Using the Knowledge of a Domain Expert to Train Markov Logic Networks

Tivadar Papai Shalini Ghosh

September 2, 2011

1 Motivation

A Markov logic network (MLN) [1] defines a probability distribution over truth assignments to ground predicates (atoms). The probability distribution is determined by a set of weighted first-order logic formulae (knowledge base). The weights of these formulae are generally set to maximize the probability of the training data. In certain settings however, we do not have access to sufficient amount of training data. Our aim is to develop a weight learning algorithm when only a small amount of training data is available or when training data is not available at all, relying on the knowledge (subjective probabilities) of a domain expert.

2 Formalization

2.1 Markov Logic and the Exponential Families of Probability Distributions

The probability of a truth assignment (world) x in a Markov logic network is defined as:

$$Pr(X = x) = \frac{\exp(\sum_i w_i n_i(x))}{Z} \tag{1}$$

where $n_i(x)$ is the number of true groundings of the i-th formula, w_i is the weight of the i-th formula and Z is the normalization factor.
The probability distributions defined by Markov logic networks belong to

the exponential families of distributions, and (1) can be rewritten in a more general form:

$$Pr(X = x) = \exp\left(\langle \theta, f(x) \rangle - A(\theta)\right) \tag{2}$$

θ_i are the natural parameters of the distribution, f_i are the features and $A(\theta)$ responsible for the normalization. As we can tell from the comparison of (1) and (2), θ corresponds to w, f_i to n_i and $A(\theta) = \log Z$. In (2) the distribution is parameterized by its natural parameters (θ). It can also be parameterized by its mean parameters, where the means are defined as the expected values of the features:

$$\mu_i = \sum_x f_i(x) Pr(X = x) = \mathbb{E}\left[f_i\right] \tag{3}$$

There is a many-to-one mapping from θ to μ. Let $m(\theta) = \mu$ be this mapping. $m(\theta)$ is one-to-one mapping if the distribution belongs to a minimal exponential family. We have not investigated thoroughly under what conditions (set of weighted formulae) would the distribution defined by a Markov logic network belong to a minimal exponential family. See our justifications in section 2.3 why we neglected the importance of doing this investigation.

2.2 Defining Prior on the Mean Parameter

To our best knowledge in MLNs priors only on the natural parameters (weights) were applied before, typically using a 0 mean Gaussian distribution, expressing the preference that we should have as few non-zero weight features as possible while maximizing the likelihood of the data.

When domain experts are available they can provide subjective probabilities for all or some of the features, i.e., they can give an estimate how likely it is that a randomly chosen grounding of the formula corresponding to the feature is true. E.g., $Pr\left(\text{Smokes}(X) \implies \text{Cancer}(X)\right) = 0.9$ means that for a randomly chosen individual it is true that (s)he either does not smoke or if (s)he does (s)he has lung cancer as well, with 0.9 probability according to the expert. An expert could also have access to statistics which could tell how many percentage of the population smokes.

In absence of training data we would try to match μ normalized by the size of the domain of each predicate to the subjective probabilities provided by the domain expert. In many cases it is impossible to match the subjective probabilities of the expert, because they provide subjective probabilities

which cannot be matched no matter how we choose the weights in the system. E.g., consider the case when according to the expert the probability of $P(x)$ is 0.5, and the probability of $P(x) \vee Q(x)$ is 0.4. It is easy to see that in this situation there is not any weight vector that would provide a matching normalized μ for both subjective probabilities.

Let $\Pi(\mu)$ be the prior on μ. Let $Pr(D|\mu) = Pr(D|\theta) = \prod_i Pr(d_i|\theta)$ be the probability of the i.i.d. training data given θ or μ. Then the log-likelihood of the can be written as:

$$L = \log Pr(D|\mu) + \log \Pi(\mu) \tag{4}$$

The gradient of L w.r.t. θ:

$$\frac{\partial L}{\partial \theta} = \frac{\partial \log Pr(D|\theta)}{\partial \theta} + \frac{\partial \log \Pi(\mu)}{\partial \mu}\frac{\partial \mu}{\partial \theta} = \sum_i (f(d_i) - \mu) + \Sigma_\theta \frac{\partial \log \Pi(\mu)}{\partial \mu} \tag{5}$$

where we used that $\frac{\partial \mu}{\partial \theta} = \Sigma_\theta$. ($\Sigma_\theta$ is the covariance matrix.) The proof of this derivation can be found e.g. in [2].

When we do not have much training data, the concaveness of (4) depends on the choice of $\log \Pi(\mu)$. Even in simple cases, $\log \Pi(\mu)$ is not concave. E.g., consider a one dimensional problem where $\log \Pi(\mu)$ is a triangle shaped function, taking its maximum at $\mu = \mu'$.

$$\frac{\partial^2 \log \Pi(\mu)}{\partial \theta^2} = \frac{\partial \sigma}{\partial \theta}\frac{\partial \log \Pi(\mu)}{\partial \mu} + \sigma \frac{\partial^2 \log \Pi(\mu)}{\partial \mu^2} = \frac{\partial \sigma}{\partial \theta}\frac{\partial \log \Pi(\mu)}{\partial \mu} \tag{6}$$

Since $\frac{\partial \log \Pi(\mu)}{\partial \mu}$ is positive when μ is close to μ' and $\mu > \mu'$, and negative when $\mu < \mu'$, i.e., whether $\log \Pi(\mu)$ w.r.t. θ is concave depends on μ'.

However, if we have sufficient training data available and if Σ_θ is positive definite it can be shown that L is concave by using that $\frac{\partial^2 L}{\partial \theta^2} = -N \cdot \Sigma_\theta + \frac{\partial^2 \log \Pi(\mu)}{\partial \theta^2}$ where N is the number of the training data, since for any (proper size) $v \neq 0$ vector and any (proper size) A matrix:

$$\lim_{N \to +\infty} v^T \left(-N\Sigma + A\right) v = -\infty$$

Even though, L can have more than one local optimum, gradient ascent algorithms will still serve as a basis for our optimization algorithm.

2.3 Choosing the Prior

Our goal is that when no training data is available we would like $\frac{\mu_i}{g_i}$ to be as close to s_i as possible, where g_i is the number of all possible groundings of the i-th formula and s_i is the corresponding subjective probability provided by the expert. A prior that can capture this goal is e.g. a truncated Gaussian distribution of $\frac{\mu_i}{g_i}$ centered around s_i. Let $\overline{\mu}_i = \frac{\mu_i}{g_i}$, then the prior:

$$\Pr(\mu) \propto \exp\left(-(\overline{\mu} - s)^T(\overline{\mu} - s)\right) = \exp\left(-\alpha \sum_i (\overline{\mu}_i - s_i)^2\right) \tag{7}$$

At the first sight it looks tempting that if we could restrict the search space to MLNs which would define probability distributions belonging to a minimal exponential family, so that we could have the inverse function of m (m^{-1}) and could define a prior on θ directly by penalizing it being far from $\theta' = m^{-1}(s_i g_i)$. The problem with this approach is that a small change in μ could result in a large change in θ when μ is either close to 0 or to 1 (i.e., when θ is close to $+\infty$ and $-\infty$). Not to mention the problem of dealing with the problem of inconsistent subjective probabilities.

3 Implementation

The implementation was done in C as part of PCE. The input format of the training data files is specified as follows:

- The first line should contain the number of training data / worlds.
- Whitespace lines are ignored.
- Each world is separated by >> in a new line.
- There cannot be any hidden atoms.
- The truth assignment of every atom in every world has to be specified.
- Every truth assignment must be in the format [∼]predicate_name(ground_arguments) and has to occupy a whole line. ∼ is used if the ground atom's truth value is false.

The input format of the PCE format which specifies the MLN is extended by:

- LEARN formula [probability]; - the same syntax as ADD. The probability is the subjective probability of the formula. If probability is not provided, the formula will not have a subjective probability associated with and its weight is influenced directly by the training data.
- TRAIN path/to/training_data; - starts the training with the provided path to the training data file (absolute paths are recommended).
- Limitations of the current implementation includes not allowing CONST declaration appearing after any LEARN or TRAIN, all LEARNs have to precede TRAIN.

Sample input MLN file:

```
mcsat_params 10000, 0.01, 5.0, 0.01, 100, 10;
sort Dummy;
const a,b,c,d: Dummy;
predicate p(Dummy) hidden;
predicate q(Dummy) hidden;


learn [x] p(x) 0.4;
learn ~p(a) or q(a) 0.7;
learn [x] ~p(x) or q(x) 0.5;

train "subj_prob_test/test.pcein.data";
```

If the last lines are changed to:

```
learn [x] p(x);
learn ~p(a) or q(a);
learn [x] ~p(x) or q(x);

train "subj_prob_test/test.pcein.data";
```

then only the data in the training data file will be used for training. If the last lines are:

```
learn [x] p(x) 0.4;
learn ~p(a) or q(a) 0.7;
learn [x] ~p(x) or q(x) 0.5;

train;
```

then the training only relies on the subjective probabilities of the domain expert.
Sample training data file:

```
2

p(a)
p(b)
p(c)
~p(d)

q(a)
~q(b)
q(c)
~q(d)

>>

p(a)
p(b)
p(c)
~p(d)

~q(a)
~q(b)
q(c)
~q(d)
```

To generate training data files, all we need to do is to add –dumpsamples=path when running mcsat where path is the path to the file where we want to store

the generated samples. The samples generated during the first mcsat call in the input file will be written out.

The implementation contains two different weight training algorithms. The first one is a simple gradient ascent algorithm using (5). The second method is L-BFGS based. The L-BFGS library requires an objective function to work with, and since the computation of the log-likelihood of the training data is intractable we are computing the pseudo log-likelihood and using its gradient instead of the gradient of the log-likelihood. Note that, L-BFGS library has to be available to run PCE. (http://www.chokkan.org/software/liblbfgs/)

The output besides some debug information will contain the learned weights for the rules which will be displayed as clauses.

Known bugs:

- ADD and LEARN cannot appear at the same time
- We cannot have formulas in the input file which cannot be converted into one equivalent clause

4 Experiments

The gradient ascent algorithm for consistent subjective probabilities always converged, and for inconsistent ones certain weights converged to $+\infty$ or $-\infty$. However, the convergence was very slow. Often after 10000 iterations of weight updates the weights were oscillating providing 0.1 magnitude of change in the mean values. Also, when training data was provided the convergence speed further slowed down. L-BFGS did not always converge, many times stopped with an error message that the minimum step size was too small or every step it made during the line search did not bring closer to the optimum. These phenomena can be explained by that we probably ran into a non-convex optimization problem and also by that the inference algorithm provides noisy results, there is a fluctuation in the computed μ, hence we can get different values for the gradient at the same spot. One could avoid this by approximating the gradient by fitting a (quadratic) function during each line search on a few gradient values we get along the line. I experimented with doing a random step when the algorithm got stuck at one point, but it did not help much. Right now, the gradient ascent algorithm is used since it provided better convergence results when both training data and subjective

probabilities were provided. However, we can turn on L-BFGS by changing the following lines in weight_training.h:

```
#define LBFGS_MODE 1
#define USE_PLL 1
```

The second flag is set to use pseudo-log-likelihood, since L-BFGS requires an objective function to work with.

5 Future Work

A more reasonable choice for a prior could be:

$$\Pr(\mu) \propto \exp\left(-(\overline{\mu} - s)^T A(\overline{\mu} - s)\right) = \exp\left(-\sum_i A_{i,i}(\overline{\mu}_i - s_i)^2\right) \quad (8)$$

where A is a diagonal matrix with non-negative elements, $A_{i,i}$ representing the confidence of the expert in the i-th subjective probability. E.g., the expert might have higher confidence in his subjective probability for what percentage of the population smokes than for how likely it is that smoking causes lung cancer. $A_{i,i} = 0$ represents zero confidence, i.e., the weight of the i-th formula will be trained solely from the training data. The way we could implement this feature is straightforward.

Also, since the prior itself would allow more than one w which would maximize (4), a term proportional to w^2 could be added to the log-likelihood. This term would serve the function of a regularizer, keeping the random variables (truth assignments to the individual atoms) as independent as possible by setting 0 weight to as many formulas as possible while keeping log-likelihood at its maximum.

Right now, our stopping criteria is that when μ does not change much in the last 100 iterations the algorithm stops. This is clearly not the best stopping criteria since the computation of μ provides noisy results and can result in either too early or that this criteria will be never met. The early termination will also prevent the network from reaching a sufficient accuracy, i.e., when subjective probabilities are either $> 1 - \epsilon$ or $< \epsilon$, where $\epsilon < 0.01$.

On the long run gradient ascent cannot be used, because of its slowness and the previously mentioned weaknesses. Using a modified version of L-BFGS could be a next step where the noisy gradient computation would be replaced by fitting a quadratic function on a sufficient noisy gradient point during each line search.

Experimenting with priors different from truncated Gaussian such as Beta distribution is also a future goal. Truncated Gaussian was chosen because of the simplicity of computing its gradient.

Since (4) may have non-global optima we have to think about what non-convex optimization tools we could use.

Right now we allow a prior based on the knowledge of one expert, however combining more than one experts knowledge into a prior on μ is also amongst our future goals.

A theoretical analysis is also future goal. It could be done on that how many samples we would need to have probabilistic bounds / guarantees on the usefulness of subjective probabilities. Such analyses would be feasible if we can assume a certain type of noise on the subjective probabilities provided by the expert.

The weight learning algorithm only works now if there are no latent variables during the training, i.e., we have access to the complete truth assignments. Working out a framework for this scenario is a possible future goal.

6 Conclusion

When no training data is available the only way we can do weight learning in an MLN is by using subjective probabilities provided by an expert. When a small set of training data is provided we need to combine the regular weight learning for MLNs with relying on the subjective probabilities provided by the expert. We managed to formalize this problem using Bayesian statistics by putting a prior on the mean parameters of the MLN. Finding the optimal weights in the resulting optimization problem is not straightforward since the problem in not all cases can be reduced to a non-convex optimization problem depending on the prior. Currently, a gradient ascent algorithm is in use for the weight learning which still has the problem of getting stuck in local optima and being slow.

References

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

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