cluster a formula, we propagate the clustering throughout the MLN before clustering the next formula. Propagating the tied formulas means that we replace partitions of the + variable domains with constants. Note that in doing so we remove + variables from an untied formula and replace it with constant symbols.

Algorithm 1 summarizes our approach. The first step is to train a classifier for each untied formula using a non-relational model, using features corresponding to groundings of the +-variables in the untied formula. For this, we create independent instances corresponding to each grounding of the non-+ variables to create the training data, and learn the parameters of the non-relational model. We then initialize the weights for an untied formula from the learned parameters, and perform clustering to obtain clusters of ground formulas corresponding to the untied formula that have similar weights in the non-relational model. We replace each of these clusters by a single ground formula with weight initialized to be the mean weight of the cluster. We also change the dataset to reflect by replacing objects with their corresponding cluster symbols. We then propagate the changes (from objects to cluster symbols) to other formulas in the MLN that have shared atoms.

3.4 Semi-Formal Analysis

Let $\mathbf{w}$ be the weight vector that we would learn if we used the original MLN with untied weights, i.e., we learn a separate weight for every untied formula. Let $\mathbf{w}'$ be the weight vector that we will learn, if we tie the weights of untied formulas with a quantizer $\mathcal{Q}$. We assume that for all the untied formulas grouped together by $\mathcal{Q}$, the difference between the weights that would be learned before formula tying and after formula tying are bounded by ϵ. Specifically, $|w - \mathcal{Q}(w)| \leq \epsilon$, where $w \in \mathbf{w}$ and $\mathcal{Q}(w) \in \mathbf{w}'$ represents the weight of the group to which w belongs as given by $\mathcal{Q}$. Let $\ell(\mathbf{w}, \mathcal{D})$ represent the likelihood when we learn $\mathbf{w}$ and $\ell(\mathbf{w}', \mathcal{D})$, the likelihood when we learn $\mathbf{w}'$ from $\mathcal{D}$. We can show the following result, similar to the result shown in Chou et al. [3].

Theorem 1. *$\ell(\boldsymbol{w} : \mathcal{D}) - \hat{\ell}(\boldsymbol{w}' : \mathcal{D}) \leq 2\epsilon M$, where M is the number of groundings in the MLN.*

Proof.

$$\begin{aligned}
\ell(\boldsymbol{w} : \mathcal{D}) &= \sum_i w_i N_i(\mathcal{D}) - \log(\sum_{\mathcal{D}'} \exp(\sum_i w_i N_i(\mathcal{D}'))) \\
&\leq \sum_i (\mathcal{Q}(w_i) + \epsilon)(N_i(\mathcal{D})) \\
&\quad - \log(\sum_{\mathcal{D}'} \exp(\sum_i (\mathcal{Q}(w_i) - \epsilon) \\
&\quad (N_i(\mathcal{D}')))) \\
&= \sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}) + \sum_i \epsilon N_i(\mathcal{D}) \\
&\quad - \log(\sum_{\mathcal{D}'} \exp(\sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}') \\
&\quad \exp(-\sum_i \epsilon N_i(\mathcal{D}')))) \\
&\leq \sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}) + \epsilon M \\
&\quad - \log(\sum_{\mathcal{D}'} \exp(\sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}') \\
&\quad \exp(-\epsilon M))) \\
&= 2\epsilon M + \sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}) \\
&\quad - \log(\sum_{\mathcal{D}'} \exp(\sum_i \mathcal{Q}(w_i) N_i(\mathcal{D}') \\
&= \hat{\ell}(\boldsymbol{w}' : \mathcal{D}) + 2\epsilon M
\end{aligned}$$

□

Thus, from Theorem 1's result, reducing ϵ will reduce the likelihood difference between the two models. Intuitively, to reduce ϵ, we need to predict which of the united formulas will end up with the same weights. Our approach of using a non-relational classifier performs exactly this prediction by considering instances independently. Essentially, we are decomposing the relational database $\mathcal{D}$ into a set of i.i.d instances D, and our non-relational model learns similarity of weights using D. Our assumption is that these similarities in weights will continue to hold when we relate the i.i.d instances in D through $\mathcal{D}$, thus reducing ϵ. Similarly, having smaller number of formulas will reduce the difference in likelihood, but at the same time, it can increase the error ϵ.

Algorithm 1: Formula Tying

```
Input: MLN structure $\mathcal{M}$, Training data $\mathcal{D}$, Query $\mathbf{Q}$, Non-relational learner $\mathcal{R}$
Output: $\mathcal{M}'$, $\mathcal{D}'$
1  Let $f_1 \dots f_n$ be the untied formulas in $\mathcal{M}$
2  for each $f_i$ do
       // Construct a classifier for $f_i$
3      Encode each grounding of non-query +-variables in $f_i$ as a feature for $\mathcal{R}$
4      for each grounding of non-+ variables in $f_i$ do
5          $X_i$ = $i$-th grounding of non-+ variables in $f_i$
6          $y_i$ = Class of query corresponding to $X_i$
7          $X = X \cup X_i$
8          $y = y \cup X_i$
       // Perform non-relational learning
9      $\Theta$ = Parameters of $\mathcal{R}$ learned from $(X, y)$
       // Weight initialization for relational learner
10     Initialize $f_i$ with $\Theta$
11     $\mathcal{X}_+$ = + variables in $f_i$
12     $\mathcal{P}^{(1)} \dots \mathcal{P}^{(m)}$ = Partitions of domains of $\mathcal{X}_+$
13     for $e \in \mathcal{P}^{(1)} \times \dots \times \mathcal{P}^{(m)}$ do
14         $\theta$ = Cluster center for $e$
15         $\mathbf{C}$ = Map each partition in $e$ to a constant
           // Replace clustered formulas with a single formula
16         $(f',\theta)$ = Replace + variables in $f_i$ with $\mathbf{C}$
17         Add $(f',\theta)$ to $\mathcal{M}'$
           // Propagate changes to training data and the other formulas
18         $\mathcal{D}'$ = Replace objects in $e$ with $\mathbf{C}$ in $\mathcal{D}$
19         $\mathcal{M}'$ = Propagate $C$ to $f_{i+1} \dots f_n$
20 Copy remaining formulas from $\mathcal{M}$ to $\mathcal{M}'$
21 return $(\mathcal{M}',\mathcal{D}')$
```

Weight Initialization. Due to convexity of the CLL function, irrespective of the starting states, in theory, max-likelihood learning will converge to a globally optimal solution given enough iterations. However, the main problem is that performing exact max-likelihood learning is a hard problem. Specifically, the gradient is given by,

$$\frac{\partial}{\partial w_i}\ell(\boldsymbol{w} : \mathcal{D}) = N_i(\mathcal{D}) - \mathbb{E}_{\boldsymbol{w}}[N_i(\mathcal{D})]$$

Computing $\mathbb{E}_{\boldsymbol{w}}[N_i(\mathcal{D})]$, is typically infeasible since it requires exact inference on the MLN. Therefore, relational weight learners approximate the gradient by computing $\mathbb{E}_{\boldsymbol{w}}[N_i(\mathcal{D})]$ approximately. This approximation means that we cannot guarantee convergence to the globally optimal $\boldsymbol{w}$, and depending on the initialization, the weights may converge to different local minima. Good weight initialization is well-known to be effective in avoiding local minima [23], especially when dealing with a large number of parameters, therefore, our weight initialization will typically be effective in high-dimensional untied formulas.

4 Experiments

4.1 Setup

We evaluated our approach on three real-world applications modeled using MLNs, namely, collective classification of web-page topics [13] (WebKb), entity resolution [21] (ER), and information extraction [18] (IE). For ER and IE, we used the Cora dataset, that consists of 1295 citations of 132 different papers. For WebKb, the dataset consists of 4165 web-pages from 4 different universities. The MLN structure and the data for each of these applications are publicly available in Alchemy [12]. Note that, we chose these MLNs since they are some of the most popular benchmarks for evaluating MLN inference and learning algorithms.

We conducted our experiments on 8GB quad-core machines. We evaluated the performance of our approach regarding accuracy and running time. As our non-relational learner, we used the multiclass SVM implementation in scikit-sklearn. For the relational learner, we used the contrastive divergence implementation in Tuffy [16]. Tuffy is arguably the leading general-purpose learning and inference system for MLNs. We compared our approach (SVMTied) with three other approaches, SVM classifier (SVM), Tuffy with the original MLN with untied weights as is (OrigTuffy), using tied weights with random initial weights (RandomizedTuffy), and evaluated accuracy using cross-validated F1-score.

4.2 Results

Our first application predicts topics in web-pages using the WebKB MLN. The untied formula relates words in the document to topics. There is only one query predicate in this MLN, which encodes the seven classes/topics of web-pages. We tried to use Tuffy to learn the MLN with untied formulas, but Tuffy did not work on the original MLN that contains + variable formulas, and timed out after 24 hours. The results for the remaining systems are shown in Table 1. As shown here, the weight initialization plays an essential role in improving performance. When we cluster the untied formulas based on random weights, we obtain performance that is worse than using SVMs, which of course ignores relational dependencies. We used the same number of clusters (80 clusters) in both RandomizedTuffy and our approach. Our approach significantly outperforms the other approaches on this application. Lowd and Domingos [13] obtained an AUC score of around 0.8 for this dataset learning a separate weight for each formula. This illustrates that tying could help improve generalization in some cases, where learning too many weights could overfit the model.

SVM	**OrigTuffy**	**RandomizedTuffy**	**SVMTied**
0.42	X	0.31	**0.84**

Table 1: F1-Score comparison for WebKB.

Query	**SVM**	**OrigTuffy**	**RandomizedTuffy**	**SVMTied**
SameAuthor	0.71	X	0.33	**0.92**
SameTitle	0.63	X	0.54	**0.85**
SameVenue	0.68	X	0.32	**0.79**
SameBib	0.51	X	0.29	**0.78**

Table 2: F1-Score comparison for ER.

The second application performs entity resolution (ER) using the CORA dataset. Specifically, the idea is to predict if fields belong to the same author, venue, title, and finally if two citations match. Thus, there are multiple query predicates in the MLN for this case. The untied formulas use word-based features to predict the queries. Once again, OrigTuffy failed to run on this problem, and we timed out after 24 hours. For both RandomizedTuffy and SVMTied, we used 40 clusters. Our proposed approach once again significantly outperformed the other methods on all queries for this application as shown in Table 2. SVMs outperformed using random weight tying, with random weight initialization. Singla and Domingos [21] obtained an AUC score of around 0.91 for SameAuthor, 0.90 for same venue and 0.99 for SameBib. However, Singla and Domingos use domain-specific methods to make the approach more scalable. For example, they apply McCallum et al.'s [14] canopy approach with TF-IDF to reduce the number of plausible pairs. It is sometime hard to generalize these methods to other problems, and therefore, in our approach, we did not apply these methods. In future, we will explore systematic ways to incorporate such domain-specific knowledge into MLN weight learning in order to improve accuracy.

Query	SVM	OrigTuffy	RandomizedTuffy	SVMTied
InfieldAuthor	0.65	X	0.36	**0.79**
InfieldTitle	0.45	X	0.4	**0.68**
InfieldVenue	0.52	X	0.42	**0.71**
SameCitation	0.48	X	0.28	**0.72**

Table 3: F1-Score comparison for IE.

Our final application performs joint segmentation and citation matching. We use the Information Extraction (IE) MLN and the citeseer dataset for this application. The untied formulas relate word based features with predicting segments of text, and the segmented text is used in predicting if two citations are the same. The segmented fields can be of type author, tile or venue. As shown in Table 3, on all the query variables, our proposed method using 35 clusters significantly outperforms the other approaches. The best published result on this dataset is by Poon and Domingos [18]. Specifically, using the joint segmentation (without incorporating entity resolution feedback), Poon and Domingos obtained an F1-score of nearly 94%. However, the MLN we used is a simpler version (also specified in the Alchemy website), since existential quantification that is needed by the Poon and Domingos MLN is not well-supported in current relational learners. Note that, Poon and Domingos mention in their work that they have modified the Alchemy system to run their MLN, which was not publicly available to us.

Varying Number of Clusters. Increasing the number of clusters reduces the quantization error. We verify whether reducing the quantization error results in improved performance in terms of classification accuracy. Specifically, we learn our models with a varying number of clusters. Fig. 2 shows our results where the average F1 score over all queries is plotted against the number of clusters for each of our applications. As can be seen by our results, increasing the number of clusters typically improves performance over all datasets. However, it should also be noted that learning becomes harder as we increase the number of clusters, and the weights learned may not be optimal. Thus, in some cases, we see the performance not improving with an increase in clusters, or in some cases, even degrading by a little. Choosing the optimal number of clusters is something we will consider in future work.

Running Time. We compare the running times of the various learning methods in Table 4. As expected, SVMs take very little time to train as compared to relational learners. Note that SVMTied is significantly faster than RandomizedTuffy in terms of training times. This illustrates that proper weight initialization can significantly help speed up convergence of the relational learner.

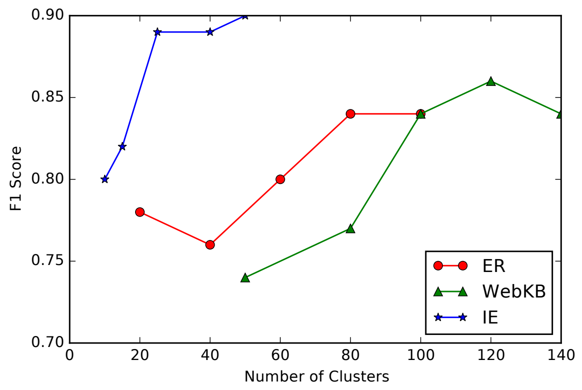

Figure 2: Performance of our approach as we vary number of clusters

Application	SVM	OrigTuffy	RandomizedTuffy	SVMTied
WebKB	26	X	200	127
ER	18	X	324	199
IE	20	X	303	145

Table 4: Running time (in minutes) comparison.

5 Conclusion

In this paper, we proposed an efficient weight learning approach for MLNs that have untied formulas with a large number of groundings. Specifically, we tied together those groundings that are likely to have similar weights by learning a non-relational classifier that uses the groundings of untied formulas as features. We assumed that similarity of parameters in the non-relational model for groundings of the untied formulas will translate to similarity of weights in the relational model. Further, using the parameters learned from the non-relational model, we initialized weights for relational learning, which as we showed empirically yields much more accurate results. Our experiments on multiple applications showed that our approach significantly improves accuracy and scalability of learning, as compared to learning the untied formulas as is.

Future work includes automatically inferring the optimal number of parameters required to learn a dataset, learning the relational model jointly with the non-relational model, etc.

Acknowledgements

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

References

[1] Babak Ahmadi, Kristian Kersting, Martin Mladenov, and Sriraam Natarajan. Exploiting Symmetries for Scaling Loopy Belief Propagation and

Relational Training. *Machine Learning*, 92(1):91–132, 2013.

[2] J. Besag. Statistical Analysis of Non-Lattice Data. *The Statistician*, 24:179–195, 1975.

[3] Li Chou, Somdeb Sarkhel, Nicholas Ruozzi, and Vibhav Gogate. On Parameter Tying by Quantization. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 3241–3247. AAAI Press, 2016.

[4] M. Collins. 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*, pages 1–8, Philadelphia, PA, 2002. ACL.

[5] Mark Craven and Seán Slattery. Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, 43(1/2):97–119, 2001.

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

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

[8] Jan Van Haaren, Guy Van den Broeck, Wannes Meert, and Jesse Davis. Lifted Generative Learning of Markov Logic Networks. *Machine Learning*, 103(1):27–55, 2016.

[9] Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[10] Stefanie Jegelka, Suvrit Sra, and Arindam Banerjee. Approximation Algorithms for Tensor Clustering. In *Algorithmic Learning Theory, 20th International Conference, ALT*, pages 368–383, 2009.

[11] Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. Exploring Markov Logic Networks for Question Answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 685–694, 2015.

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

[13] D. Lowd and P. Domingos. Efficient Weight Learning for Markov Logic Networks. In *Principles of Knowledge Discovery in Databases*, pages 200–211, Warsaw, Poland, 2007.

[14] A. McCallum, K. Nigam, and L. Ungar. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[15] Happy Mittal, Vibhav Gogate, Shubhankar Suman Singh, and Parag Singla. Fine Grained Weight Learning in Markov Logic Networks. In *Workshop on Statstical Relational AI*, 2016.

[16] Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

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

[18] H. Poon and P. Domingos. Joint Inference in Information Extraction. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 913–918, Vancouver, Canada, 2007. AAAI Press.

[19] Somdeb Sarkhel, Deepak Venugopal, Tuan Anh Pham, Parag Singla, and Vibhav Gogate. 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.*, pages 1067–1073, 2016.

[20] Somdeb Sarkhel, Deepak Venugopal, Nicholas Ruozzi, and Vibhav Gogate. Efficient Inference for Untied MLNs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4617–4624, 2017.

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

[22] Deepak Venugopal, Chen Chen, Vibhav Gogate, and Vincent Ng. 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)*, pages 831–843. ACL, 2014.

[23] Lodewyk F. A. Wessels and Etienne Barnard. Avoiding False Local Minima by Proper Initialization of Connections. *IEEE Trans. Neural Networks*, 3(6):899–905, 1992.