Integrating Multiple Learning Components Through Markov Logic

Thomas G. Dietterich and **Xinlong Bao**
1148 Kelley Engineering Center
Oregon State University
Corvallis, OR 97331, U.S.A
{tgd, bao}@eecs.oregonstate.edu

Abstract

This paper addresses the question of how statistical learning algorithms can be integrated into a larger AI system both from a practical engineering perspective and from the perspective of correct representation, learning, and reasoning. Our goal is to create an integrated intelligent system that can combine observed facts, hand-written rules, learned rules, and learned classifiers to perform joint learning and reasoning. Our solution, which has been implemented in the CALO system, integrates multiple learning components with a Markov Logic inference engine, so that the components can benefit from each other's predictions. We introduce two designs of the learning and reasoning layer in CALO: the MPE Architecture and the Marginal Probability Architecture. The architectures, interfaces, and algorithms employed in our two designs are described, followed by experimental evaluations of the performance of the two designs. We show that by integrating multiple learning components through Markov Logic, the performance of the system can be improved and that the Marginal Probability Architecture performs better than the MPE Architecture.

Introduction

Statistical machine learning methods have been developed and applied primarily in stand-alone contexts. In most cases, statistical learning is employed at "system build time" to construct a function from a set of training examples. This function is then incorporated into the system as a static component. For example, in the NCR check reading system (LeCun *et al.* 1998), the neural network system for reading check amounts is a sophisticated but static component of the system.

Our aims are more ambitious. We want to address the question of how statistical learning algorithms can be integrated into a larger AI system through a reasoning layer. We seek AI systems that exhibit end-to-end learning across all (or at least most) components of the system after deployment. We also anticipate that these systems will be built out of many separately-constructed components which may or may not have learning capabilities. Hence, we cannot insist on a uniform learning and reasoning algorithm for all of the components of the system. Instead, we seek to provide a learning and reasoning layer that can interconnect these components in such a way that if the components implement specified interfaces, the system will exhibit the desired end-to-end learning behavior.

In this paper, we present and evaluate two designs for the learning and reasoning layer, both based on Markov Logic (Domingos & Richardson 2006) as the reasoning engine. The rest of the paper is organized as follows. First, we describe the CALO system, which provides the framework and constraints within which this work was carried out. Second, we describe the architecture, interfaces, and algorithms employed in our two designs. This is followed by some experimental evaluations of the performance of the two designs. The paper concludes with an agenda for future research in this area.

Learning in CALO

The work described here arose as part of the CALO project. CALO (**C**ognitive **A**gent that **L**earns and **O**rganizes) is an integrated AI system to support knowledge workers at the computer desktop. One of the primary functions of CALO is to help the user keep files, folders, email messages, email contacts, and appointments organized. CALO models the user's computer work life as consisting of a set of projects, where each project involves a set of people, meetings, files, email messages, and so on.

Figure 1 shows a portion of CALO's relational model of this information with objects as nodes and relations as arcs. All of the objects and many of the relations are observed with certainty (e.g., which files are stored in which folders), but the associations between projects and all of the other objects are inferred by applying a mix of hand-written rules and learned classifiers. CALO also provides interfaces for the user to associate objects with projects, so that when an incorrect association is made, the user can correct it and establish new associations.

Rules are employed to capture domain knowledge. Some of the rules used in CALO are shown in Table 1. For example, the Attachment Rule says that if a document is attached to an email message, both the document and the message tend to be associated with the same project. End-users can write their own rules as well.

One issue that arises when integrating learned classifiers

Name	Rule
Attachment Rule	$\forall\, E, P, D\ attached(D, E)\ \supset\ [projectOf(E, P)\ \equiv\ projectOf(D, P)][2.20]$
Sender Rule	$\forall\, E, P, H\ sender(E, H)\ \supset\ [projectOf(E, P)\ \equiv\ projectOf(H, P)][2.20]$
Recipient Rule	$\forall\, E, P, H\ recipient(E, H)\ \supset\ [projectOf(E, P)\ \equiv\ projectOf(H, P)][2.20]$
User Input Rule	$\forall\, E, P\ userFeedback(E, P)\ \supset\ projectOf(E, P)[2.94]$
Single Project Rule	$\forall\, E\ \exists!\ P\ projectOf(E, P)[1.73]$

Table 1: Some rules used in CALO. Markov Logic weights are given in square brackets.

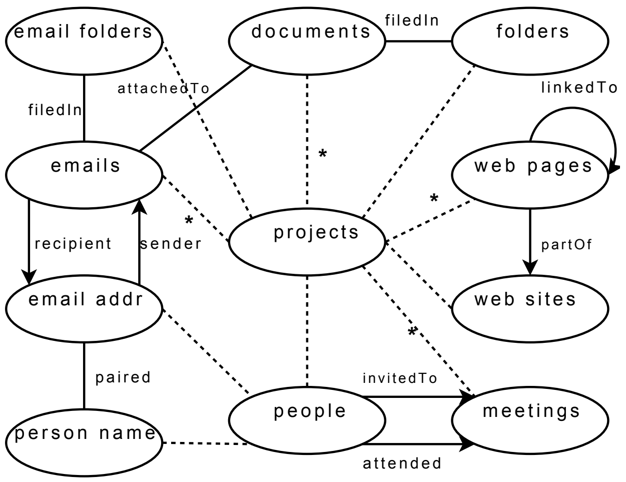


Figure 1: A portion of CALO's relational model. Dashed lines indicate uncertain relations; asterisks indicate relations learned by classifiers.

with rules is the problem of *distal credit assignment*. A learned classifier can make a prediction that then triggers a sequence of rule firings that then results in a project association for another object. If user marks this association as incorrect, it could be because the final rule in the sequence was wrong, but it is more likely that the fault lies with the learned classifier. Distal credit assignment is the process of propagating the user correction back along the chain of inference to the component that is most likely responsible for the error. The learning and reasoning layer described in the next section can solve the distal credit assignment problem; however, we do not have space to discuss it in this paper. Instead, we will focus on how multiple learning components can be integrated via this layer to improve the performance of the whole system.

A key observation of our work is that when rules and classifiers are intermixed, the rules can establish connections between learning tasks. This creates the opportunity to perform *relational co-training*. Co-training is a method for semi-supervised learning in which there is a single, standard supervised learning problem, but each object has two "views" or representations (Blum & Mitchell 1998). Two classifiers are learned from labeled training examples, one from each view, but with the added constraint that the two classifiers should agree on the available unlabeled data. There are many algorithms for co-training, e.g., (Blum & Mitchell 1998; Nigam & Ghani 2000).

In the case of CALO, instead of multiple views of objects, we have multiple objects with relations among them. Figure 2 is a Venn diagram showing documents, email messages, and the subset of documents attached to email messages. The combination of the document classifier f_D, the email classifier f_M, and the Attachment Rule creates an opportunity for relational co-training. The two classifiers can be trained on their respective labeled data but with the added constraint that for the documents and email messages in region A, the predictions should be consistent. Because this co-training happens through the relations between objects or the chains of relations/rules, we call this learning behavior *relational co-training*.

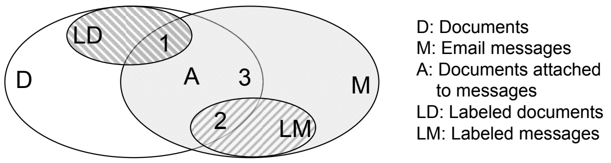


Figure 2: Venn diagram showing document-email relations.

Through relational co-training, multiple learning components can be integrated into the CALO system. We now describe two designs for an architecture that can support this behavior in a general, unified way.

Integrating Learning and Reasoning

The starting point for our design is to divide the knowledge of the system into indefeasible knowledge and defeasible (probabilistic) knowledge. Indefeasible knowledge includes events that are observed with certainty. The defeasible knowledge includes all things that are uncertain. For example, in Figure 2, document objects, email objects and the $attached(D, E)$ relations are indefeasible knowledge. The Attachment Rule and predictions from the email and document classifiers are defeasible knowledge.

The probabilistic knowledge base is represented in Markov Logic. A Markov Logic knowledge base consists of a set of weighted first-order clauses and a set of constant symbols. (Conceptually, the formulas that appear in the indefeasible knowledge base are treated as having infinite weights in Markov Logic.) The knowledge base defines a probability distribution over the set of possible worlds that can be constructed by grounding the clauses using the constant symbols and then assigning truth values to all of the ground formulas. A possible world is a truth assignment that does not entail a contradiction. Given such a truth assignment α, let $SAT(\alpha)$ be the set of ground formulas that are satisfied. For each formula F, let $w(F)$ be the weight assigned to that formula in the Markov Logic. The score of the truth assignment is defined as the sum of the weights of the satisfied formulas:

$$\text{score}(\alpha) = \sum_{F \in SAT(\alpha)} w(F).$$

The probability of truth assignment is then defined as

$$P(\alpha) = \frac{\exp \text{score}(\alpha)}{\sum_\omega \exp \text{score}(\omega)}, \qquad (1)$$

where ω indexes all possible (contradiction-free) truth assignments. The normalized exponentiated sum on the right-hand side defines a Gibbs distribution that assigns non-zero probability to every possible world. The score of a possible world α is proportional to the log odds of that world according to α.

If we consider the weight $w(F)$ on a particular formula F, we can interpret it as the amount by which the log odds of a possible world will change if F is satisfied compared to a world in which F is not satisfied (but all other formulas do not change truth value).

In Table 1, the numbers in the square brackets after the formulas are the weights assigned to them. These weights can be adjusted through a weight learning process, but that is beyond the scope of this paper. In this paper, we assume that the weights of the rules are all fixed. Weights for the classifiers' predictions are assigned by taking the predicted probabilities and converting them to log odds according to:

$$w = \log \frac{P(Class|Object)}{1 - P(Class|Object)}, \qquad (2)$$

The MPE Architecture

Figure 3 shows our first architecture, called the MPE Architecture. The box labeled PCE denotes the Probabilistic Consistency Engine. This is the inference engine for the Markov Logic system. In this first architecture, its task is to compute the possible world with the highest score. This is known in probabilistic reasoning as the Most Probable Explanation (MPE). The grounded Markov Logic knowledge base defines a Weighted Maximum Satisfiability (Weighted MaxSAT) problem, for which many reasonably efficient solvers are available. Our implementation employs a mix of the MaxWalkSat (Kautz, Selman, & Jiang 1997) algorithm for fast, approximate solution and the Yices linear logic solver (Dutertre & de Moura 2006) for exact solution. The implementation employs the Lazy-SAT algorithm (Singla & Domingos 2006) which seeks to compute only those groundings of formulas that are needed for computing the weighted MaxSAT solution.

Under this architecture, the learning components draw their training examples from the MPE and post their predictions into the Probabilistic Knowledge Base. We need to take care, however, that a learning component does not treat its own predictions (which may be very weak) as labels for subsequent training. That is, a prediction such as $projectOf(report1.doc, CALO)$ might be asserted with a probability of 0.51 (log odds of 0.04). But this might then cause the MPE to contain

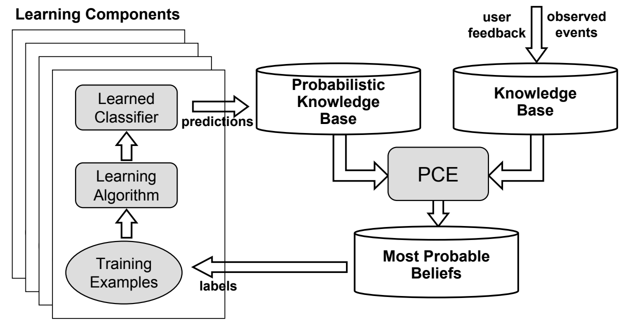


Figure 3: The MPE Architecture.

$projectOf(report1.doc, CALO)$. If the document classifier treated this as an ordinary training example, it would rapidly lead to "dogmatic" behavior where the classifier would become absolutely certain of all of its predictions.

To address this problem, the PCE computes a separate MPE for each learning component. When computing the MPE for learning component i, the PCE ignores all assertions that were made by component i. Hence, we call this the "Not Me MPE" for component i.

In order to be integrated into the architecture, each learning component must be able to accept (unweighted) training examples of the form $TargetPredicate(Object, Class)$ and produce probabilistic predictions of the form "$TargetPredicate(Object, Class)$ with probability p". The architecture ensures that the following three invariants are always true:

- For each learning component, the set of training examples for that component consists exactly of all instances of $TargetPredicate(Object, Class)$ that appear in the Not Me MPE for that component.
- For each learning component, the learned classifier is always the classifier that results from applying the learning algorithm to the current set of training examples.
- For each learning component, each $Object$, and each $Class$, the Probabilistic Knowledge Base has an assertion of the form $TargetPredicate(Object, Class)$ with weight w computed according to Equation 2 where $P(Class|Object)$ is the probability assigned to $Class$ by the current learned classifier.

These invariants are maintained by the following infinite loop:

1. Accept new constants and new assertions.
2. Compute the global MPE and the Not Me MPEs for each learning component.
3. Update the training examples for each learning component (if necessary).
4. Recompute the learned classifier for each learning component (if necessary).
5. Update the probabilistic predictions of each learning component (if necessary).
6. Repeat

In practice, these steps are performed concurrently and asynchronously. We employ anytime algorithms for the MPE calculations.

The MPE Architecture supports relational co-training over multiple learning components. Consider the Attachment example shown in Figure 2. The Not Me MPE for the document classifier includes: (1) labeled examples from LD; (2) documents in region 2 with class labels propagated from email labels via the Attachment Rule; and (3) documents in region 3 with class labels propagated from the predicted email labels by the email classifier. The Not Me MPE for the email classifier is similar.

Then both classifiers will be retrained based on their own Not Me MPEs. Each classifier will be influenced by the predictions from the other classifier, as transmitted by the Attachment Rule. This may cause the classifiers to change some of their predictions. This will result in changes in the Not Me MPEs and, therefore, additional changes in the classifiers, which will cause the process to repeat.

Experiments have revealed that this process can loop. The predicted class labels of some of the "unsupervised" documents and email messages and flip back and forth forever. To prevent this, we relax slightly the enforcement of the third invariant. After a classifier has been retrained, if its predicted probability for an example does not change by more than ϵ (e.g., 0.05), then the change is ignored. Obviously, setting ϵ large enough will cause the iterations to terminate.

The Marginal Probability Architecture

The MPE Architecture was initially adopted because the MPE computation is relatively efficient. However, there are several problems with this design. First, there are often "ties" in the computation of the MPE, because there can be many literals that can be either true or false without changing the score of the possible worlds. We attempt to identify and ignore such literals, but this is difficult in general. Consequently, some of the training examples can receive random labels, which can hurt performance. Second, each new classifier that is added to CALO requires an additional Not Me MPE computation. Since our long-term goal is to have 10-20 classifiers, this will slow down the reasoning by a factor of 10-20. Finally, from a learning perspective, this approach to joint training is known to produce biased results. The MPE approach is directly analogous to the use of the Viterbi approximation in training hidden Markov models and to the k-Means clustering algorithm for training Gaussian mixture models. In all of these cases, it is well-established that the resulting model fit is biased and does not maximize the likelihood of the training data.

These considerations led us to develop the Marginal Probability Architecture, which is shown in Figure 4. The basic structure is the same. The key change is that instead of computing the MPE, the PCE computes the marginal probability of each ground instance of the target predicates for the various classifiers. This is the probability $P(TargetPredicate(Object, Class))$ given the contents of both the Knowledge Base and the Probabilistic Knowledge Base.

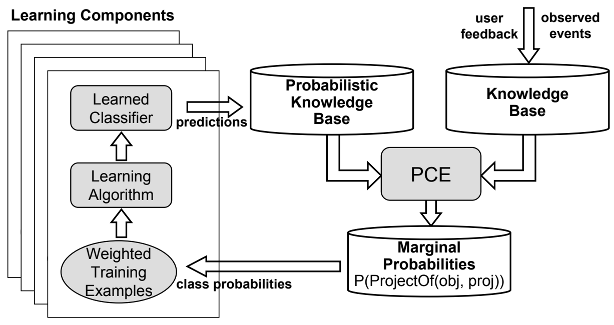


Figure 4: The Marginal Probability Architecture.

In addition, instead of standard training examples, each learning algorithm that is integrated into the architecture must be able to accept *weighted training examples* of the form "$TargetPredicate(Object, Class)$ with probability p" and produce probabilistic predictions of the same form. The architecture ensures that the following three invariants are always true:

- For each learning component, the set of training examples for that component consists exactly of the set of all groundings of the form $TargetPredicate(Object, Class)$ weighted by the marginal probability of those ground instances as computed by the PCE.
- For each learning component, the learned classifier is always the classifier that results from applying the learning algorithm to the current set of weighted training examples.
- For each learning component, each $Object$, and each $Class$, the Probabilistic Knowledge Base has an assertion of the form $TargetPredicate(Object, Class)$ with weight w computed according to Equation 2 where $P(Class|Object)$ is the probability assigned to $Class$ by the current learned classifier.

The PCE computes the marginal probabilities by applying the Lazy MC-SAT algorithm (Poon & Domingos 2006). MC-SAT is a Markov Chain Monte Carlo algorithm based on slice sampling. It generates a sequence of possible worlds that (after convergence) is drawn from the distribution defined by Equation 1. Lazy MC-SAT extends the MC-SAT algorithm to instantiate ground clauses only as needed. In our implementation, the Lazy MC-SAT sampling process is constantly running in the background. Because new evidence and new or updated predictions are continually being added to the indefeasible knowledge base and the probabilistic knowledge base, the MC-SAT process probably never converges. Consequently, we base our probability estimates on the 200 most recent samples from this process.

Note that in this design, the predictions of each learning algorithm influence its own weighted training examples. If the learning algorithm maximizes the weighted log likelihood of the training examples, then this architecture can be viewed as an implementation of the EM algorithm (McLachlan & Krishnan 1997) applied to semi-supervised learning, and it will converge to a local maximum in the data likelihood.

The Marginal Probability Architecture also supports the relational co-training over multiple learning components. In the Attachment example, the document classifier will be trained with (1) labeled examples from LD; (2) its own probabilistic predictions on unlabeled documents that are not attached to any email messages; (3) documents in region 2 with class probabilities propagated from email labels via the Attachment Rule; and (4) documents in region 3 with class probabilities combining the predictions from both the document classifier and the email classifier. In this example, the marginal probabilities of any document in region 3 computed by the PCE are equivalent to the probabilities obtained by multiplying the predicted probabilities from the two classifiers for each class and renormalizing so that they sum to 1. The email classifier is trained similarly. Then the trained classifiers will predict the projects of all unlabeled objects and assert the weighted predictions into the PCE. Again, new marginal probabilities will be computed and fed to the learning components. This process will iterate until the changes in the marginal probabilities are smaller than ϵ.

In practice, in the CALO system, new observations are arriving continually and both the MPE and the marginal probability computations are only approximate, so the whole learning and reasoning process never terminates.

Experimental Evaluations

In this section, we present the results of experiments on the two architectures.

Experiment Settings and Data Collection

We performed our experiments on a set of data collected from 14 CALO users (all knowledge workers). These users manually labeled their own files into 4 projects. Each of them labeled about 100 files. The results presented here are averaged across all 14 users. Unfortunately, these files do not include email messages, nor are any of the files attached to email messages. Instead, we randomly split the files for each user into sets D and M of equal size to simulate documents and emails. Then we generated a set A of attachments by randomly pairing files from D and M under two constraints: (1) each pair contains one file from D and one file from M; (2) two files in a pair must be labeled with the same project. By doing this, we enforce the Attachment Rule to be perfectly true in our experiments. In the experiments, we vary the size of A to contain 20%, 30%, 40%, 50% and 60% of the files.

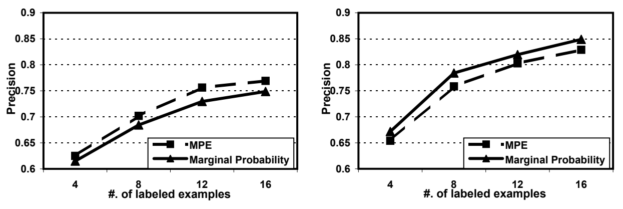
A subset of D was randomly chosen to be Labeled Documents (LD), and similarly a subset of M was randomly chosen to be Labeled Emails (LM). All other files in D and M are treated as unlabeled. We varied the sizes of LD and LM to contain 4, 8, 12, and 16 files.

Two widely-used learning algorithms, Naive Bayes and Logistic Regression, were examined in our experiments. Because our aim is to create a general architecture in which any learning algorithm can be used, we did not tune the two algorithms. They are used as black boxes that support the standard learning interfaces: telling examples, training classifier, and predicting examples. The files and emails are represented using Bag-Of-Words features. The accuracy of the final classifiers over all unlabeled documents/emails was measured at convergence. In each single experiment, the two classifiers are trained using the same learning algorithm.

Results - Single Classifier

Before we can evaluate the effects of relational co-training among multiple classifiers, we want to make sure that if there is only one classifier in the system, the application of the reasoning layer will not result in any weird behavior.

In fact, it is quite interesting to examine the behavior of a single classifier under our two architectures. Assume we only have one classifier, the document classifier. Under the MPE Architecture, the Not Me MPE contains only the labeled documents in LD, and it will not change. That means the classifier works exactly as if there is no reasoning layer. Therefore, the performance of the single classifier under the MPE Architecture can be treated as the baseline measure. Under the Marginal Probability Architecture, the classifier will be trained with the labeled documents in LD and its own probabilistic predictions on the unlabeled documents in D. The training-and-predicting iterations will continue until convergence.



(a) Naive Bayes (b) Logistic Regression

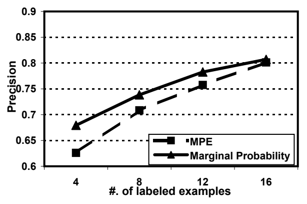
Figure 5: Learning Curves - Single classifier.

Figure 5 shows the learning curves of the single classifier under the two architectures. We can see that both learning algorithms show expected learning behaviors under either architecture. Under the Marginal Probability Architecture, Logistic Regression performs better than baseline, while Naive Bayes performs worse. However, the differences are small enough that we can claim it is safe to integrate single classifiers into the reasoning layer.

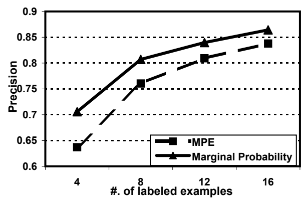
Results - Two Classifiers

Now we integrate two classifiers, the document classifier and the email classifier, to test their performance under the simulated Attachment scenario.

Figure 6 shows the learning curves of the classifiers under the two architectures when 40% of the documents and emails are attached to each other. Note that the classifiers' performance under the MPE Architecture is almost identical to the baseline performance (dashed lines in Figure 5). Under the Marginal Probability Architecture, however, we observe an improvement over the baseline performance, and Logistic Regression benefits from relational co-training more than Naive Bayes.

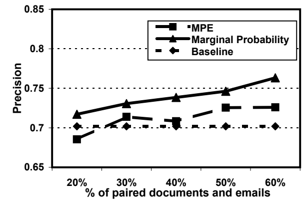


(a) Naive Bayes

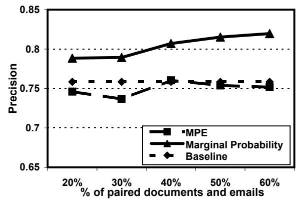


(b) Logistic Regression

Figure 6: Learning Curves - Two classifiers.



(a) Naive Bayes



(b) Logistic Regression

Figure 7: Co-training performance with different size of the attachment set A.

Figure 7 shows how the performance of the "integrated" classifiers changes with the size of the attachment set A. The number of labeled examples is fixed at 8 in this experiment. As expected, the performance improvement becomes larger when the number of attached pairs increases, because more attached pairs lead to more co-training opportunities. Under the MPE Architecture, this trend is not so obvious, and the performance is close to the baseline even when there are more attachment pairs. Under the Marginal Probability Architecture, on the other hand, for both Naive Bayes and Logistic Regression, performance goes up with more co-training opportunities and it is significantly better than the baseline performance.

Conclusion and Future Work

In this paper, we applied a Markov Logic approach to integrate multiple learning components into the CALO system. A probabilistic inference engine, PCE, is implemented for this purpose. Two designs of the interfaces between PCE and the learning components, the MPE Architecture and the Marginal Probability Architecture, were described and evaluated. Experimental results showed that both designs can work, but that the Marginal Probability Architecture works better on improving the performance of the learning components when there are co-training opportunities.

There are many more forms of reasoning and problem solving that remain beyond the scope of our current designs. Some that we hope to incorporate in the near future include entity resolution, ranking, and information extraction. The long term goal is to support all kinds of reasoning and problem solving found in AI systems.

Another direction we want to pursue is to have the PCE automatically adjust the weights of the rules in the Probabilistic Knowledge Base. Humans are good at writing rules but not so good at assigning weights to the rules they write. We also are studying methods for allowing CALO to learn its own rules. We plan to develop online, incremental algorithms based on the weight learning and rule learning algorithms currently implemented in the Alchemy Markov Logic System (Kok & Domingos 2005; Lowd & Domingos 2007).

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), or the Department of the Interior. The authors thank Pedro Domingos, Tomas Uribe, and the CALO Year 3 test participants for helping with this research.

References

Blum, A., and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *COLT-1998*, 92–100. Morgan Kaufmann.

Domingos, P., and Richardson, M. 2006. Markov logic networks. *Machine Learning* 62:107–136.

Dutertre, B., and de Moura, L. 2006. A fast linear-arithmetic solver for DPLL(T). In Ball, T., and Jones, R. B., eds., *International Conference on Computer Aided Verification (CAV-06)*, volume 4144 of *Lecture Notes in Computer Science*. Berlin / Heidelberg: Springer. 81–94.

Kautz, H.; Selman, B.; and Jiang, Y. 1997. A general stochastic approach to solving problems with hard and soft constraints. In *The Satisfiability Problem: Theory and Applications*. New York, NY: American Mathematical Society. 573–586.

Kok, S., and Domingos, P. 2005. Learning the structure of markov logic networks. In *ICML-2005*, 441–448. ACM Press.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Lowd, D., and Domingos, P. 2007. Efficient weight learning for markov logic networks. In *CIKM-2007*, 86–93.

McLachlan, G., and Krishnan, T. 1997. *The EM algorithm and extensions*. New York: Wiley.

Nigam, K., and Ghani, R. 2000. Analyzing the effectiveness and applicability of co-training. In *CIKM-2000*, 86–93.

Poon, H., and Domingos, P. 2006. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI-2006*, 458–463. Menlo Park, CA: AAAI Press/MIT Press.

Singla, P., and Domingos, P. 2006. Memory-efficient inference in relational domains. In *AAAI-2006*, 488–493. Menlo Park, CA: AAAI Press/MIT Press.