Efficient Weight Learning For Markov Logic Networks

Markov logic, weight learning, scaled conjugate gradient, diagonal Newton

Abstract

Markov logic networks (MLNs) combine Markov networks and finite first-order logic, and are a powerful and increasingly popular representation for statistical relational learning. The state-of-the-art method for discriminative learning of MLN weights is a voted perceptron algorithm, which is essentially gradient descent with an MPE approximation to the expected sufficient statistics (true clause counts). Unfortunately, these can vary widely between clauses, causing the learning problem to be highly ill-conditioned, and making gradient descent very slow. In this paper, we explore several alternatives, from per-weight learning rates to second-order methods. In particular, we focus on two approaches that avoid computing the partition function: diagonal Newton and scaled conjugate gradient. In experiments on standard SRL datasets, we obtain order-of-magnitude speedups, or more accurate models given comparable learning times.

1. Introduction

Statistical relational learning (SRL) focuses on domains where data points are not i.i.d. (independent and identically distributed). It combines ideas from statistical learning and inductive logic programming, and interest in it has grown rapidly in recent years (Fern et al., 2006). One of the most powerful representations for SRL is Markov logic, which generalizes both Markov random fields and finite first-order logic (Richardson & Domingos, 2006). Representing a problem as a Markov logic network (MLN) involves simply writing down a list of first-order formulas and learning weights for those formulas from data. The first step is the task of the knowledge engineer; the second is the focus of this paper.

Currently, the best-performing algorithm for learning MLN weights is Singla and Domingos' (2005) voted perceptron, based on Collins' (2002) earlier one for hidden Markov models. Voted perceptron uses gradient descent to optimize the conditional likelihood of the query atoms given the evidence. Weight learning in Markov logic is a convex optimization problem, and thus gradient descent is guaranteed to find the global optimum. However, convergence to this optimum may be extremely slow. MLNs are exponential models, and their sufficient statistics are the number of times each clause is true in the data. Because this number can easily vary by orders of magnitude from one clause to another, a learning rate that is small enough to avoid divergence in some weights is too small for fast convergence in others. This is an instance of the well-known problem of ill-conditioning in numerical optimization, and many candidate solutions for it exist (Nocedal & Wright, 2006). However, the most common ones are not easily applicable to MLNs, because of the nature of the function being optimized. As in Markov random fields, computing the likelihood in MLNs requires computing the partition function, which is generally intractable. This makes it difficult to apply methods that require performing line searches, which involve computing the function as well as its gradient. These include most conjugate gradient and quasi-Newton methods (e.g., L-BFGS). Two exceptions to this are scaled conjugate gradient (Møller, 1993) and Newton with diagonalized Hessian (Becker & Cun, 1989; Ricotti et al., 1988). In this paper we show how they can be applied to MLN learning, and verify empirically that they greatly speed up convergence. We also obtain good results with a simpler method: per-weight learning rates, with a weight's learning rate being the global one divided by the corresponding clause's empirical number of true groundings.

Voted perceptron approximates the expected sufficient statistics in the gradient by computing them at the MPE state (i.e., the most likely state of the non-evidence atoms given the evidence ones). Since in an

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

MLN the conditional distribution can contain many modes, this may not be a good approximation. Also, using second-order methods requires computing the Hessian (matrix of second-order partial derivatives), and for this the MPE approximation is no longer sufficient. We address both of these problems by instead computing expected counts using MC-SAT, a very fast MCMC algorithm for Markov logic recently introduced by Poon and Domingos (2006).

The remainder of our paper is organized as follows. In Section 2, we describe Markov logic in greater detail. In Section 3, we present several algorithms for MLN weight learning. We compare these algorithms empirically on real-world datasets in Section 4, and conclude in Section 5.

2. Markov logic

A Markov logic network (MLN) consists of a set of first-order formulas and their weights, $\{(w_i, f_i)\}$. Intuitively, a formula represents a noisy relational rule, and its weight represents the relative strength or importance of that rule. Given a finite set of constants, we can instantiate an MLN as a Markov random field (MRF) in which each node is a grounding of a predicate (atom) and each feature is a grounding of one of the formulas (clauses). This leads to the following joint probability distribution for all atoms:

$$P(\mathrm{X}=\mathrm{x}) = \frac{1}{Z}\left(\sum_i w_i n_i(x)\right)$$

where n_i is the number of times the ith formula is satisfied by the state of the world x, and Z is a normalization constant, required to make the probabilities of all worlds to sum to one.

The formulas in an MLN are typically specified by an expert, or they can be obtained (or refined) by inductive logic programming or MLN structure learning (Kok & Domingos, 2005). Many complex models, and in particular many non-i.i.d. ones, can be very compactly specified using MLNs.

Exact inference in MLNs is intractable. Instead, we can perform approximate inference using Markov chain Monte Carlo, and in particular Gibbs sampling (Gilks et al., 1996). However, when weights are large convergence can be very slow, and when they are infinite (corresponding to deterministic dependencies) ergodicity breaks down. This remains true even for more sophisticated alternatives like simulated tempering. A much more efficient alternative, which also preserves ergodicity in the presence of determinism, is the MC-SAT algorithm, recently introduced by Poon and Domingos (2006). MC-SAT is a "slice sampling" MCMC algorithm that uses a modified satisfiability solver (WalkSAT) to sample from the slice. The solver is able to find isolated modes in the distribution very efficiently, and as a result the Markov chain mixes very rapidly. The slice sampling scheme ensures that detailed balance is (approximately) preserved. In this paper we use MC-SAT for inference.

3. Weight learning

Given a set of formulas and a database of atoms, we wish to find the formulas' MAP weights, i.e., the weights that maximize the product of their prior probability and the data likelihood. In this section, we describe a number of alternative algorithms for this purpose.

Richardson and Domingos (2006) originally proposed learning weights generatively using pseudo-likelihood (Besag, 1986). Pseudo-likelihood is the product of the conditional likelihood of each variable given the values of its neighbors in the data. While efficient for learning, it can give poor results when long chains of inference are required at query time. Singla and Domingos (2005) showed that pseudo-likelihood is consistently outperformed by discriminative training, which maximizes the conditional likelihood of the query predicates given the evidence ones. Thus, in this paper we focus on this type of learning.[1]

3.1. Voted perceptron

The simplest way to optimize weights is by gradient descent. In an MLN, the derivative of the conditional log-likelihood with respect to a weight can written as the difference between the number of true groundings of the corresponding clause in the data, and the expected number of true groundings according to the model:

$$\frac{\partial}{\partial w_i} \log P(Y=y|X=x) = n_i - E_w[n_i]$$

where y is the state of the non-evidence atoms in the data, and x the state of the evidence. Computing the expectations $E_w[n_i]$ is generally intractable, but they can be approximated by the counts in the MPE state, which is the most probable state of the non-evidence atoms given the evidence ones. This is the basic idea of the voted perceptron (VP) algorithm (Collins, 2002). VP initializes all weights to zero, and then repeatedly finds the MPE solution given the current weights, uses

[1] For simplicity, we omit prior terms throughout; in our experiments, we use a zero-mean Gaussian prior on all weights with all algorithms.

it to compute the gradient $\mathbf{g}$, and uses $\mathbf{g}$ to update the weight vector $\mathbf{w}$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \mathbf{g}$$

where η is the learning rate (equal to 1 in the original VP). To combat overfitting, instead of returning the final weights, VP returns the average of the weights from all iterations of gradient descent. Collins originally proposed VP for training hidden Markov models discriminatively, and in this case the MPE state is unique and can be computed exactly in polynomial time using the Viterbi algorithm. In MLNs, MPE inference is intractable, but can be reduced to solving a weighted maximum satisfiability problem, for which efficient algorithms exist, like MaxWalkSAT (Kautz et al., 1996). This was the approach followed by Singla and Domingos (2005). However, it is potentially complicated by the fact that the MPE state may no longer be unique, and MaxWalkSAT is not guaranteed to find it.

3.2. Contrastive divergence

A presumably more accurate and stable alternative is to use MCMC to approximate the expected counts. While running an MCMC algorithm to convergence at each iteration of gradient descent is infeasibly slow, others have shown that a few iterations of MCMC yield enough information to choose a good direction for gradient descent (Hinton, 2002). Hinton named this method *contrastive divergence*, because it can be interpreted as optimizing a difference of K-L divergences. While the MCMC algorithm typically used with contrastive divergence is Gibbs sampling, for MLNs the much faster alternative of MC-SAT is available. Because successive samples in MC-SAT are much less correlated than successive sweeps in Gibbs sampling, they carry more information and are likely to yield a better descent direction. In particular, the different samples are likely to be from different modes, reducing the error and potential instability associated with choosing a single mode.

In our experiments, we found that five samples were sufficient, and additional samples were not worth the time: any increased accuracy that 10 or 100 samples might bring was outweighed by the increased time per iteration. We avoid the need for burn-in by starting at the last state sampled in the previous iteration of gradient descent. After every five steps of gradient descent, we reinitialize MC-SAT using MWS. The goal of this reinitialization is to further reduce the risk of getting stuck in a mode.

3.3. Per-weight learning rates

Voted perceptron and contrastive divergence are both simple gradient descent procedures, and as a result highly vulnerable to the problem of ill-conditioning. Ill-conditioning occurs when the *condition number*, the ratio between the highest and lowest eigenvectors of the Hessian, is far from one. In ill-conditioned problems, gradient descent is very slow, because no single learning rate is appropriate for all weights. In MLNs, the Hessian is the negative covariance matrix of the clause counts. Because some clauses can have vastly greater numbers of true groundings than others, the variances of their counts can be correspondingly larger, and ill-conditioning becomes a serious issue.

One solution to this is to have a different learning rate for each weight. Since tuning every learning rate separately is impractical, we use a simple heuristic to assign a learning rate to each weight:

$$\eta_i = \frac{\eta}{n_i}$$

where η is the user-specified global learning rate and n_i is the number of true groundings of the ith formula. When computing this number, we ignore the groundings that are satisfied by the evidence, irrespective of the truth values of the query atoms (e.g., $A \Rightarrow B$ when A is false). This is because, being fixed, they cannot contribute to the variance.

3.4. Diagonal Newton

When optimizing a quadratic function, Newton's method can move to the global minimum or maximum in a single step. It does so by multiplying the gradient, g, by the inverse Hessian, H^{-1}:

$$\mathbf{w} = \mathbf{w} - \mathbf{H}^{-1}\mathbf{g}$$

The Hessian for an MLN is simply the negative covariance matrix:

$$\frac{\partial}{\partial w_i \partial w_j} \log P(Y = y | X = x) = E_w[n_i]E_w[n_j] - E_w[n_i n_j]$$

Like the gradient, this can be estimated using samples from MC-SAT. However, when the number of weights is large, explicitly representing the Hessian and inverting it are infeasible. In the diagonal Newton method, we invert the diagonalized Hessian, making the assumption that clauses are uncorrelated.

In reality, our objective function is only locally quadratic, and clauses may be correlated. Therefore, instead of taking a full Newton step, we take a small

step in the Newton direction. Since the second derivatives of the clause counts are just the clause variances, this can be written as:

$$w_i = w_i + \eta \frac{(n_i - E_w[n_i])}{(E_w[n_i^2] - E_w[n_i]^2)}$$

All expectations can be computed from MC-SAT samples. Since this can be noisy, we smooth each variance estimate by averaging with the previous one. This takes advantage of the fact that weights often change only gradually from iteration to iteration, and is analogous to the smoothing methods often used when learning neural networks by stochastic gradient descent.

When the Hessian is small, the equation above can lead to very large updates, causing the weights to diverge. A simple way to alleviate this problem is to add a small constant term λ to the denominator. In regions of low Hessian, this causes Newton to behave like gradient descent, while leaving it essentially unchanged in regions of large Hessian.

Our per-weight learning rates can actually be seen as a crude approximation of the diagonal Newton method. The number of true groundings not satisfied by evidence is a heuristic approximation to the count variance, which the diagonal Newton method uses to rescale each dimension of the gradient. The diagonal Newton method, however, can adapt to changes in the second derivative at different points in the weight space. Its main limitation is that clauses can be far from uncorrelated. The next method we discuss overcomes this problem.

3.5. Scaled conjugate gradient

Gradient descent can be sped up by, instead of taking a small step of constant size at each iteration, performing a line search to find the optimum along the chosen descent direction. However, on ill-conditioned problems this is still inefficient, because line searches along successive directions tend to partly undo the effect of each other: each line search makes the gradient along its direction zero, but the next line search will generally make it non-zero again. In long narrow valleys, instead of moving quickly along the valley floor to the optimum, gradient descent zigzags.

A solution to this is to impose at each step the condition that the gradient along previous directions remain zero. The directions chosen in this way are called *conjugate*, and the method *conjugate gradient*. Conjugate gradient methods are some of the most efficient available, on a par with quasi-Newton ones. Unfortunately, applying them to MLNs is difficult, because line searches require computing the objective function, and therefore the partition function Z, which is highly intractable. (Computing Z is equivalent to computing all moments of the MLN, of which the gradient and Hessian are the first two.)

Fortunately, we can use the Hessian instead of a line search to choose a step size. This method is known as *scaled conjugate gradient*, and was originally proposed by (Møller, 1993) for training neural networks. Given a conjugate search direction $\mathbf{d}$, gradient $\mathbf{g}$, and Hessian matrix $\mathbf{H}$, we compute the step size α as follows:

$$\alpha = \frac{\mathbf{d}^T\mathbf{g}}{\mathbf{d}^T\mathbf{H}\mathbf{d} + \lambda\mathbf{d}^T\mathbf{d}}$$

With a quadratic error function, adjusting the weights by $\alpha\mathbf{d}$ places them at the optimum of the function along $\mathbf{d}$. The role of the λ term is similar to that in diagonal Newton. Additionally, it has the useful effect of making the computation of α more robust to noise in our estimate of the Hessian.

In models with thousands of weights or more, storing the entire Hessian matrix can be impractical. However, when the Hessian appears only inside a quadratic form, as above, the value of this form can be computed simply as:

$$\mathbf{d}^T\mathbf{H}\mathbf{d} = (E_w[\textstyle\sum_i d_i n_i])^2 - E_w[(\sum_i d_i n_i)^2]$$

Conjugate gradient is typically more effective with a preconditioner. A preconditioner is a linear transformation that attempts to reduce the condition number of the problem, speeding up the conjugate gradient algorithm. Good preconditioners approximate the inverse Hessian. We use the inverse diagonal Hessian as our preconditioner.

4. Experiments

4.1. Datasets

Our experiments used two standard relational datasets representing two important relational tasks: Cora for entity resolution, and WebKB for collective classification.

The Cora dataset consists of 1295 citations of 132 different computer science papers, drawn from the Cora Computer Science Research Paper Engine. This dataset was originally labeled by Andrew McCallum[2], segmented into fields by Bilenko and Mooney (2003), and further cleaned by Singla and Domingos (2006). We used the version by Singla and Domingos in our experiments, with five splits for cross-validation.

[2] www.cs.umass.edu/ mccallum/data/cora-refs.tar.gz

The task on Cora is to predict which citations refer to the same paper, given the words in their author, title, and venue fields. The labeled data also specifies which pairs of author, title, and venue fields refer to the same entities. In our experiments, we evaluated the ability of the model to deduplicate fields as well as citations. Since the number of possible equivalances is very large, we used the canopies found by Singla and Domingos (2006) to make this problem tractable.

The MLN we used for this is the "MLN(B+C+T)" model described by (Singla & Domingos, 2006). Its formulas link words to citation equivalence, words to field equivalence, and field equivalence to citation equivalence. In this way, word co-occurrence affects the probability that two citations are the same both indirectly through field similarities and directly. These rules are repeated for each word appearing in the database so that individualized weights can be learned, representing the relative importance of each word in each context. This model also features transitive closure for all equivalance predicates. The total number of weights is 5486. During learning, the number of ground clauses exceeded 3 million.

The WebKB dataset consists of labeled web pages from the computer science departments of four universities. We used the relational version of the dataset from (Craven & Slattery, 2001), which features 4165 web pages and 10,935 web links, along with the words on the webpages, anchors of the links, and neighborhoods around each link.

Each web page is marked with some subset of the categories person, student, faculty, professor, department, research project, and course. Our goal is to predict these categories from the words on the web page and their linking structures.

We used a very simple MLN for this model, consisting only of formulas linking words to page classes, and page classes to the classes of linked pages. The "word-class" rules were of the following form, with all combinations of positive and negative literals:

```
Has(page,word) => Class(page,class)
```

We learned a separate weight for each of these rules for each `(word, class)` pair. Classes of linked pages were related by the formula:

```
Class(page1,class1) ^ LinksTo(page1,page2)
  => Class(page2,class2)
```

We learned a separate weight for this rule for each pair of classes. When instantiated for each word and class, the model contained 21,728 weights. While simple to write, this model represents a complex, non-i.i.d. probability distribution in which query predicates are linked in a large graph. During learning, the number of ground clauses exceeded 300,000.

4.2. Metrics

To score our models, we run MC-SAT for 100 burn-in and 1000 sampling iterations. The marginal conditional probability of each query atom is the fraction of samples in which the atom was true with a small prior to prevent zero counts.

From these marginal probabilities, we estimate conditional log-likelihood (CLL) by averaging the log marginal probabilities of the true values of the query predicates. CLL is the metric all of the algorithms are attempting to optimize.

However, in cases such as entity resolution where the class-distribution is highly skewed, CLL can be a poor metric. For this reason, we also look at AUC, the area under the precision-recall curve. Precision is (# true positives)/(# predicted positives), and recall is (# true positives)/(# actual positives). By setting different probability thresholds, we get different recalls and precisions. The area under the curve is computed by plotting precision as a function of recall and numerically integrating the function using trapezoids.

The disadvantage of AUC is that it ignores calibration: AUC only looks at whether true atoms are given higher probability than false atoms.

4.3. Methodology

We ran our experiments using five-way cross-validation for Cora and four-way cross-validation for WebKB.

Each algorithm was tuned separately on each train/test split using one held-out training database. For conjugate gradient methods, we tuned lambda; for all other algorithms, we tuned the learning rate. The tuning procedure consisted of running each algorithm for four hours with various values of the tuning parameter. The values used for tuning ranged over three to five orders of magnitude and were based on preliminary experiments.

For each split, we saved the parameter values that yielded the models with the best area under the precision-recall curve (AUC) and conditional log likelihood (CLL) on the hold-out data. When different models excelled at the different metrics, we tried to pick a reasonable compromise configuration.

After tuning all algorithms, we reran them for 12 hours with their respective training sets, including the held-

out validation data.

4.4. Results

Our results for the Cora and WebKB datasets are shown in Figures 1 and 2, respectively. The abbreviations for the algorithms are as follows: VP is voted perceptron, CD is contrastive divergence, -PW indicates per-weight learning rates, DN is diagonal Newton, SCG is scaled conjugate gradient, and PSCG is scaled conjugate gradient preconditioned with the diagonalized Hessian.

PSCG is the most accurate of all the algorithms compared, obtaining the best CLL and AUC on both Cora and WebKB. It is also among the fastest to converge on WebKB, and the fastest after the per-word methods on Cora. The worst performers are the pure gradient descent methods; on Cora, they are far from converging even after 12 hours, while most of the other methods converge in less than an hour.

On Cora, the algorithms that adjust the gradient using true clause counts or count variance do much better than those that do not. This suggests that Cora is an extremely ill-conditioned dataset. Without a preconditioner, even scaled conjugate gradient does poorly. This is because, like voted perceptron and contrastive divergence, the first step it takes is in the direction of the gradient. On a very ill-conditioned dataset like Cora, the gradient is a very poor choice of search direction.

On WebKB, the ill-conditioning is less of an issue: while the per-word learning rates help voted perceptron, they actually hurt contrastive divergence. A similar effect seems to occur with diagonal Newton: it converges much more slowly than simple contrastive divergence. The cause of this may have something to do with our MLN: our formulas linking words to page classes are strongly correlated. This makes the diagonalized Hessian a poor approximation of the actual Hessian matrix. However, it is still very useful as a precondition. Scaled conjugate gradient does very well with the preconditioner and only average without.

A salient phenomenon in these experiments is that the performance of some of the algorithms sometimes degrades markedly with additional learning time. At first sight this might seem to be a typical example of overfitting, but the fact that the behavior is very different for different algorithms, which all optimize the same model and objective function, suggests otherwise. We have investigated this, and found that the problem is caused by inference falling into a single mode, and is also observed if we perform a different run of inference on the *training* data. When learning optimizes weights for one mode and inference falls into another, performance is (not surprisingly) poor. Further, when the learner adjusts the weight vector and the resulting clause counts do not change as a result, do to the approximate character of the inference, the learner repeatedly makes the same adjustment to the weights, making the mode deeper and further degrading performance. Clearly, this is a key problem for future work.

5. Conclusion

Weight learning for Markov logic networks can be extremely ill-conditioned, making simple gradient descent-style algorithms very slow to converge. In this paper we studied a number of more sophisticated alternatives, of which the best-performing one is preconditioned scaled conjugate gradient. We attribute this to its effective use of second-order information. However, the simple heuristic of dividing the learning rate by the true clause counts for each weight also gives good results. Using one of these methods instead of gradient descent can greatly reduce learning time, or yield a better model given a fixed learning time. We also identified further problems in MLN weight learning, which will be the focus of future work.

References

Becker, S., & Cun, Y. L. (1989). Improving the convergence of back-propagation learning with second order methods. *Proceedings of the 1988 Connectionist Models Summer School* (pp. 29–37). San Mateo, CA: Morgan Kaufmann.

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, *48*, 259–302.

Bilenko, M., & Mooney, R. (2003). Adaptive duplicate detection using learnable string similarity measures. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 39–48). Washington, DC: ACM Press.

Collins, M. (2002). Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*. Philadelphia, PA.

Craven, M., & Slattery, S. (2001). Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, *43*, 97–119.

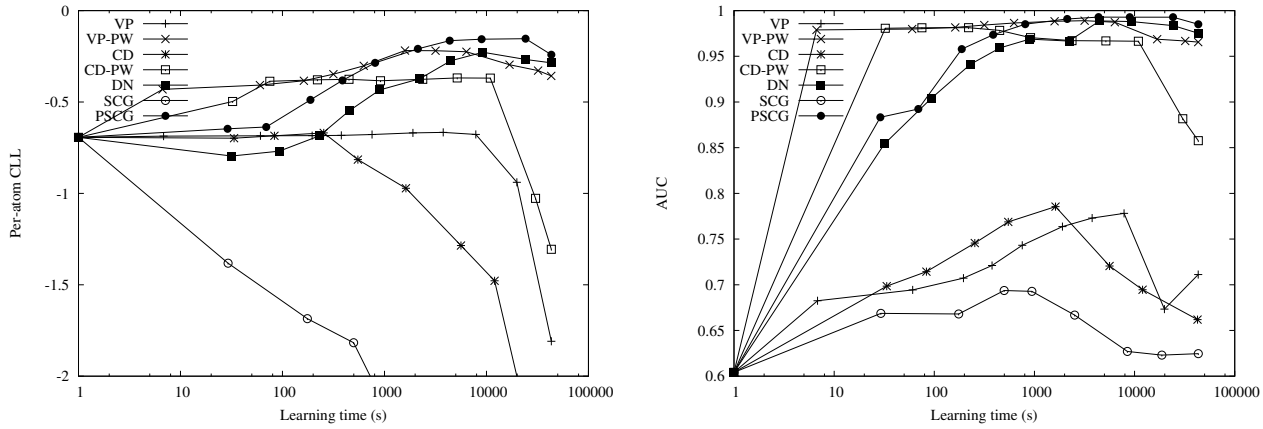

Figure 1. CLL and AUC results for the Cora dataset. Note that learning times are shown on a logarithmic scale.

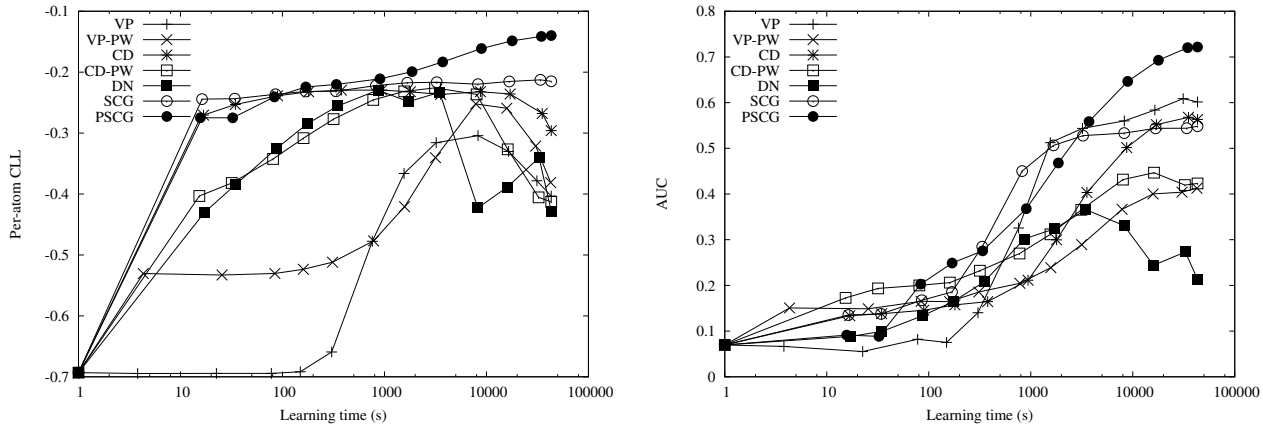

Figure 2. CLL and AUC results for the WebKB dataset. Note that learning times are shown on a logarithmic scale.

Fern, A., Getoor, L., & Milch, B. (Eds.). (2006). *Proceedings of the ICML-2006 Workshop on Open Problems in Statistical Relational Learning*. Pittsburgh, PA.

Gilks, W. R., Richardson, S., & Spiegelhalter, D. J. (Eds.). (1996). *Markov chain Monte Carlo in practice*. London, UK: Chapman and Hall.

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

Kautz, H., Selman, B., & Jiang, Y. (1996). A general stochastic approach to solving problems with hard and soft constraints. In D. Du, J. Gu and P. M. Pardalos (Eds.), *The satisfiability problem: Theory and applications*, 573–586. New York, NY: American Mathematical Society.

Kok, S., & Domingos, P. (2005). Learning the structure of Markov logic networks. *Proceedings of the Twenty-Second International Conference on Machine Learning* (pp. 441–448). Bonn, Germany: ACM Press.

Møller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*, 525–533.

Nocedal, J., & Wright, S. (2006). *Numerical optimization*. New York, NY: Springer.

Poon, H., & Domingos, P. (2006). Sound and efficient inference with probabilistic and deterministic dependencies. *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston, MA: AAAI Press.

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Ricotti, L. P., Ragazzini, S., & Martinelli, G. (1988). Learning of word stress in a sub-optimal secondorder backpropagation neural network. *Proceedings of the IEEE International Conference on Neural Networks* (pp. 355–361). San Diego, CA: IEEE.

Singla, P., & Domingos, P. (2005). Discriminative training of Markov logic networks. *Proceedings of the Twentieth National Conference on Artificial Intelligence* (pp. 868–873). Pittsburgh, PA: AAAI Press.

Singla, P., & Domingos, P. (2006). Entity resolution with Markov logic. *Proceedings of the Sixth IEEE International Conference on Data Mining* (pp. 572–582). Hong Kong: IEEE Computer Society Press.