Fine Grained Weight Learning in Markov Logic Networks

Happy Mittal, Shubhankar Suman Singh
Dept. of Comp. Sci. & Engg.
I.I.T. Delhi, Hauz Khas
New Delhi, 110016, India
happy.mittal@cse.iitd.ac.in,
shubhankar039@gmail.com

Vibhav Gogate
Dept. of Comp. Sci.
Univ. of Texas Dallas
Richardson, TX 75080, USA
vgogate@hlt.utdallas.edu

Parag Singla
Dept. of Comp. Sci. & Engg.
I.I.T. Delhi, Hauz Khas
New Delhi, 110016, India
parags@cse.iitd.ac.in

Abstract

Markov logic networks (MLNs) represent the underlying domain using a set of weighted first-order formulas and have been successfully applied to a variety of real world problems. A parameter tying assumption is made, i.e., ground formulas coming from the same first-order logic formula have identical weights. This assumption may be inaccurate in many scenarios and can lead to high bias during learning. On the other extreme, one could learn a different weight for each grounding resulting in a model with high variance. In this paper, we present a principled approach to exploit this trade-off by modeling each constant coming from a hidden subtype, and tying the parameters only for those formula groundings which have the same subtype(s) for the respective arguments. We propose two different approaches for automatically discovering the subtypes and learning the parameters of the model 1) a K-means clustering based approach b) a joint learning approach using an EM based formulation. The two extremes described above fall out as a special case of our formulation. Preliminary experiments on a benchmark MLN show that our algorithm can learn significantly better parameters compared to available alternatives.

Introduction

Several real world domains such as those in NLP, biology and vision need to represent the underlying uncertainty as well as represent the relational structure. Statistical relational models (Getoor and Taskar 2007) achieve this by combining the power of logical representations with statistical models. One such powerful model is Markov logic (Richardson and Domingos 2006) which represents the domain as a set of weighted first-order formulas and can be seen as generating templates for generating ground Markov networks. Despite their popularity, there has been somewhat limited focus on learning the parameters in MLNs, and most existing algorithms are variants of the early work proposed by Singla and Domingos (2005) and Lowd and Domingos (2007). Huynh and Mooney (2009) have looked at learning the MLN parameters in a max-margin framework and then extended this further to an online max-margin setting (Huynh and Mooney 2011). Haaren et al. (2015) have exploited the symmetry in the model to come up with lifted parameter learning algorithm in MLNs.

Existing literature for learning the weights makes the parameter tying assumption, i.e., all groundings of a first-order logic formula will have the same weight in the ground Markov network. Though this assumption is important for generalizing the model to future domains, it can lead to high bias since all the groundings of a first-order logic formula may not represent the same statistical regularity. On the other extreme, a different weight can be learned for each formula grounding resulting in a classical Markov network parameter learning problem. In addition to leading to a high variance model, this alternative approach also has the problem that it can not generalize to new domains containing unseen constants.

In this paper, we propose a principled approach for exploiting this trade-off between bias and variance by learning identical weights for a subset of formula groundings. We model the problem by having each constant in the domain coming from one of the subtypes. Formula groundings associated with the same subtype signature are forced to obey the parameter tying (but not otherwise). We propose two different approaches for automatically discovering the subtypes from the data and learning the parameters of our model. In the first approach, we perform k-means clustering over constants belonging to each type. The feature vector for each constant is composed of the ground formula weights in which the constant appears, where the formula weights have been learned with untied set of parameters. The constants in each cluster are modeled as belonging to the same subtype. Once the subtypes have been discovered, the parameters are relearned by tying the parameters over the discovered subtypes. In the second and more principled solution, each constant subtype is modeled as a hidden variable in the network. We formulate a joint learning problem where we discover the hidden subtypes as well as learn the parameters (with tying enforced across the subtypes) using the EM algorithm. In the latter case, the k-means clustering is used as an initialization to the EM algorithm.

The two extremes described earlier fall out as special cases of our learning formulation i.e., one with a single subtype and the other with a separate subtype for every constant. The number of subtypes to be discovered depends on the domain (and the amount of training data available) and can be determined empirically. Our algorithm can generalize to unseen constants since we learn the parameters for subtypes and not for individual constants. These subtypes are inferred in a joint fashion (along with the query variables) during the inference procedure. Preliminary experiments on a benchmark MLN show that our approach can learn significantly better parameters compared to available alternatives.

Our approach is motivated by the recent work of Chou et al. (2016) for learning the parameters of a Markov network while making the parameter tying assumption. Their setting is purely propositional and can not generalize to new domain. Our subtypes can be seen as giving a relational framework for deciding which formula groundings should be tied to each other, and helping us generalize to unseen constants. Further, unlike us, they do not have any EM based formulation for joint learning.

Our approach also bears similarity to existing structure learning approaches for MLNs (Kok and Domingos 2007; Kok and Domingos 2009; Kok and Domingos 2010) where one of the side effects of learning is an implicit clustering of the underlying constants. Nevertheless, the end objective in all these algorithms is to learn the structure of the MLN which can be a significantly more expensive task. In our case, we would like to restrict our focus to the problem of refining the granularity of the learned weights at the level of subtypes, while the formulas are already provided. Our proposed approach for weight learning is tailored to this task and therefore, a direct comparison with structure learning approaches may not be in line.

We first present some background on Markov logic and weight learning. We then propose our model for defining an MLN distribution in presence of subtypes. We present the two approaches for discovering the subtypes and learning the parameters. We then describe how to use the learned model at the prediction time in a joint setting. We present our preliminary experimental evaluation. We conclude with directions for future work.

Background

Markov Logic Network (MLN) is a set of pairs (f_i, w_i), where f_i is a first order formula, and w_i is the weight of the corresponding formula. w_i indicates how likely it is that the formula f_i is true in the world. Larger the weight, larger is the probability of formula being true. In the extreme case, a weight can be infinite, which means a formula is true for every object i.e. the formula becomes pure first order logic formula. An MLN acts as a template for creating a Markov Network, in which each ground atom becomes a node, and each ground formula defines a feature of the model. Given an assignment on a set of ground atoms X (evidence), probability distribution on the assignments of the remaining ground atoms Y (query) is given by

$$P(Y = y|X = x; w) = \frac{1}{Z_x} \exp\left(\sum_{i=1}^{n} w_i n_i(x, y)\right)$$

where $n_i(x, y)$ is the number of satisfied groundings of i^{th} first order formula under the assignment (x, y) and Z_x is the normalization constant. When expressed as a function of the parameters, log of the above probability is referred to as the likelihood function denoted as $LL(w)$. In a slight abuse of notation, we will use the term log-likelihood of the data (given the parameters) to refer to the likelihood function described above.

Weight Learning

Weights of an MLN can be learned either generatively or discriminatively. In generative learning, there is no separate notion of query or evidence atoms. The weights are learned by maximizing the log-likelihood of the entire set of ground atoms. Given an assignment $(X = x)$ to the ground atoms, the gradient of the log-likelihood with respect to a particular weight w_i is given by

$$\frac{\partial}{\partial w_i} LL(w) = n_i(x) - E[n_i(x)]_{P(X)}$$

In other words, the gradient is the difference between actual number of satisfied groundings of i^{th} formula in the data and its expected number of satisfied groundings according to current model. Due to intractability of computing this gradient, Richardson and Domingos (2006) proposed maximizing pseudo log-likelihood (Besag 1986) of the data. Suppose there are n ground atoms in the data, and each ground atom X_l has some truth value x_l, then pseudo log-likelihood of the data is given by

$$\log P^*(X = x) = \sum_{l=1}^{n} \log P(X_l = x_l | MB(X_l))$$

where $MB(X_l)$ is the Markov Blanket of the ground atom X_l in the ground Markov Network. So pseudo log-likelihood of the data is the sum of conditional log-likelihood of all ground atoms given their Markov Blankets. Gradient of pseudo log-likelihood with respect to a particular weight w_i is given by

$$\frac{\partial LL^*(w)}{\partial w_i} = \sum_{l=1}^{n} [n_i(x) - P(X_l = 0|MB_x(X_l)) n_i(x_{[X_l=0]}) - P(X_l = 1|MB_x(X_l)) n_i(x_{[X_l=1]})$$

Here $n_i(x_{[X_l=0]})$ is the number of true groundings of i^{th} formula when X_l is forced to be 0 (and keeping remaining data unchanged). Similarly for $n_i(x_{[X_l=1]})$.

Singla and Domingos (2005) proposed discriminative learning for MLN weights, in which weights are found by maximizing the log-likelihood of the query atoms given the evidence. The gradient of the conditional log-likelihood is given by :

$$\frac{\partial}{\partial w_i} LL(w) = n_i(x, y) - E[n_i(x, y)]_{P(Y|X)}$$

Lowd and Domingos (2007) improved the discriminative learning by proposing several methods, the most effective being preconditioned scaled conjugate gradient, which uses second order information to optimize the conditional log-likelihood.

Representation

Motivation

In the standard MLN setting, the parameters of the groundings of each first order logic formula are tied to each other i.e., all the groundings of a formula are constrained to have the same weight. In many scenarios, this may be a restrictive assumption since different subsets of the groundings may represent statistical regularities with differing strengths. In other words, the assumption of tying all the parameters together may result in a model with high bias. On the other extreme, we may want to learn a different weight for each of the groundings of the formula. The problem then reduces to the standard Markov network learning problem with features having independent (untied) weights. But this requires a large amount of data to learn each weight separately and may result in substantial overfitting. In other words, we get a model with low bias but high variance. Further, the model may not generalize to new domains (with new constants) since the template structure of the model is lost in this setting.

In this paper, we propose a framework to deal with this problem by learning weights which are tied only for a subset of the groundings of the given formula. We propose an approach to automatically partition the formula groundings such that the groundings which fall under the same partition have identical weights and groundings which belong to different partitions are free to have different weights. Our approach can be seen as a principled approach for achieving the right trade-off between bias and variance. To take an example, consider a Friends and Smokers MLN (Domingos and Lowd 2009) with the formula $Smokes(x) \rightarrow Cancer(x)$. Now, under the standard setting, we learn a single weight for this formula. On the other hand, the weight of this constraint may depend on some underlying attributes such as ethnicity. These parameters may or may not be explicitly specified in the model. Then, ideally we would like to a) automatically discover these hidden attributes b) learn a different weight for each attribute, which is exactly what our framework can achieve. Next, we present our proposed solution in detail.

Incorporating SubTypes

In our framework, a natural partition over the formula groundings can be achieved by partitioning the underlying set of constants into separate subtypes. Intuitively, all the formula groundings with the same subtype signature would then get the same weight. Next, we formalize the notion of subtyping in an MLN. Consider an MLN M consisting of set of pairs (f_i, w_i). For every constant c of some type t in M, we associate a subtype with it, given by the function $sub(c)$. Intuitively, subtype of a constant indicates the hidden attribute of that constant. In our smoking example, it corresponds to hidden ethnicity of each person. This subtype $sub(c)$ of constant c can take any one of the subtypes specified by the set $SubTypeSet(t) = \{s_t^1, s_t^2, \ldots, s_t^k\}$, where $SubTypeSet(t)$ is a function specifying all the subtypes a constant of type t can have.

Once we have subtype information about each constant, we can use that information to tie the appropriate ground formulas. For this, we define **subtype signature** of a ground formula. Let (f, w) be a first order formula f with weight w. Let $\langle v_1, v_2, \ldots, v_l\rangle$ be an l-tuple of variables appearing in f. Let $\langle t_1, t_2, \ldots, t_l\rangle$ be their corresponding types. Now consider any grounding f_g of f. Let $\langle c_1, c_2, \ldots, c_l\rangle$ be an l-tuple of constants appearing in f_g i.e. instantiations of the variables $\langle v_1, v_2, \ldots, v_l\rangle$. Then subtype signature of f_g is an l-tuple $\langle sub(c_1), sub(c_2), \ldots, sub(c_l)\rangle$. We tie weights of all ground formulas which have same subtype signature. In other words, we partition the set of ground formulas of a first order formula such that in each partition, all ground formulas have same subtype signature. Possible number of such partitions is the number of different subtype signatures a ground formula can have. We define **granularity** g_f of a formula f as the number of possible partitions of that formula, which is given by $g_f = \prod_{j=1}^{l} |SubTypeSet(t_j)|$. So the granularity g_f of a formula f tells the number of partitions of that formula, and for each partition, we have to learn a separate weight. For each formula f in the MLN, we have to learn g_f different weights $\{w^1, w^2, \ldots, w^{g_f}\}$. Now we describe how to add subtype information in MLN M, and how to partition the first order formulas.

Modifying MLN

In an MLN M, we add subtype information of each constant by creating a predicate $HasSubType^t(subtype, const)$, where t is the type of the constant. A grounding $HasSubType^t(ST, c)$ of this predicate is True if $sub(c) = ST$, otherwise False. Note that a constant c can have exactly one subtype, hence for each constant c, exactly one grounding of the predicate $HasSubType^t(subtype, c)$ is True. Let s denote a truth assignment to all the groundings of the predicates of the form $HasSubType^t(subtype, const)$.

ModifyMLN procedure in Algorithm 1 describes how to modify MLN M to partition each first order formula. For every formula f in M, we create g_f new formulas (one for each partition). Each new formula encodes subtype of every variable appearing in f by conjunction of $HasSubType$ predicates. A $'+'$ in front of variable st denotes per-constant learning. For example, if a formula f has variables $\langle v_1, v_2, \ldots, v_l\rangle$ of types $\langle t_1, t_2, \ldots, t_l\rangle$, then the formula $HasSubType^{t_1}(+st, v_1) \wedge HasSubType^{t_2}(+st, v_2) \wedge \ldots \wedge HasSubType^{t_l}(+st, v_l) \wedge f$ specifies all new formulas of f with all possible subtype configurations of variables i.e. all possible partitions of f. Each new formula is allowed to have a separate weight of its own.

Now given evidence $X = x$, the probability distribution over the modified MLN becomes :

$$P(Y = y, S = s | X = x) = \frac{1}{Z_x} \exp\left(\sum_{i=1}^{n} \sum_{j=1}^{g_{f_i}} w_i^j n_i^j(x, y, s)\right) \quad (1)$$

Here $n_i^j(x, y, s)$ is the the number of satisfied groundings of formula f_i^j under the assignment $X = x, Y = y, S = s$. It is not difficult to see that above formulation results in a partition of the ground formulas where the formulas in each partition element are exactly those which have the same subtype signature.

Algorithm 1 Learning fine grained weights in an MLN

```
**FGLearn(MLN $M$, Data $D$, Type $t$)**
  $M' \leftarrow$ ModifyMLN($M$, $t$)
  $M_{FG} \leftarrow$ LearnWts($M'$)
  return $M_{FG}$

**ModifyMLN(MLN $M$, Type $t$)**
  $M' = \phi$
  **for each** $(f, w)$ in $M$ **do**
    **for each** variable $v$ of type $t$ in $f$ **do**
      $f \leftarrow f \wedge HasSubType^t(+st, v)$
    **end for**
    Add $(f, w)$ to $M'$
  **end for**
  return $M'$
```

Learning Weights

Let us first deal with the case when the subtype information is available in the domain. Then we can find weights of the modified MLN by maximizing the conditional log-likelihood $LL(w)$ of probability distribution given in equation 1. Gradient of this $LL(w)$ with respect to a particular weight w_i^j is given by

$$\frac{\partial LL(w)}{\partial w_i^j} = n_i^j(x, y, s) - E[n_i^j(x, y, s)]_{P(Y,S|X)} \quad (2)$$

We can use standard discriminative weight learning methods(Lowd and Domingos 2007) to find best weights which maximize CLL of data.

For many applications, the subtypes for constants in the domain may be implicit (hidden). In such scenarios, we need to model the subtypes as hidden predicates. Below we describe two different methods to deal with such scenarios.

K-means Clustering

The idea in k-means clustering is to define a feature vector for each constant (with the same type) and then cluster the given constants into a set of k clusters (subtypes) using k-means. Let us assume that our MLN has been standardized apart (Mittal et al. 2014), i.e., the variables are renamed in a manner such that each formula has a set of variables which are disjoint from the variables appearing in the other formulas. We first propose to learn a different weight for each of the formula groundings i.e., we create the ground Markov network where each feature (ground formula) has a different weight and we learn it using the standard MLN weight learning (see Background). We will use the weights learned in this manner for defining the feature vectors as follows.

Consider a constant c with type $t = type(c)$. Let $v_1, v_2, \cdots, v_r$ be the variables in the MLN with type t. Let w_i be the average weight of the ground formulas in which c appears as an instantiation of variable v_i. Then, the feature vector for constant c is defined as the r-tuple $\langle w_1, w_2, \cdots, w_r \rangle$. Intuitively, w_i captures the average strength of the constraints in which constant c appears as an instantiation of the variable v_i. For example, consider an MLN with two formulas $P(x) \rightarrow Q(y)$ and $P(z)$. Let x, z be of type t and let y come from a different type. Then, the feature vector for a constant c of type t is given by the tuple $\langle w_1, w_2 \rangle$ where w_1 is the average weight of the ground formulas of the form $P(c) \rightarrow Q(y)$ (for different substitutions of y) and w_2 is the weight of the ground formula $P(c)$ (there is only one formula with constant c in this case). Once feature vector has been constructed we can apply k-means algorithm over it to get the desired clusters (subtypes).

We should note that our idea bears some similarity to the parameter tying framework of Chou et al. (2016). But there are some important differences. In their case, they are tying the parameters for the features (referred to as formulas below) of a Markov network. Their approach discovers a clustering over the formulas in the model as opposed to a clustering over the constants in our case, which in turn leads to a partition over the formula groundings. Further, they use a single dimensional feature vector for every formula during the k-means clustering (i.e. its weight when the parameters are untied). In our case, we have to deal with an r-sized feature vector where r depends on the number of variables of the given type. More importantly, since we partition the constants into subtypes, we can generalize our approach to new domains by jointly inferring hidden types of new constants (see the Section on Generalization to new Domains). This is not possible in case of Chou et al.

Also, there have been other approaches for clustering the constants in an MLN which could potentially be used for our task. In their structure learning algorithm (Kok and Domingos 2009), authors cluster the constants based on gain in posterior probability of the data using a greedy agglomerative approach. Venugopal et al. (2014) use k-means algorithm similar to ours but use a slightly different criteria for constructing the feature vector based on the number of satisfied groundings each constant is involved in. Broeck and Darwiche (2013) have proposed an approach for approximating binary evidence using low rank boolean matrix factorization and the resulting approximation can be seen as inducing a clustering over constants involved in the evidence. Experimenting with these alternate clustering approaches for our fine-grained learning is a direction for future work.

Joint Learning of SubTypes

Rather than following a pipelined approach as described above, a more principled approach to learn the parameters would be to formulate the problem in a joint learning setting. More specifically, we can treat the subtypes of constants as hidden variables in the data, and then maximize the log likelihood of the data by summing over hidden (subtype) variables. Using equation 1, the conditional likelihood $P(Y = y|X = x)$ can be written as:

$$P(Y = y|X = x) = \sum_s \frac{1}{Z} \exp\left(\sum_{i=1}^{n} \sum_{j=1}^{g_{f_i}} w_i^j n_i^j(x, y, s)\right) \quad (3)$$

This can be optimized using the standard EM formulation. The E-step corresponds to filling in the s (hidden) values under the distribution $P(S = s|Y = y, X = x)$ given the current weight vector w. The conditional distribution $P(S = s|Y = y, X = x)$ can be computed from Equation 1 replacing the normalization constant Z_x by Z_{yx} (since both y and x are given now). M-step corresponds to finding the maximum likelihood parameters of the model using filled in s values. Gradient expression is as described in Equation 2 with the difference that $n_i^j(x, y, s)$ are now replaced by counts computed using values filled in the E-step.

Alternatively, we can try to directly optimize the log-likelihood (log of the expression in Equation 3), denoted by $LL(w)$ using gradient descent. The expression of the gradient of $LL(w)$ with respect to the weight w_i^j can be given as:

$$\frac{\partial LL(w)}{\partial w_i^j} = E[n_i^j]_{P(S|Y,X)} - E[n_i^j]_{P(Y,S|X)}$$

Moving along this gradient bears a very close similarity to the EM based optimization. The first term in the above expression corresponds to the expected counts computed under the distribution $P(S|Y, X)$ and hence, can be seen as the E step of the EM algorithm. The second term represents the expected counts computed under the distribution $P(Y, S|X)$ and is identical to the corresponding term in the gradient based optimization when s values are known (see Equation 2). Hence, gradient descent above is analogous to performing M-step in EM with filled in values. The difference is : In EM, the E step is done once for an entire run of maximization, whereas in the explicit gradient based approach, the E step computation is done for every update of the weight vector w. In principle, we can have a combination of the two algorithms where the E step computation is done after every few updates of the weight vector and exploring this in detail is a direction for future work. Note that either of the algorithms above will result in discovery of the subtypes (treated as soft assignments) as well as simultaneous learning of the parameters. We can initialize the parameters of the model using the k-means clustering approach proposed in the previous section. For more details on the relative merits of the two approaches, see Koller and Friedman (2009).

Our method can be seen as a principled way to find a trade-off between bias and variance. Choosing the right number of subtypes will result in finding the sweet spot. We can also come up with PAC learning bounds for our framework as a function of the number of subtypes used. Extending earlier work on Markov networks in coming up with such bounds (Bradley and Guestrin 2012) is a future direction. It is worthwhile to note that in our joint formulation above, g_f copies of each ground formula (for a first-logic formula f) will be created as we discover the hidden subtypes. This may complicate the underlying inference and learning. We propose to use existing methods such as Approximation by Quantization (Gogate and Domingos 2012) and Structured Message Passing (Gogate and Domingos 2013) to deal with this blow up. Exploring them in detail is a direction for future work.

Generalize to New Domains

Once the weights have been learned, we need to be able to apply the learned model to new domains with unseen constants. This can be achieved in a straightforward manner by simply treating the subtypes of the new constants as hidden and inferring them jointly along with the target variables as done in the E step above. Note that this kind of generalization is not possible in the parameter tying approach of Chou et al. (2016) since the parameter learning is done for the propositional network. This ability to generalize is an important advantage of our MLN based approach over earlier work.

Experiments

To demonstrate efficacy of our approach, we performed some preliminary experiments on the IMDB dataset.[1]. It contains information about movies, actors, directors, and who worked under whom. The actors and directors belong to the type person, and the movies have their own type. The dataset contains information about 20 movies and 278 persons distributed across five different databases. [2] Table 1 lists all the rules we have in our MLN. Our inference task here was to predict whether a person is an actor or director, whether he/she has acted in a particular movie, and whether he/she has worked under some other person. We randomly set a subset of the ground atoms as evidence. We used Alchemy (Kok et al. 2008) system for learning and doing inference. All our experiments were run on 2.20 GHz Xeon(R) E5-2660 v2 server with 40 cores and 128 GB RAM.

Methodology

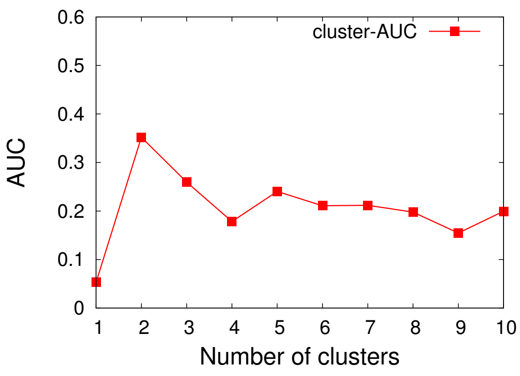
There are two types of constants in this dataset : person and movie. For our experiments, we focused on using subtyping information about constants of type person only. Intuitively, it makes sense because in general, we expect a group of actors to work with a certain group of directors more often than others. Subtyping of persons captures this characteristic of the domain. We found the subtype of each constant by using the k-means algorithm as described earlier. We varied k from 1 to 10. Note that $k = 1$ refers to default MLN setting, in which all ground formulas of a first order formula are tied. We learned the weights by generative learning (which maximizes pseudo log-likelihood of the data). For inference, we randomly selected $r\%$ of evidence from the data, and performed inference with rest of the data as query. We varied r from 10-90%. As an evaluation measure, we calculated area under the precision-recall curve (AUC) of our predicated values. We performed 5-fold cross validation, and averaged out the AUCs. We performed all these experiments on 3 sets of randomly selected evidence to further smoothen out the results.

[1]Dataset available at Alchemy website `https://alchemy.cs.washington.edu/data/`

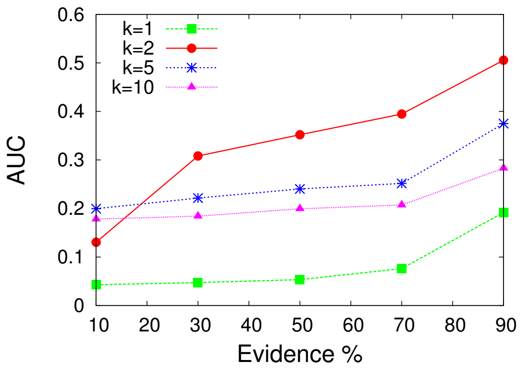
[2]A few persons which appeared in multiple databases had to be removed since the available software implementation crashed when constants were shared across databases. The number given here is after these constants were removed.

$WorkedUnder(p1, p2) \rightarrow Actor(p1)$
$WorkedUnder(p1, p2) \rightarrow Director(p1)$
$Director(p1) \wedge Actor(p2) \wedge Movie(m, p1) \wedge Movie(m, p2) \rightarrow WorkedUnder(p2, p1)$
$Director(p1) \wedge Actor(p2) \wedge Movie(m, p2) \wedge WorkedUnder(p2, p1) \rightarrow Movie(m, p1)$
$Director(p1) \wedge Actor(p2) \wedge Movie(m, p1) \wedge WorkedUnder(p2, p1) \rightarrow Movie(m, p2)$
$Director(p1) \wedge Actor(p2) \rightarrow WorkedUnder(p2, p1)$

Table 1: Rules of IMDB dataset



(a) No. of clusters vs AUC



(b) AUC for different clusters with varying evidence

Figure 1: Results for k-means on IMDB

Results

Figure 1a shows AUCs when k i.e. cluster size was varied from $1 - 10$ with fixed evidence $r = 50\%$. We see that the default setting of MLN in which $k = 1$ has lowest AUC. $k = 2$ achieves highest AUC and after that, as we increase the number of clusters, AUC decreases (but never goes below AUC of $k = 1$). Figure 1b shows AUCs for 4 different cluster sizes ($k = 1, 2, 5$ and 10) when evidence percentage was varied from $10 - 90\%$. It can be clearly seen that $k = 2$ dominates other cluster results for every evidence percentage except 10 where larger number of clusters seem to do somewhat better. Our experiments clearly show that there are actually hidden features of constants through which we can group constants, and tie only the appropriate ground formulas. Learning these hidden features (in form of subtypes) always seems to benefit over the standard MLN formulation. Automatically discovering the right number of clusters (either by cross-validation or using some other technique) is a direction for future exploration.

Conclusion and Future Work

In this paper, we proposed a novel framework for learning the parameters of an MLN where the weights of only a subset of the ground formulas are tied to each other, lying between the two extremes of tying all or none of the ground formula weights. We introduced the notion of the subtype of a constant, which captures the hidden trait of that constant, and enables us to partition the formula groundings based on their type signatures. We proposed two different algorithms for discovering the subtype information (a) k-means based clustering algorithm, (b) a joint EM based formulation. Our preliminary experiments demonstrated the efficacy of our approach.

There are several directions for future work. Current experiments only test the k-means based algorithm for discovering subtypes. We would like to examine how well the EM based joint learning method performs. We would also like to do a more comprehensive experimental study to test the efficacy of our proposed model on a variety of domains. Other directions include coming up with an automatic method for deciding the right number of subtypes and coming up with PAC bounds for our learning framework.

Acknowledgements

Happy Mittal was supported by the TCS Research Scholar Program and the IJCAI travel grant. Vibhav Gogate was partially supported by the DARPA Probabilistic Programming for Advanced Machine Learning Program under AFRL prime contract number FA8750-14-C-0005 and by the NSF award 1528037.

References

[Besag 1986] Besag, J. 1986. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)* 259–302.

[Bradley and Guestrin 2012] Bradley, J. K., and Guestrin, C. 2012. Sample complexity of composite likelihood. In *International Conference on Artificial Intelligence and Statistics*, 136–160.

[Chou et al. 2016] Chou, L.; Sarkhel, S.; Ruozzi, N.; and Gogate, V. 2016. On parameter tying by quantization.

[Domingos and Lowd 2009] Domingos, P., and Lowd, D. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

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

[Gogate and Domingos 2012] Gogate, V., and Domingos, P. 2012. Approximation by quantization. *arXiv preprint arXiv:1202.3723*.

[Gogate and Domingos 2013] Gogate, V., and Domingos,

P. 2013. Structured message passing. *arXiv preprint arXiv:1309.6832*.

[Huynh and Mooney 2009] Huynh, T. N., and Mooney, R. J. 2009. Max-margin weight learning for Markov logic networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer. 564–579.

[Huynh and Mooney 2011] Huynh, T. N., and Mooney, R. J. 2011. Online max-margin weight learning for Markov logic networks. In *SDM*, 642–651. SIAM.

[Kok and Domingos 2007] Kok, S., and Domingos, P. 2007. Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, 433–440. ACM.

[Kok and Domingos 2009] Kok, S., and Domingos, P. 2009. Learning Markov logic network structure via hypergraph lifting. In *Proceedings of the 26th annual international conference on machine learning*, 505–512. ACM.

[Kok and Domingos 2010] Kok, S., and Domingos, P. 2010. Learning Markov logic networks using structural motifs. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 551–558.

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

[Koller and Friedman 2009] Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

[Lowd and Domingos 2007] Lowd, D., and Domingos, P. 2007. Efficient weight learning for Markov logic networks. In *Knowledge discovery in databases: PKDD 2007*. Springer. 200–211.

[Mittal et al. 2014] Mittal, H.; Goyal, P.; Gogate, V.; and Singla, P. 2014. New rules for domain independent lifted MAP inference. In *Proc. of NIPS-14*, 649–657.

[Richardson and Domingos 2006] Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62.

[Singla and Domingos 2005] Singla, P., and Domingos, P. 2005. Discriminative Training of Markov Logic Networks. In *Proc. of AAAI-05*.

[Van den Broeck and Darwiche 2013] 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*, 2868–2876.

[Van Haaren et al. 2015] Van Haaren, J.; Van den Broeck, G.; Meert, W.; and Davis, J. 2015. Lifted generative learning of Markov logic networks. *Machine Learning* 103(1):27–55.

[Venugopal and Gogate 2014] Venugopal, D., and Gogate, V. 2014. Evidence-based clustering for scalable inference in Markov logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 258–273. Springer.