Learning First-Order Probabilistic Models with Combining Rules

Sriraam Natarajan NATARASR@EECS.ORST.EDU
Prasad Tadepalli TADEPALL@EECS.ORST.EDU
Eric Altendorf ALTENDER@EECS.ORST.EDU
Thomas G. Dietterich TGD@EECS.ORST.EDU
Alan Fern AFERN@EECS.ORST.EDU
Angelo Restificar ANGELO@EECS.ORST.EDU
School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331-3202, USA

Abstract

First-order probabilistic models allow us to model situations in which a random variable in the first-order model may have a large and varying numbers of parent variables in the ground ("unrolled") model. One approach to compactly describing such models is to independently specify the probability of a random variable conditioned on each individual parent (or small sets of parents) and then combine these *conditional distributions* via a combining rule (e.g., Noisy-OR). This paper presents algorithms for learning with combining rules. Specifically, algorithms based on gradient descent and expectation maximization are derived, implemented, and evaluated on synthetic data and on a real-world task. The results demonstrate that the algorithms are able to learn the parameters of both the individual parent-target distributions and the combining rules.

1. Introduction

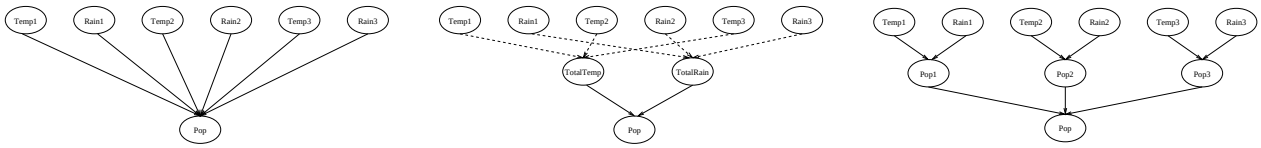
New challenging application problems in networked and relational data have led to the development of algorithms for learning in first-order relational probabilistic models. Several formalisms have been introduced including probabilistic relational models (PRMs) (Getoor et al., 2001), directed acyclic probabilistic entity-relationship (DAPER) models (Heckerman et al., 2004), Bayesian logic programs (BLPs) (Kersting & Raedt, 2002), probabilistic horn abduction language (Poole, 1993), probabilistic logic programs (PLPs) (Ngo & Haddawy, 1995), stochastic logic programs (SLPs) (Muggleton, 1996), Bayesian logic (BLOG) models (Milch et al., 2004), relational Bayesian networks (RBNs) (Jaeger, 1997), and Markov logic (ML) (Domingos & Richardson, 2004). In most of these models, it is possible to describe a generalized conditional probability distribution in which a particular random variable (the "target") is conditioned on a set of parent variables in such a way that when the model is converted to ground form ("unrolled"), the number of parent nodes is large and varies from one instance of the target variable to another. For example, the size of a population of mosquitos depends on the temperature and the rainfall each day since the last freeze. In one location, there might have been 19 days since the last freeze whereas in another location, there might have been only 3 days (see Figure 1(a)).

There are two main approaches to deal with this "multiple-parent" problem: aggregators and combining rules. An aggregator is a function that takes the *values* of the parent variables and combines them to produce a single aggregate value which then becomes the parent of the target variable. In the mosquito problem, we might define the total temperature and the total rainfall as aggregate variables. These are well-defined for any number of parents, and they can be computed deterministically (shown as dashed lines in Figure 1(b)). The population node then has only two parents: TotalTemp and TotalRain.

The second approach to the multiple-parent problem is to have a distribution $P(Pop \mid Temp, Rain)$ for population given a single temperature-rain pair and then combine the *distributions* from all such related pairs via a combining rule such as Noisy-OR, Noisy-AND, or Mean (see Figure 1(c)). The advantage of this method is that it can capture interactions between the *Temp* and *Rain* variables that are lost when temperature and rain are aggregated separately. In effect, the different days are "voting" about the probability of the mosquito population.

How can we learn in the presence of aggregators and combining rules? Most aggregators are deterministic and have no adjustable parameters, so they pose no additional problems for learning. However, some aggregators may have

Appearing in *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.



(a) (b) (c)

Figure 1. Three Bayesian networks describing the influence of daily temperature and rainfall on the population of mosquitos. (a) a network with no aggregation or combination rules leads to a very complex conditional probability distribution, (b) a network with separate aggregation for temperature and rainfall, (c) a network with separate prediction of the mosquito population each day followed by a combining rule to predict the overall population.

internal parameters. Suppose, for example, that we aggregated the temperatures as "degree days above 50 degrees": $DD(50) = \sum_i max(0, Temp_i - 50)$. The threshold of 50 might be learned from training data.

Learning with combining rules is more difficult, because the individual predicted target variables (e.g., Pop1, Pop2, ...) are unobserved, so the probabilistic model becomes a latent variable model. However, the latent variables are constrained to all share the same conditional probability distribution, so the total number of parameters remains small. In previous work, Koller and Pfeffer developed an expectation maximization (EM) algorithm for learning in the presence of combining rules and missing data in relational context (Koller & Pfeffer, 1997). Kersting and DeRaedt implemented a gradient descent algorithm for the same (Kersting & Raedt, 2001).

In this paper, we generalize and extend the above work to weighted combinations of individual combining rules. We present algorithms based on both gradient descent and EM. The algorithms are tested on two tasks: a folder prediction task for an intelligent desktop assistant and a synthetic task designed to evaluate the ability of the algorithms to recover the true conditional distribution.

The rest of the paper is organized as follows. Section 2 introduces the necessary background on probabilistic relational languages and the need for combining rules. Section 3 presents the two gradient descent and EM algorithms that we have designed for learning the parameters in the presence of combining rules. Section 4 explains the experimental results on a real-world dataset and on the synthetic datasets. Section 5 concludes the paper and points out a few directions for future research.

2. Probabilistic Relational Languages

In this section, we give a brief introduction to our first order conditional influence language (FOCIL), which will serve as our formalism for specifying and learning combining rules. However, we note that our learning techniques are not tied to this particular language and are applicable to other probabilistic modeling languages that support combining rules e.g., BLPs.

In the spirit of PRMs, we consider modeling domains that are described in terms of objects of various types. The type of an object determines the set of attributes or features that describe it. In addition, predicates define properties of objects (e.g., the type) and relationships among those objects. In this work, we assume that the domain of objects and relations among the objects are known and that we are interested in modeling the probabilistic relationships among the attributes of the objects.

2.1. Conditional Influence Statements

The core of our language consists of first-order conditional influence (FOCI) statements, which are used to specify probabilistic influences among the attributes of objects in a given domain. Each FOCI statement has the form:

If $\langle condition \rangle$ *then* $\langle qualitative\ influence \rangle$

where *condition* is a set of literals, each literal being a predicate symbol applied to the appropriate number of variables. A $\langle qualitative\ influence \rangle$ is of the form $X_1, \ldots, X_k$ *Qinf* Y, where the X_i and Y are of the form $V.a$, where V is a variable in *condition* and a is an object attribute. This statement simply expresses a directional dependence of the *resultant* Y on the *influents* X_i. Associated with each FOCI statement is a *conditional probability function* that specifies a probability distribution of the resultant conditioned on the influents, e.g. $P(Y|X_1, \ldots, X_k)$ for the above statement. We will use P_i to denote the probability function of the i'th FOCI statement. As an example, consider the statement,

```
If {Person(p)} then p.diettype Qinf
p.fitness,
```

which indicates that a person's type of diet influences their fitness level. The conditional probability distribution $P(p.\text{fitness} \mid p.\text{diettype})$ associated with this statement (partially) captures the quantitative relationships between the attributes. As another example, consider the statement

```
If {Student(s) , Course(c) , Takes(t,s,c)}
then s.iq, c.diff Qinf t.grade,
```

which indicates that a student's IQ and a course's difficulty influence the grade of the student in the course. Note that

since `Takes` is a many-to-many relation, we have introduced an argument t to represent the instance of the student taking a course. It can be interpreted as representing a student-course pair.

Given a fixed domain of objects and a database of facts about those objects, FOCI statements define Bayesian network fragments over the object attributes. In particular, for the above statement, the unrolled Bayesian network includes a variable for the grade of each student-course object, the IQ of each student, and the difficulty of each course. The parents of each grade variable are the IQ and difficulty attributes corresponding to the appropriate student and course. Each grade variable has an identical conditional probability table $P(t.grade|s.iq, c.diff)$—that is, the table associated with the above rule.

FOCI statements capture the same kind of influence knowledge as in PRMs, BLPs, and DAPER models, with some differences. Unlike PRMs, which only allow path expressions, the conditions can express arbitrary conjunctions of literals. In BLPs, the conditions are not syntactically separated from the influents. DAPER models attach arbitrary first-order conditions to the Bayes net arcs. FOCIL attaches them to hyper-arcs, which allows it to express conditions that relate parents to one another.

In addition, our language supports qualitative constraints such as monotonicity, saturation and synergies. Although in this paper we do not learn with these constraints, we have well-defined semantics of the constraints in FOCIL and learning algorithms for propositional models with monotonicity constraints (Altendorf et al., 2005).

2.2. Combining Rules

The following example illustrates the multiple-parent problem described in the introduction. Consider an intelligent desktop assistant that must predict the folder of a document to be saved. Assume that there are several tasks that a user can work on, such as proposals, courses, budgets, etc. The following FOCI statement says that a task and the role the document plays in that task influence its folder.

```
If {task(t), document(d), role(d,r,t)} then
     t.id,r.id Qinf d.folder.
```

Typically a document plays several roles in several tasks. For example, it may be the main document of one task but only a reference in some other task. Thus there are multiple task-role pairs $(t_1, r_1), \ldots, (t_m, r_m)$, each yielding a distinct folder distribution $P(d.folder \mid t_i.id, r_i.id)$. We need to combine these distributions into a single distribution for the folder variable. We could apply some kind of aggregator (e.g., the most frequently-occurring task-role pair) as in PRMs (Getoor et al., 2001). However, it is easy to imagine cases in which a document is accessed with low frequency across many different tasks, but these individual accesses, when summed together, predict that the document is stored in a convenient top-level folder rather than in the folder of the most frequent single task-role pair. This kind of summing of evidence can be implemented by a combining rule.

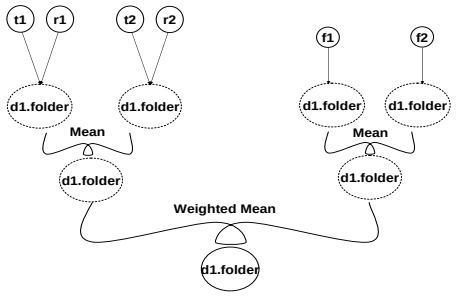


Figure 2. Use of Combining rules to combine the influences of task and role on the one hand and the source folder on the other on the folder of the current document.

In the above example, a combining rule is applied to combine the distributions due to different influent instances of a single FOCI statement. In addition, combining rules can be employed to combine distributions arising from multiple FOCI statements with the same resultant. The following example captures such a case (see Figure 2 for the unrolled network):

```
WeightedMean{
  If {task(t), doc(d), role(d,r,t)} then
     t.id, r.id Qinf (Mean) d.folder.
  If {doc(s), doc(d), source(s,d)} then
     s.folder Qinf (Mean) d.folder.}
```

Figure 3. Example of specifying combining rules in FOCIL.

The expression in Figure 3 includes two FOCI statements. One statement is the task-role influence statement discussed above. The other says that the folder of the source document of d influences d's folder. The source of a document is a document that was edited to create the current document. There can be multiple sources for a document. The distributions corresponding to different instances of the influents in the same statement are combined via the *mean* combining rule (indicated by the keyword "Mean"). The two resulting distributions are then combined with a *weighted mean* combining rule. The precise meanings of these rules are described in the next section.

3. Learning Model Parameters

In this section, we present algorithms for learning the parameters of the combining rules and the conditional probability tables (CPTs).

3.1. Problem Setup

Consider a generic influence statement S_i:

if $\langle condition \rangle$ then $X_1^i, \ldots, X_k^i$ *Qinf* Y.

We assume without loss of generality that each influence statement S_i ('rule i' for short) has k influents, X_1^i through X_k^i (which we jointly denote as $\mathbf{X}^i$), that influence the target variable. When this rule is instantiated or "unrolled" on a specific database, it generates multiple, say m_i, sets of influent instances, which we denote as $\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i$. This is shown in Figure 4. In the figure, the instantiations of a particular statement are combined with the *mean* combining rule, while the distributions of the different statements are combined via the *weighted mean* combining rule.

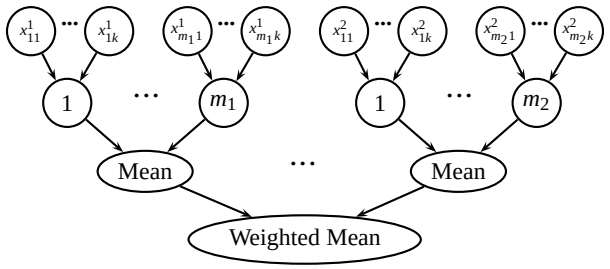


Figure 4. Unrolling of FOCI statements

The role of the combining rule is to express the probability $P_i(Y|\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i)$ as a function of the probabilities $P_i(Y|\mathbf{X}_j^i)$, one for each j, where P_i is the CPT associated with S_i. Since these instance tuples are unordered and can be arbitrary in number, our combining rule should be symmetric. For example, with the mean combining rule, we obtain:

$$P(y|\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i) = \frac{1}{m_i} \sum_{j=1}^{m_i} P_i(y|\mathbf{X}_j^i) \tag{1}$$

If there are r such rules, we need to estimate the conditional probability $P(Y|X_{1,1}^1 \ldots X_{m_r,k}^r)$. Since each rule is distinctly labeled and its instances can be identified, the combining rule need not be symmetric, e.g., weighted mean. If w_i represents the weight of the combining rule, the "weighted mean" is defined as:

$$P(Y|X_{1,1}^1 \ldots X_{m_r,k}^r) = \frac{\sum_{i=1}^r w_i P(Y|\mathbf{X}_1^i \ldots \mathbf{X}_{m_i}^i)}{\sum_{i=1}^r w_i} \tag{2}$$

We write $x_{j,1}^i, \ldots, x_{j,k}^i \equiv \mathbf{x}_j^i$ to denote the values of $\mathbf{X}_j^i$ and y to denote the value of Y. We write $\theta_{y|\mathbf{x}^i}$ to denote $P_i(y|\mathbf{x}^i)$. Note that in this case we omit the subscript j because the parameters θ do not depend on it.

3.2. Gradient Descent for Mean-squared Error

In this section, we derive the gradient-descent algorithm for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are present. Let the l^{th} training example e_l be denoted by $(\langle x_{l,1,1}^1, \ldots, x_{l,m_{l,r_l},k}^{r_l} \rangle, y_l)$, where $x_{l,j,p}^i$ is the p^{th} input value of the j^{th} instance of the i^{th} rule on the l^{th} example. The predicted probability of class y on e_l is given by

$$P(y|e_l) = \frac{1}{\sum_i w_i} \sum_i^{r_l} \frac{w_i}{m_{l,i}} \sum_j^{m_{l,i}} P_i(y|\mathbf{x}_{l,j}^i). \tag{3}$$

In the above equation, r_l is the number of rules the example satisfies, i is an index of the applicable rule, and $m_{l,i}$ is the number of instances of rule i on the l^{th} example. The squared error is given by

$$E = \frac{1}{2} \sum_{l=1}^n \sum_y (I(y_l, y) - P(y|e_l))^2. \tag{4}$$

Here y is a class label, and y_l is the true label of l^{th} example. $I(y_l, y)$ is an indicator variable that is 1 if $y_l = y$ and 0 otherwise. Taking the derivative of negative squared error with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, we get

$$\begin{aligned} \frac{-\partial E}{\partial \theta_{y|\mathbf{x}^i}} &= \sum_{l=1}^n \sum_y \Bigg[(I(y_l, y) - P(y|e_l)) \\ &\qquad \left[\frac{1}{\sum_{i'} w_{i'}} \frac{w_i}{m_{l,i}} \#(\mathbf{x}^i|e_l) \right] \Bigg]. \end{aligned} \tag{5}$$

Here $\#(\mathbf{x}^i|e_l)$ represents the number of occurrences of the tuple $\mathbf{x}^i$ in the x-instances of the i^{th} rule of example e_l. Gradient descent increments each parameter $\theta_{y|\mathbf{x}^i}$ by - $\alpha \frac{\partial E}{\partial \theta_{y|\mathbf{x}^i}}$ in each iteration. After each iteration, the parameters are normalized so that the distributions are well-defined[1].

When we adjust the weights of the rules using examples, we preserve the sum of the weights of the matching rules in each example, so that the overall sum of all weights is preserved, and the dependencies between the weights are properly taken into account when we take the derivatives. In particular, the gradients with respect to rule weights are computed as follows:

$$\frac{-\partial E}{\partial w_i} = \sum_{l=1}^n \left[\delta(e_l, i) - \frac{1}{r_l} \sum_r \delta(e_l, r) \right], \tag{6}$$

[1]Though we did not find it necessary in this problem, another approach would be to reparameterize the objective function with exponentials, thus incorporating the normalization constraint.

where $\delta(e_l, r)$ is given by

$$\sum_y (I(y_l, y) - P(y|e_l)) \frac{1}{\sum_{i'} w_{i'}} \frac{1}{m_{l,r}} \sum_j P_r(y|\mathbf{x}^r_{l,j}) \quad (7)$$

3.3. Gradient Descent for Loglikelihood

In the context of probabilistic modeling, it is more common to maximize the log likelihood of the data given the hypothesis (Binder et al., 1997). From the definition of $P(y_l|e_l)$, we can see that this is

$$L = \sum_l \log P(y_l|e_l). \quad (8)$$

Taking the derivative of L with respect to $P(y|\mathbf{x}^i) = \theta_{y|\mathbf{x}^i}$, gives

$$\frac{\partial L}{\partial \theta_{y|\mathbf{x}^i}} = \sum_l \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'} w_{i'}} \sum_i^{r_l} \frac{w_i}{m_{l,i}} \#(\mathbf{x}^i|e_l). \quad (9)$$

As before, the partial derivative of L with respect to the weights is given by

$$\frac{\partial L}{\partial w_i} = \sum_{l=1}^{n} \left[\delta(e_l, i) - \frac{1}{r_l} \sum_r \delta(e_l, r) \right]. \quad (10)$$

But now,

$$\delta(e_l, r) = \frac{1}{P_r(y_l|e_l)} \frac{1}{\sum_i w_i} \frac{1}{m_{l,r}} \sum_j P_r(y_l|\mathbf{x}^r_{l,j}). \quad (11)$$

We have found that it is important to have separate learning rates for the CPT parameters and for the combining-rule weights. In particular, the weights should be updated much more slowly than the conditional probabilities. This is because the each iteration of each example only changes a few of the CPT parameters, whereas it changes most of the weights.

3.4. Expectation-Maximization

Expectation-Maximization (EM) is a popular method to compute maximum likelihood estimates given incomplete data (Dempster et al., 1977). EM iteratively performs two steps: the *Expectation* step, where the algorithm computes the expected values of the missing data based on the current parameters, and the *Maximization* step, where the maximum likelihood of the parameters is computed based on the current expected values of the data. We adapted the EM algorithm for two-component mixture models from (Hastie et al., 2001). Consider n rules with the same resultant. Accordingly, there will be n distributions that need to be combined via a weighted mean. Let w_i be the weight for rule i, such that $\sum_i w_i = 1$.

Table 1. EM Algorithm for parameter learning in FOCIL

1. Take initial guesses for parameters θ and weights w_i
2. E Step: $\forall i$ and for each instantiation of each rule, compute the responsibilities
$$\gamma^i_{l,j} = \frac{(w_i)\frac{1}{m_{l,i}}\theta_{y|\mathbf{x}^i_{l,j}}}{\sum_{l,i',j}(w_{i'})\frac{1}{m_{l,i'}}\theta_{y|\mathbf{x}^{i'}_{l,j}}}$$
3. M Step: Compute the new parameters:
$$\forall i, \mathbf{x}^i \quad \theta_{y|\mathbf{x}^i} = \frac{\sum_{l,j} \gamma^i_{l,j}|_{y,\mathbf{x}^i}}{\sum_{y,l,j} \gamma^i_{l,j}|_{y,\mathbf{x}^i}}$$
and, if instantiations of at least two rules are present in l, compute
$$\forall i \quad w_i = \left(\sum_{l,j} \gamma^i_{l,j}\right) / n_2$$
where n_2 is the number of examples with two or more rules instantiated.
4. Continue E and M steps until convergence.

The EM algorithm for parameter learning in FOCIL is presented in Table1. In the *expectation* step, we compute the responsibility of each instantiation of each rule. The responsibilities reflect the relative density of the training points under each rule (Hastie et al., 2001). Note that we consider the weight of the current rule and the number of instantiations of the current rule while computing the responsibility of an instantiation of the current rule. In the *maximization* step, we use these responsibilities to update the CPTs. We use the responsibilities of the instantiations of an example to compute the weights if at least two rules are instantiated in the example. If an example matches less than two rules, the weights do not affect the distribution.

For instance, consider the update of $P(y_1 \mid \mathbf{x}^i)$. This is the fraction of the sum of all the responsibilities when $Y = y_1$ over all Y given $\mathbf{x}^i$. Likewise, the weight of the current rule is the fraction of the sum of the responsibilities of all instantiations of the rule over the number of examples with two or more rules instantiated.

4. Experiments and Results

In this section, we describe results on the two data sets that we employed to test the learning algorithms. The first is based on the folder prediction task, where we applied two rules to predict the folder of a document. The second data set is a synthetic one that permits us to test how well the learned distribution matches the true distribution. We present the results for both the experiments and compare them with the propositional classifiers.

4.1. Folder prediction

We employed the two rules that were presented earlier in Figure 3, combined using the weighted mean. As part of the Task Tracer project (Dragunov et al., 2005), we collected data for 500 documents and 6 tasks. The documents were stored in 11 different folders. Each document was manually assigned to a role with respect to each task with which it was associated. A document was assigned the *main* role if it was modified as part of the task. Otherwise, the document was assigned the *reference* role, since it was opened but not edited. A document is a *source* document if it was opened, edited, and then saved to create a new document or if large parts of it were copied and pasted into the new document. Since the documents could play several roles in several tasks, the number of $\langle t, r \rangle$ pairs vary[2].

We applied Gradient Descent and EM algorithms to learn both the parameters of the CPTs and the weights of the weighted mean combining rule. We employed 10-fold cross-validation to evaluate the results. Within each fold, the learned network was applied to rank the folders of the current document and the position of the correct folder in this ranking was computed (counting from 1). The results are shown in Table 2, where the counts report the total number of times (out of 500) that the correct folder was ranked 1^{st}, 2^{nd}, etc. The final row of the table reports the mean reciprocal rank of the correct folder (the average of the reciprocals of the ranks). It is clear from the table that all the three relational algorithms performed very well: almost 90% of the documents had their correct folders ranked as 1 or 2 by all three algorithms[3].

To compare these results with propositional learners, we flattened the data using as features the numbers of times each task-role pair and each source folder appears in each example. We then used Weka to run J48 and Naive Bayes algorithms on this new dataset. J48 on the flattened data also performs as well as the relational classifiers while Naive Bayes does a little worse on the same data set. All the relational algorithms attributed high weights to the second rule compared to the first (see Table 3).

To test the importance of learning the weights, we altered the data set so that the folder names of all the sources were randomly chosen. As can be seen in Table 3, with this change, the source document rule is assigned low weight by the learning algorithms with a small loss in the score.

4.2. Synthetic data set

To realistically model a complex real-world domain, it is not enough to have a good classification accuracy on a single task. To use these predictions in complex inferences, it is important to accurately model the probability distributions. To estimate the accuracy of the learned model, we constructed a synthetic data set. The data are generated using a synthetic target as defined by two FOCI statements, each of which has two influents and the same target attribute. The two influents in each rule have a range of 10 and 3 values respectively. The target attribute can take 3 values. The probability values in the distribution of the synthetic target are randomly generated to be either between 0.9 and 1.0 or between 0.0 and 0.1. This is to make sure that the probabilistic predictions on examples are not too uncertain. The rule weights are fixed to be 0.1 and 0.9 to make them far from the default, 0.5. Each example matches a rule with probability 0.5, and when it does match, it generates a number of instances randomly chosen between 3 and 10. This makes it imperative that the learning algorithm does a good job of inferring the hidden distributions both at the instance level and the rule level.

We trained the learning algorithms on 30 sets of 2000 training examples and tested them on a set of 1000 test examples. The average absolute difference between corresponding entries in the true distribution and the predicted distribution was averaged over all the test examples. Like the folder data set, we flattened the data set by using the counts

Rank	EM	GD-MS	GD-LL	J48	NB
1	349	354	346	351	326
2	107	98	113	100	110
3	22	26	18	28	34
4	15	12	15	6	19
5	6	4	4	6	4
6	0	0	3	0	0
7	1	4	1	2	0
8	0	2	0	0	1
9	0	0	0	6	1
10	0	0	0	0	0
11	0	0	0	0	5
Score	0.8299	0.8325	0.8274	0.8279	0.797

Table 2. Results of the learning algorithms on the folder prediction task. GD-MS: Gradient descent for Mean Square error; GD-LL: Gradient descent for log-likelihood; J48: Decision Tree; NB: Naive Bayes for loglikelihood.

		EM	GD-MS	GD-LL
Original data set	Weights	$\langle .15, .85 \rangle$	$\langle .22, .78 \rangle$	$\langle .05, .95 \rangle$
	Score	.8299	.8325	.8274
Modified data set	Weights	$\langle .9, .1 \rangle$	$\langle .84, .16 \rangle$	$\langle 1, 0 \rangle$
	Score	.7934	.8021	.7939

Table 3. Results of learning the weights in the original data set and the modified data set.

[2] On average, each document participated in 2 $\langle t, r \rangle$ pairs, although a few documents participated in 5 to 6 $\langle t, r \rangle$ pairs.

[3] If the algorithm were to rank the folders at random, the score would be around 0.2745.

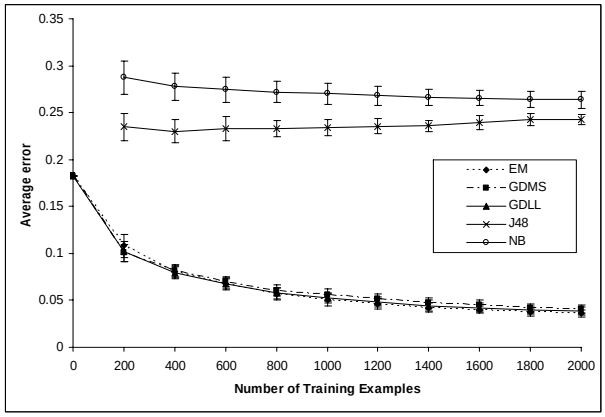


Figure 5. Learning curves for the synthetic data. EM: Expectation Maximization; GDMS: Gradient descent for Mean Square error; GDLL: Gradient descent for log likelihood; J48: Decision tree; NB: Naive Bayes.

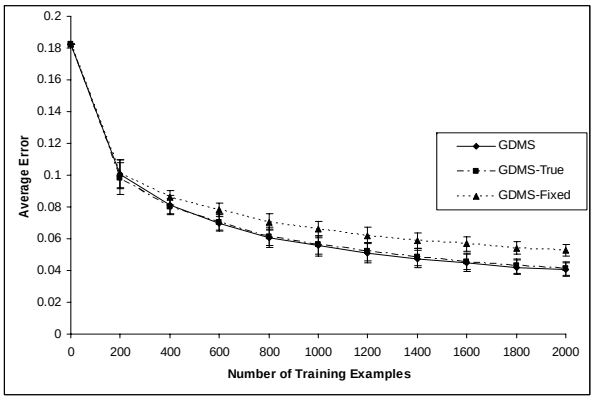


Figure 6. Learning curves for mean squared gradient descent on the synthetic data. GDMS: learning the weights; GDMS-True: gradient descent with true weights; GDMS-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

of the instances of the parents as features and used Weka to run J48 and Naive Bayes on this modified data set.

The results are presented in Figure 5. All three relational algorithms have a very low average absolute error between the true and the predicted distribution. The overlapping of the error bars suggests that there is no statistically significant difference between the algorithms' performances. On the other hand, the propositional classifiers perform poorly on this measure compared to the relational algorithms.

As with the folder data set, we wanted to understand the importance of learning the weights. Hence, for each learning algorithm, we compared three settings. The first setting is the normal situation in which the algorithm learns the weights. In the second setting, the weights were fixed at $\langle 0.5, 0.5 \rangle$. In the third setting, the weights were fixed to be their true values.

The results are presented in Figures 6, 7, and 8. There are three curves in each figure corresponding to the three settings. In all three algorithms, the first setting (weights are learned) gave significantly better error rates than the second setting (weights fixed at $\langle 0.5, 0.5 \rangle$) (Figures 6,7,8). This clearly demonstrates the importance of learning the weights. There was no significant difference between learning the weights and knowing the true weights. This shows that our algorithms effectively learn the weights of the combining rules.

5. Conclusion and Future Work

Combining rules help exploit causal independence in Bayesian networks and make representation and inference more tractable in the propositional case (Heckerman & Breese, 1994). In first order languages, they allow succinct representation and learning of the parameters of networks where the number of parents of a variable varies from one instantiation to another. They make it possible to express pairwise interactions between influences and provide a natural representation for many models of influence.

We showed that we can employ classic algorithm schemes such as gradient descent and EM to learn the parameters of the conditional influence statements as well as the weights of the combining rules. The performances of the three algorithms were quite similar on both data sets. In the folder data set, the propositional classifiers performed as well as the relational ones. This is partly because the examples in this dataset often have only one or two task-role pairs, which makes it an easier problem. In the synthetic domain, all examples have at least 3 task-role pairs, and the propositional algorithms performed poorly. The experiments also show that learning the probability model is much more difficult than learning to classify. We showed that gradient descent and EM can also learn the weights of the combining rules in addition to the CPTs of the FOCI statements.

One of our goals is to extend this work to more general classes of combining rules and aggregators including tree-structured CPTs and noisy versions of other symmetric functions. The combining rules must be "decomposable" in the sense that they involve only a small number of independent parameters that can be learned in a tractable fashion. The relationship between the aggregators and combining rules must be better understood and formalized. Efficient inference algorithms must be developed that take advantage of the decomposability of the combining rules as well as the flexibility of the first-order notation. Finally, we would like to develop more compelling applications in knowledge-rich and structured domains that can benefit from the richness of the first-order probabilistic languages.

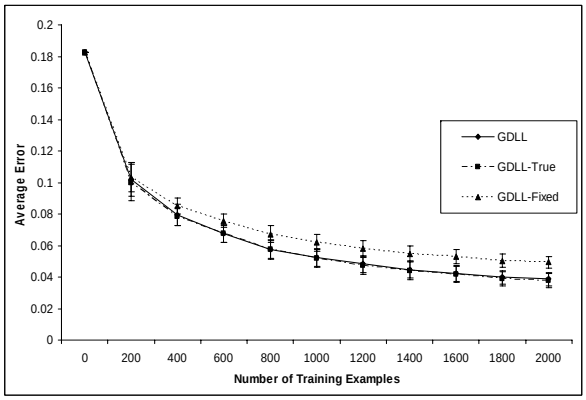


Figure 7. Learning curves for log-likelihood gradient descent on the synthetic data. GDLL: learning the weights; GDLL-True: gradient descent with true weights; GDLL-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

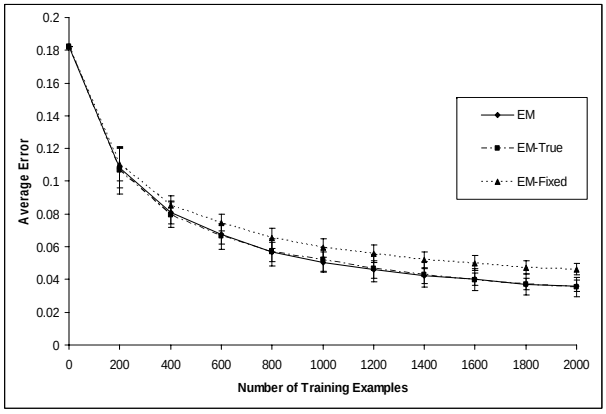


Figure 8. Learning curves for EM on the synthetic data. EM: learning the weights; EM-True: EM with true weights; EM-Fixed: EM with weights fixed as $\langle 0.5, 0.5 \rangle$.

6. Acknowledgement

The authors gratefully acknowledge support of the Defense Advanced Research Projects Agency under DARPA grant HR0011-04-1-0005. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA. We thank the reviewers for their excellent comments.

References

Altendorf, E. E., Restificar, A. C., & Dietterich, T. G. (2005). Learning from sparse data by exploiting monotonicity constraints. *Proceedings of UAI 05*.

Binder, J., Koller, D., Russell, S., & Kanazawa, K. (1997). Adaptive Probabilistic networks with hidden variables. *Machine Learning, 29*, 213–244.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society, B.39*.

Domingos, P., & Richardson, M. (2004). Markov logic: A unifying framework for statistical relational learning. *Proceedings of the SRL Workshop in ICML*.

Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L., & Herlocker, J. L. (2005). Tasktracer: A desktop environment to support multi-tasking knowledge workers. *Proceedings of IUI*.

Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). Learning probabilistic relational models. *Invited contribution to the book Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical learning*. Springer.

Heckerman, D., & Breese, J. S. (1994). *Causal independence for probability assessment and inference using bayesian networks* (Technical Report MSR-TR-94-08). Microsoft Research.

Heckerman, D., Meek, C., & Koller, D. (2004). *Probabilistic models for relational data* (Technical Report MSR-TR-2004-30).

Jaeger, M. (1997). Relational Bayesian networks. *Proceedings of UAI-97*.

Kersting, K., & Raedt, L. D. (2001). Adaptive bayesian logic programs. *Proceedings of the ILP '01* (pp. 104–117).

Kersting, K., & Raedt, L. D. (2002). *Basic principles of learning bayesian logic programs* (Technical Report 172).

Koller, D., & Pfeffer, A. (1997). Learning probabilities for noisy first-order rules. *IJCAI* (pp. 1316–1323).

Milch, B., Marthi, B., & Russell, S. (2004). Blog: Relational modeling with unknown objects. *Proceedings of the SRL Workshop in ICML*.

Muggleton, S. (1996). Stochastic logic programs. *Advances in Inductive Logic Programming* (pp. 254–264).

Ngo, L., & Haddawy, P. (1995). Probabilistic logic programming and Bayesian networks. *Proceedings ACSC95*.

Poole, D. (1993). Probabilistic Horn abduction and bayesian networks. *Artificial Intelligence, Volume 64, Numbers 1, pages 81-129*.