Learning First-Order Probabilistic Models with Combining Rules

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

Editor:

Abstract

Many real-world domains exhibit rich relational structure and stochasticity and motivate the development of models that combine predicate logic with probabilities. These models describe probabilistic influences between attributes of objects that are related to each other through known domain relationships. To keep these models succinct, each such influence is considered independent of others, which is called the assumption of "independence of causal influences" (ICI). In this paper, we describe a language that consists of quantified conditional influence statements and captures most relational probabilistic models based on directed graphs. The influences due to different statements are combined using a set of combining rules such as Noisy-OR. We motivate and introduce multi-level combining rules, where the lower level rules combine the influences due to different ground instances of the same statement, and the upper level rules combine the influences due to different statements. We present algorithms and empirical results for parameter learning in the presence of such combining rules. Specifically, we derive and implement algorithms based on gradient descent and expectation maximization for different combining rules and evaluate them on synthetic data and on a real-world task. The results demonstrate that the algorithms are able to learn both the conditional probability distributions of the influence statements and the parameters of the combining rules.

1. Introduction

New challenging application problems that involve rich relational data and probabilistic influences have led to the development of relational probabilistic models (Getoor and Taskar, 2007). The advantage of these models is that they can succinctly represent probabilistic dependencies between the attributes of different related objects, leading to sample-efficient learning. Models are succinct because parameters are shared between different instantiations of the same "rule" applied to different objects. In many cases, inference is accomplished by instantiating the rules with all possible variable bindings yielding a propositional Bayesian network. The nodes of the Bayesian network denote random variables that represent different attributes of objects or relationships between objects.

In many cases, a single parameterized rule can result in multiple instantiated sets of parents that influence a single ground target variable. More over, the number of these parent variables might change from one instance to the other. Consider, for example, the problem of modeling the spread of a disease such as West Nile virus. One might posit that the spread depends on the mosquito population in a given location. The size of the population of mosquitos in turn depends on the temperature and the rainfall of 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)).

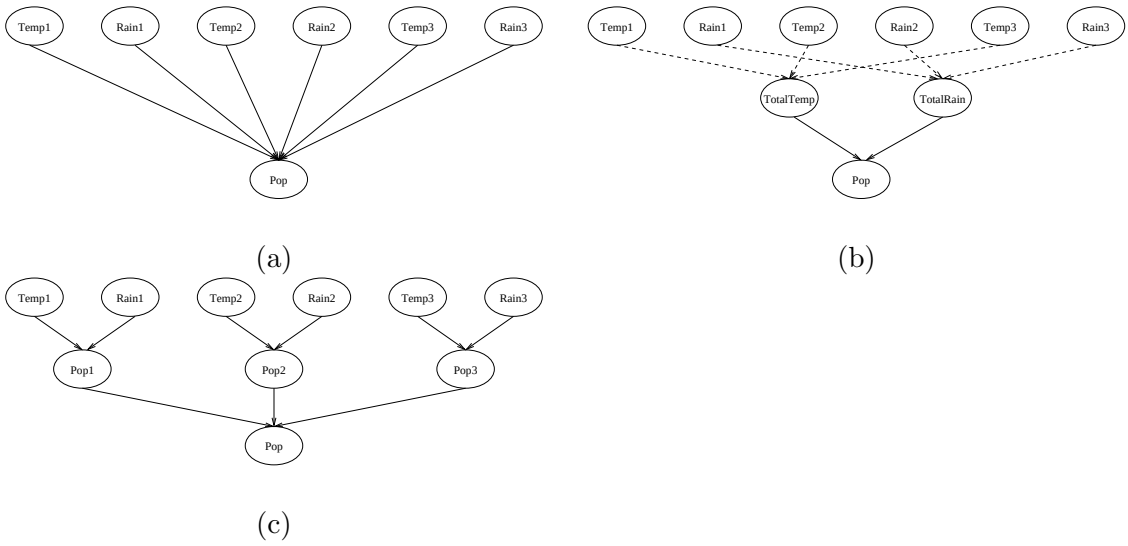

Figure 1: Three Bayesian networks describing the influence of daily temperature and rainfall on the population of mosquitos. (a) a network with no aggregation or combination rules leads to a very complex conditional probability distribution, (b) a network with separate aggregation for temperature and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

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 average 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: AverageTemp and TotalRain.

The second approach to the multiple-parent problem is to assume "independence of causal influence," (ICI) where multiple causes on a target variable can be decomposed into several independent causes whose effects are combined to yield a final value. In other words, each parent or set of related parents produces a different value for the child variable, all of which are combined using a deterministic or stochastic function. Depending on how the causes are decomposed and the effects are combined, we can express the conditional

distribution of the target variable given all the causes as a function of the conditional distributions of the target variable given each independent cause using a "decomposable combining rule."

In the above mosquito example, the mosquito population of each day is a random function of the temperature-rain pair of that day and the populations of all days may be combined into a single value by a deterministic (e.g., sum, average) or a stochastic (e.g., random choice) function (Figure 1(c)). Thus the final population distribution given the data of all days is a parameterized function of the day-wise distributions $P(Pop \mid Temp, Rain)$ computed by a combining rule such as Noisy-OR, Noisy-AND, or Mean.

The advantage of this method is that it can naturally capture the underlying fine-grained structure of the causal mechanism while keeping the inference and learning tractable. In our example, it captures the interactions between the *Temp* and *Rain* variables that are lost when temperature and rain are aggregated separately.

There is, however, an additional complication. There can be multiple independent causal influences on the same target variable which are captured by different kinds of influences. For example, spraying of pesticides and the direction of the wind at that time might also influence the mosquito population at a given place and time. The net effect of this rule and that of the previous rule will have to be combined to get a final probability of the mosquito population. This can be achieved by having another combining rule which combines the different distributions of the target variable given the parent variables in each rule.

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 internal parameters. Suppose, for example, that we aggregated the temperatures as "degree days above θ degrees": $DD(\theta) = \sum_i max(0, Temp_i - \theta)$. The appropriate threshold temperature θ might be learned from training data. See (Neville et al., 2003) for an approach to learn such parameters.

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 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 and Pfeffer, 1997). Kersting and DeRaedt implemented a gradient descent algorithm for the same (Kersting and De Raedt, 2001).

In this paper, we generalize and extend the above work to multi-level combining rules. The first level of combining rules combines the distributions of the target variable due to different instantiations of the same parameterized influence rule. The second level of combining rules operates on the results of the first level and combines the multiple distributions due to different influence rules.

We derive algorithms for learning the parameters of the distributions and the combining rules based on the gradient descent and the EM algorithms. The gradient descent is implemented using two different metrics: minimizing the mean squared error and maximizing the loglikelihood. We consider 3 types of combining rules, Mean, Weighted-Mean, and Noisy-OR. The algorithms are tested on three tasks: a folder prediction task for an intelligent desktop assistant and two synthetic tasks designed to evaluate the ability of the

algorithms to recover the true conditional probabilities. This work extends and completes our preliminary results reported in (Natarajan et al., 2005).

The rest of the paper is organized as follows. Section 2 introduces the necessary background on probabilistic relational languages and motivates 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 a concrete syntax 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 share the same underlying abstract model, e.g., Bayesian Logic Programs (BLPs). We give examples of such languages and translation of their syntax to ours in the next section.

In the spirit of Probabilistic Relational Models (PRMs) (Getoor et al., 2001), we model domains in terms of objects of various types. Each type of object has associated attributes. The attributes have values of different primitive types, e.g., integers, enumerated type, etc. There are also predicates that describe properties of objects or relationships between objects of certain types. 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 influences between the attributes of the objects. The methods are extensible to probabilistic influences between relations in a straightforward way.

2.1 Conditional Influence Statements

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

If ⟨*condition*⟩ *then* ⟨*qualitative influence*⟩

where *condition* is a set of literals, each literal being a predicate symbol applied to the appropriate number of variables. The set of literals is treated as a conjunction. A ⟨*qualitative influence*⟩ 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 that occurs 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(X)} then X.diettype Qinf X.fitness`,

which indicates that a person's type of diet influences their fitness level[1]. The conditional probability distribution $P(X.\text{fitness} \mid X.\text{diettype})$ associated with this statement (partially) captures the quantitative relationships between the attributes. As another example, consider the statement

`If {Takes(Offering,Student,Course)} then Student.iq, Course.diff Qinf Offering.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 `Offering` 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 grounded 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 $P(Offering.grade|Student.iq, Course.diff)$—that is, the table associated with the above rule.

In addition, our language supports qualitative constraints such as monotonicity (e.g., a person's consumption monotonically increases with income) and synergies (e.g., the lifestyle of a family synergistically depends on the family members' incomes). 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 {role(Doc,Role,Task)} then Task.id,Role.id Qinf Doc.folder.
```

where $role(Doc, Role, Task)$ denotes that the document Doc plays the role specified by $Role$ in $Task$. 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 $(Task_1, Role_1), \ldots, (Task_m, Role_m)$, each yielding a distinct folder distribution $P(Doc.folder \mid Task_i.id, Role_i.id)$. 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, there are usually some documents, e.g., bibliographies, that are 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

1. For the ease of notation, we denote an object's name to begin with uppercase, while the attribute of an object to begin with lowercase

most frequent single task-role pair. This kind of summing of evidence can be implemented by a combining rule.

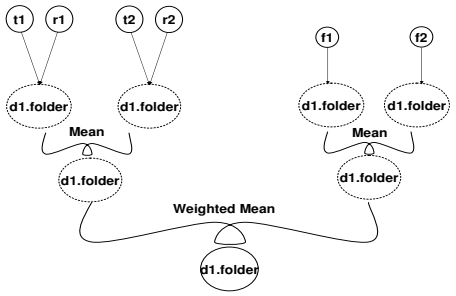

Figure 2: Use of Combining rules to combine the influences of task and role on the one hand and the source folder on the other on the folder of the current document.

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 grounded network):

```
WeightedMean{
  If {role(Doc,Role,Task)} then  Task.id, Role.id Qinf (Mean) Doc.folder.
  If {source(Src,Doc)} then Src.folder Qinf (Mean) Doc.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 *Doc* influences *Doc*'s folder. The source of a document is another document that was used 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. The two resulting distributions are then combined with the "Weighted-Mean" combining rule. The precise meanings of these rules are described in Section 2.4 through the grounding process that leads to a Bayesian network.

Our language consists of a set of *influence statements*. Each influence statement has a *resultant*, a set of *influents*, a *logical condition* and a *combining rule* to combine the influences due to the different influent instances. The set of influence statements that have the same resultant are grouped together and another combining rule is used to combine the distributions arising due to different influence statements.

The logical conditions in the influence statements specify a set of logical variables that denote objects and the relationship that must hold between the objects for the statement to

be applicable. The influents and the resultant of the influence statement correspond to the attributes of objects bound to the logical variables, and respectively represent the parents and the child nodes in the grounded Bayesian network. For instance, in the statement in Figure 3, the logical condition *role* serves to bind the logical variables Doc,Role, and Task to objects such that the document bound to Doc plays the specified role in the given task. The statement uses these bindings to declare the task and the role of the document as influents of the folder of the document.

2.3 Relationship to Other Languages

We describe our learning algorithms using the FOCIL notation. However, our learning algorithms and results are not specific to its concrete syntax. They can be applied to any language that shares its abstract model and uses combining rules to combine the results of multiple instantiated rules.

In this section, we represent several first order probabilistic models in FOCIL's syntax to show their commonalities and illustrate the generality of our learning algorithms.

Kersting and De Raedt introduced Bayesian Logic Programs (Kersting and De Raedt, 2000). BLPs combine Bayesian Networks with definite clause logic. Bayesian Logic Programs consist of two components: a qualitative component that captures the logical structure of the domain (similar to that of the Bayesian Network structure) and a quantitative component that denotes the probability distributions. An example of a BLP clause is as follows:

```
bt(X) | father(F,X), bt(F), mother(M,X), bt (M)
```

There is a CPT corresponding to this clause. In this case, the predicates $mother(M, X)$ and $father(F, X)$ would have boolean values. One could then specify the ground facts like $father(John, Tom)$ etc. The function $bt(F)$ represents the blood type of F. The above statement says that a person's blood type is a function of his father's and mother's blood types. The FOCI statement corresponding to the above BLP clause is:

```
If { mother(M,P), father(F,P)} then M.bt, F.bt Qinf P.bt
```

BLPs also use combining rules for combining the distributions due to multiple instantiations of the parent predicates. The main difference between BLPs and FOCI statements is that in the latter, the logical conditions are clearly separated from the influents. BLPs do not make this distinction in the clauses, although they *are* semantically distinguished and implemented by a separate declaration in the model.

Another representation that is closely related to both FOCIL and BLPs is Logical Bayesian Networks Fierens et al. (2005). They consist of conditional dependency clauses of the form $X|Y_1, ..., Y_k \longleftarrow Z_1, ..., Z_m$. This can be interpreted as "$Y_1, ..., Y_k$ influence X when $\langle Z_1, ...Z_k \rangle$ are true," where $Y_1, ..., Y_k$ and X are random variables and $\langle Z_1, ..., Z_m \rangle$ are logical literals. The above example of the bloodtype can be represented in LBNs as:

```
bt(X) | bt(M), bt(F) ⟵ Mother(M,X),Father(F,X)
```

More recently Getoor and Grant proposed the formalism of Probabilistic Relational Language (PRL) (Getoor and Grant, 2006). The main motivation behind this work is to represent the original work on probabilistic relational models (PRMs) (Getoor et al., 2001) in logical notation. While PRMs exclusively use aggregators to combine the influences of multiple parents, both aggregators and combining rules can be used in the PRL framework. The entities and the relationships that are represented as predicates form the logical structure of the domain. The probabilistic structure is composed of non-key attributes that form the random variables in the domain. The general structure of the influence statement is: $DependsOn(X(\alpha), Y_1(\alpha), ...Y_n(\alpha)) \longleftarrow Z(\alpha)$ and can be interpreted as "$\langle Y_1(\alpha)....Y_n(\alpha)$ influence $X(\alpha)$ when $Z(\alpha)$ is true." Consider, our bloodtype example. In PRL, we can represent it as follows:

```
DependsOn(bt(X),bt(M), bt(F)) ⟵ Mother(M,X), Father(F,X)
```

The main difference between the PRLs and LBNs lies in the fact that the PRLs allow for aggregate functions explicitly. The aggregate functions do not pose special problems for parameter learning because often they are deterministic and are given. However, inference is much more complicated with aggregate functions. In this paper we ignore aggregation and focus on combining rules. Also, in (Getoor and Grant, 2006) the authors show how to represent several kinds of uncertainties like structure uncertainty, reference uncertainty, and existence uncertainty in PRL. These extensions are out of the scope for the current paper.

Heckerman et al. introduced directed acyclic probabilistic entity-relationship (DAPER) models (Heckerman et al., 2004). DAPER models are to entity-relationship models what PRMs are to relational schema. The arcs in the DAPER models are between the attributes of the entities and relationships. The main difference with the PRMs is that the DAPER models attach arbitrary first-order conditions to the Bayes net arcs. These conditions serve to restrict the set of possible instantiations of the variables. For instance, to represent the bloodtype example, there would be arcs to the bloodtype of a person X from the bloodtypes of the person's father F and mother M with the constraints on the arcs being $Mother(M, X)$ and $Father(F, X)$. These conditions along with the arcs represent the fact that a person's bloodtype is influenced by the bloodtypes of his or her parents.

One problem with the DAPER models, however, is that they do not allow sharing of variables between the incoming arcs at the same node, and thus prohibit interaction between the influents. For instance, it is impossible to express the following rule in DAPER because the condition $role(Doc, Role, Task)$ involves both the Bayes net parents $Task$ and Doc.

```
If {role(Doc,Role,Task)} then Task.id,Role.id Qinf Doc.folder.
```

This problem can be solved by attaching the conditions jointly to all parents of a node. DAPER models allow us to specify how the distributions are combined if there are more than one possible instantiation of the free variables that satisfy the conditions.

Although the different models differ from each other in syntactic details, they all share the same underlying semantics for the core language, and express equivalent pieces of knowl-

edge. All of them also suffer from the multiple-parent problem, which can be addressed through combining rules. Thus, the algorithms discussed in this paper are relevant and applicable to all these formalisms and a few others such as Relational Bayesian Networks (RBNs) (Jaeger, 1997), Multi-Entity Bayesian Networks (MEBNs) (Laskey, 2008), and Probabilistic Logic Programs (Ngo and Haddawy, 1995).

Not surprisingly, there are also some statistical relational models for which our algorithms do not apply. For example, PRISM (Sato and Kameya, 2001) uses a representation that consists of a set of probabilistic atoms called facts, and a set of deterministic non-unit definite clauses called rules. A probability distribution is placed on the interpretations over the facts, and is extended to all literals via the minimal model semantics of definite clause programs. There is no straightforward mapping between the FOCIL representations and PRISM programs. Stochastic Logic Programs (SLPs) are very different from all the above languages since they place distributions on possible proofs of Horn programs rather than on interpretations (Muggleton, 1996). Markov Logic Networks (Domingos and Richardson, 2004) and related Conditional Random Fields (Lafferty et al., 2001) are based on undirected graphical models and significantly differ from models based on directed graphs. Markov Logic Networks, for example, are more flexible in allowing knowledge to be expressed as weighted first-order formulas, and have correspondingly harder learning and inference problems as functions of the size of the formulas.

2.4 Grounding FOCI Statements

In this section, we formally define the semantics of the FOCI statements by showing the procedure to ground them into a Bayesian network.[2] Consider a generic influence statement:

if $\langle condition \rangle$ then $X_i^1, \ldots, X_i^k$ *Qinf* Y.

For notational simplicity, we assume that the i^{th} influence statement, 'rule i' for short, has k influents, X_i^1 through X_i^k (which we jointly denote as $\mathbf{X}_i$), that influence the target variable. When this rule is instantiated or "grounded" on a specific database, it generates multiple, say m_i, sets of influent instances, which we denote as $\mathbf{X}_i^1 \ldots \mathbf{X}_i^{m_i}$. This is shown in Figure 4. In the figure, the instantiations of a particular statement are combined with the *Mean* combining rule. The distributions resulting from the different FOCI statements are combined via the *Weighted-Mean* combining rule.

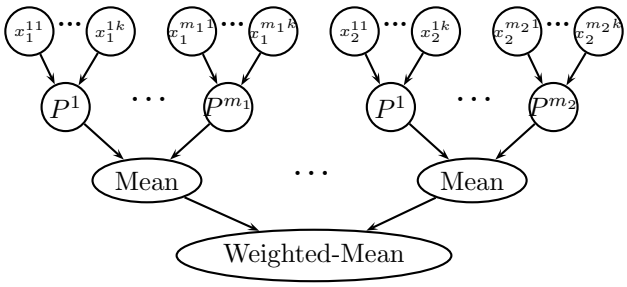

Figure 4: Grounding of FOCI statements

2. We prefer the word "grounding" to "unrolling" to describe this process, because unrolling is more frequently used to refer to instantiating dynamic Baesian networks over multiple time steps.

The role of the combining rule is to express the probability $P_i(Y|\mathbf{X}_i^1 \ldots \mathbf{X}_i^{m_i})$ as a function of the probabilities $P_i(Y|\mathbf{X}_i^j)$, one for each j, where P_i is the CPT associated with rule i. Since these instance tuples are unordered and can be arbitrary in number, our combining rule should be symmetric, i.e., its value should not depend on the order of the arguments. For example, with the mean combining rule, we obtain:

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

If there are r such rules, we need to estimate the conditional probability $P(Y|X_1^{1,1}...X_r^{m_r,k})$. 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 i^{th} combining rule, the "Weighted-Mean" is defined as:

$$P(Y|X_1^{1,1}...X_r^{m_r,k}) = \frac{\sum_{i=1}^r w_i P(Y|\mathbf{X}_i^1 \ldots \mathbf{X}_i^{m_i})}{\sum_{i=1}^r w_i} \tag{2}$$

We write $x_i^{j,1}, \ldots, x_i^{j,k} \equiv \mathbf{x}_i^j$ to denote the values of $\mathbf{X}_i^j$ 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 superscript j because the parameters θ are shared across the different instantiations of the same rule. We typically use l to index an example, and use y_l and $\mathbf{x}_l$ to denote the Y-value and the X-vector of the l^{th} example.

Note that the graph presented in Figure 4 is not a Bayesian Network, but is a tree representation of the expression for the conditional probability of the target variable given the inputs. The nodes labeled "Mean" compute the mean of the parent distributions represented by $P_1^1, \ldots, P_1^{m_1}$, etc., and the node labeled "Weighted-Mean" computes the weighted mean of its parent distributions.

Consider the equivalent Bayesian Network presented in Figure 5. This network is very similar to Figure 4. However, the nodes here represent random variables whose values are from the domain of the target variable. This Bayesian network represents the same distribution as that of Figure 4. In this Bayesian network, the values of the random variables $Y_1^1..Y_1^{m_1}$, etc. are sampled from distributions $P_1^1, \ldots, P_1^{m_1}$, etc. Then one of the parents of Y_1 is chosen randomly according to the uniform distribution and Y_1 is set to its value. This is repeated for $Y_2, \ldots, Y_r$. This ensures that the values at each of $Y_1, \ldots, Y_r$ are distributed according to the mean of the node's parent distributions. Similarly the value of the variable Y is inherited from one of $Y_1, \ldots, Y_r$ with probabilities given by the weights $w_1, \ldots, w_r$. This ensures that the final conditional distribution of Y is the weighted mean of the distributions of its parents.

Similarly, the Noisy-OR combining rule can be implemented by setting the value of Y to be a disjunction of $Y_1', \ldots, Y_r'$, where each Y_j' is a noisy version of Y_j.

The key idea behind these "decomposable" combining rules is that they can be implemented using deterministic or stochastic functions of the corresponding values of random variables. Given such a value-based implementation of the combining rules, it is possible to use the standard Bayesian network learning and inference methods on them to learn their parameters. Our implementation of the EM algorithm, described in Section 3.6, can be understood as doing exactly that.

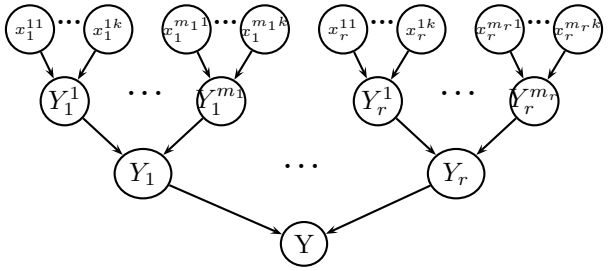

Figure 5: Value based Bayesian Network for the FOCI statements. The mean combining rule is replaced by a node that chooses a value using an uniform distribution. The Weighted-Mean rule is replaced by a node that chooses one of its parent values randomly using a biased distribution.

3. Learning Model Parameters

In this section, we present algorithms for learning the parameters of the combining rules and the conditional probability tables (CPTs). As mentioned earlier, we are using two levels of combining rules, the first to combine the multiple instances of the same rule and the second to combine the distributions due to the different rules. In this work, we use the Mean combining rule at the first level (i.e., to combine the influence due to multiple instances of the same rule) and the Weighted-Mean and the Noisy-OR at the second level (i.e., to combine the distributions due to the different rules). We present algorithms for learning in the presence of all these combining rules.

3.1 Gradient Descent Learning

Two of our learning algorithms are based on gradient descent. The idea of gradient descent training is to gradually change the parameters of the probability distributions in the direction of improved performance. We consider two measures of performance and design gradient descent algorithms for optimizing these two measures individually. The mean squared error measures the square of the difference between the true probability of a label of the example and the probability of that label predicted by the current model. Naturally, we would like to minimize the squared error. In trying to predict probabilities, it is more common to measure the performance by the loglikelihood, which is the logarithm of the likelihood of the examples under the current model. We would like to maximize the loglikelihood to improve the performance of the learner.

The generic gradient descent pseudocode for learning the parameters of the FOCI statements in the presence of combining rules is given in Figure 1. We use different step-size parameters (α and β) for the parameters and the weights respectively. It is important to have much smaller learning rates for the combining rule weights compared to those of the CPT parameters, i.e., $\beta << \alpha$. This is because each iteration of each example only changes a few of the large number of CPT parameters, whereas it changes many of the small number of weights. This is to say that for any CPT entry, the number of instances which are effected by that entry is much smaller than the total number of instances. Hence, each rule

1. Initialize the parameters θ and weights w_i randomly
2. Parameter Gradient Step: for each value of y and for all parent configurations of each rule i $\mathbf{x}_i$, compute the gradient $\frac{-\partial E}{\partial \theta_{y|\mathbf{x}_i}}$ considering each instantiation of the rule i where E is the error function that is either the negative log-likelihood or the Mean squared Error function
3. Weight Gradient Step: Compute the gradient of the weights $\frac{-\partial E}{\partial w_i}$ for each of the rule
4. Parameter Update Step: Update each parameter $\theta_{y|\mathbf{x}_i}$ by $\theta_{y|\mathbf{x}_i} = \theta_{y|\mathbf{x}_i} - \alpha \frac{\partial E}{\partial \theta_{y|\mathbf{x}_i}}$ for each value of y
5. Gradient Update Step: Update each weight w_i by $w_i = w_i - \beta \frac{\partial E}{\partial w_i}$. Normalize the weights so that the sum is preserved
6. Continue steps 2 through 5 until convergence.

Table 1: Gradient descent applied to learning parameters of FOCI statements

weight is updated much more frequently than each conditional probability and hence should be updated by smaller amount in each iteration. The gradient descent methods presented here can converge to a local optimum. In our experiments, we found that convergence to a non-global optima was not serious problem. But, in general, the use of standard techniques like random restarts would alleviate this problem. Also, the weights or the parameters could become negative when the gradient is subtracted from the current value. In these cases, we clip the values at 0.

3.2 Gradient Derivation for the Mean Squared Error for Weighted-Mean

In this section, we derive the gradient equation for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are present and are combined by the Weighted-Mean combining rule. Let the l^{th} training example e_l be denoted by $(\langle x_{l,1}^{1,1}, \ldots, x_{l,r_l}^{m_{l,r_l},k} \rangle, y_l)$, where $x_{l,i}^{j,p}$ 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

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

In the above equation, r_l is the number of rules the example e_l 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 for each example e_l 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 l indexes the examples, y is a class label, and y_l is the true label of e_l, the 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

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

The value of $P(y|e_l)$ is given in Equation 3. 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. In the second step, we assumed that the parameters for a class y only impact the probability of that class. Although this seems to ignore the fact that the conditional probabilities for all classes must add to 1, in fact this condition will be preserved by the updates as they are based on all classes. In particular, the increments for all conditional probabilities for different classes given the same evidence add up to 0 as we show below.

$$\begin{aligned}\sum_{y}\frac{-\partial E}{\partial \theta_{y|\mathbf{x}_i}} &= \sum_{l=1}^{n}\sum_{y}(I(y_l, y) - P(y|e_l))\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}_i|e_l)\right] \\ &= \sum_{l=1}^{n}\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}_i|e_l)\right]\sum_{y}(I(y_l, y) - P(y|e_l)) \\ &= \sum_{l=1}^{n}\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}_i|e_l)\right]\left(\sum_{y} I(y_l, y) - \sum_{y} P(y|e_l)\right) \\ &= \sum_{l=1}^{n}\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}_i|e_l)\right](1-1) = 0 \end{aligned} \tag{6}$$

$$\tag{7}$$

Hence each parameter will be updated as follows, leaving a proper conditional distribution as a result.

$$\theta_{y|\mathbf{x}_i} := \theta_{y|\mathbf{x}_i} + \alpha\sum_{l=1}^{n}(I(y_l, y) - P(y|e_l))\left[\frac{1}{\sum_{i'} w_{i'}}\frac{w_i}{m_{l,i}}\#(\mathbf{x}_i|e_l)\right] \tag{8}$$

The gradient with respect to rule weights is derived as follows:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l,y)-P(y|e_l))\left(\frac{-\partial}{\partial w_i}P(y|e_l)\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l,y)-P(y|e_l))\frac{-\partial}{\partial w_i}\left(\frac{1}{\sum_{i'}^{r_l} w_{i'}}\sum_{i'}^{r_l}\frac{w_{i'}}{m_{l,i'}}\sum_{j}^{m_{l,i'}} P_{i'}(y|\mathbf{x}_{l,i}^{j})\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[(I(y_l,y)-P(y|e_l))\left(\left(\frac{-\partial}{\partial w_i}\frac{1}{\sum_{i'}^{r_l} w_{i'}}\right)\sum_{i'}^{r_l}\frac{w_{i'}}{m_{l,i'}}\sum_{j}^{m_{l,i'}} P_{i'}(y|\mathbf{x}_{l,i}^{j})\right.\right. \\
&+ \left.\left.\frac{1}{\sum_{i'}^{r_l} w_{i'}}\left(\frac{-\partial}{\partial w_i}\sum_{i'}^{r_l}\frac{w_{i'}}{m_{l,i'}}\sum_{j}^{m_{l,i'}} P_i(y|\mathbf{x}_{l,i}^{j})\right)\right)\right] \\
&= \sum_{l=1}^{n}\sum_{y}\left[\frac{I(y_l,y)-P(y|e_l)}{\sum_{i'}^{r_l} w_{i'}}\left(\frac{1}{\sum_{i'}^{r_l} w_{i'}}\sum_{i'}^{r_l}\frac{w_{i'}}{m_{l,i'}}\sum_{j}^{m_{l,i'}} P_{i'}(y|\mathbf{x}_{l,i'}^{j})\right.\right. \\
&- \left.\left.\frac{1}{m_{l,i}}\sum_{j}^{m_{l,i}} P_i(y|\mathbf{x}_{l,i}^{j})\right)\right] \\
&= \sum_{l=1}^{n}\left[\left(\frac{1}{\sum_{i'}^{r_l} w_{i'}}\sum_{i'}^{r_l} w_{i'}\delta(e_l,i')\right)-\delta(e_l,i)\right], \qquad (9)
\end{aligned}$$

where

$$\delta(e_l,r) \equiv \sum_{y}\frac{I(y_l,y)-P(y|e_l)}{m_{l,r}\sum_{i'} w_{i'}}\sum_{j}^{m_{l,r}} P_r(y|\mathbf{x}_{l,r}^{j}) \qquad (10)$$

Recall that r_l is the number of applicable rules for example l. Let us define C as, $C = \frac{1}{\sum_{i'}^{r_l} w_{i'}}\sum_{i'}^{r_l} w_{i'}\delta(e_l,i')$. Since we need to *reduce* the error, we need to increment the weight w_i in proportion to $-\frac{\partial E}{\partial w_i}$. Hence, we need to increment each rule weight w_i by $\delta(e_l,i)-C$. When we update the weights, to make sure that the sum of the weights is preserved, we subtract the mean of the increments of all weights from each weight. This is the same as the "projection" procedure used by Binder et al. (1997), where, after incrementing each probability, the updated vector is projected back onto the constraint surface, where the probabilities add up to 1. This gives us the following update equation where β is the learning rate.

$$\begin{aligned}
w_i &:= w_i + \sum_{l=1}^{n}\beta(\delta(e_l,i)-C) - \beta\frac{1}{r_l}\sum_{i'}^{r_l}\big(\delta(e_l,i')-C\big) \\
&:= w_i + \sum_{l=1}^{n}\beta(\delta(e_l,i)-C+C) - \beta\frac{1}{r_l}\sum_{i'}^{r_l}\delta(e_l,i') \\
&:= w_i + \sum_{l=1}^{n}\beta(\delta(e_l,i)-\frac{1}{r_l}\sum_{i'}^{r_l}\delta(e_l,i')) \qquad (11)
\end{aligned}$$

3.3 Gradient Derivation for Loglikelihood for Weighted-Mean

As commented earlier, in the context of probabilistic modeling, it is more common to maximize the loglikelihood of the data given the hypothesis (Binder et al., 1997). Hence we turn to gradient derivation for the loglikelihood in the presence of the Weighted-Mean combining rule.

¿From the definition of $P(y_l|e_l)$, we can see that this is

$$LL = \sum_{l=1}^{n} \log P(y_l|e_l). \tag{12}$$

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

$$\frac{\partial L}{\partial \theta_{y|\mathbf{x}_i}} = \sum_{l=1}^{n} \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'} w_{i'}} \sum_{i}^{r_l} \frac{w_i}{m_{l,i}} \#(\mathbf{x}_i|e_l). \tag{13}$$

The parameters are updated by adding the gradient multiplied by a learning rate. After the update, we use the projection approach previously described to subtract the mean increment from each conditional probability, so that they add up to 1 for any distribution. The partial derivative of L with respect to the weights at each example can be derived as follows:

$$\begin{aligned}
\frac{-\partial LL}{\partial w_i} &= \sum_{l=1}^{n} \frac{\partial \log P(y_l|e_l)}{\partial w_i} \\
&= \sum_{l=1}^{n} \frac{1}{P(y_l|e_l)} \frac{\partial}{\partial w_i} \left[\frac{1}{\sum_{i'}^{r_l} w_{i'}} \sum_{i'=1}^{r_l} \frac{w_{i'}}{m_{l,i'}} \sum_{j}^{m_{l,i'}} P_{i'}(y_l|\mathbf{x}_{l,i'}^{j}) \right] \\
&= \sum_{l=1}^{n} \frac{1}{P(y_l|e_l)} \left[\frac{-1}{\left(\sum_{i'}^{r_l} w_{i'}\right)^2} \left(\sum_{i'=1}^{r_l} \frac{w_{i'}}{m_{l,i'}} \sum_{j}^{m_{l,i'}} P_{i'}(y_l|\mathbf{x}_{l,i'}^{j}) \right) + \frac{1}{\sum_{i'}^{r_l} w_{i'}} \frac{1}{m_{l,i}} \sum_{j}^{m_{l,i}} P_i(y_l|\mathbf{x}_{l,i}^{j}) \right] \\
&= \sum_{l=1}^{n} \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'}^{r_l} w_{i'}} \left[\frac{1}{m_{l,i}} \sum_{j}^{m_{l,i}} P_i(y_l|\mathbf{x}_{l,i}^{j}) - \frac{1}{\sum_{i'}^{r_l} w_{i'}} \left(\sum_{i'=1}^{r_l} \frac{w_{i'}}{m_{l,i'}} \sum_{j}^{m_{l,i'}} P_{i'}(y_l|\mathbf{x}_{l,i'}^{j}) \right) \right] \\
&= \sum_{l=1}^{n} \left[\delta(e_l, i) - \frac{1}{\sum_{i'}^{r_l} w_{i'}} \sum_{i'}^{r_l} w_{i'} \delta(e_l, i') \right]
\end{aligned} \tag{14}$$

where,

$$\delta(e_l, r) = \frac{1}{P(y_l|e_l)} \frac{1}{\sum_{i'}^{r_l} w_{i'}} \frac{1}{m_{l,r}} \sum_{j}^{m_{l,r}} P_r(y_l|\mathbf{x}_{l,r}^{j}) \tag{15}$$

Once again, the sum of the new weights is preserved by the projection method. Letting $C = \frac{1}{\sum_{i'}^{r_l} w_{i'}} \sum_{i'}^{r_l} w_{i'} \delta(e_l, i')$, Equation 11 and its justification applies exactly for the new definition of $\delta(e_l, r)$.

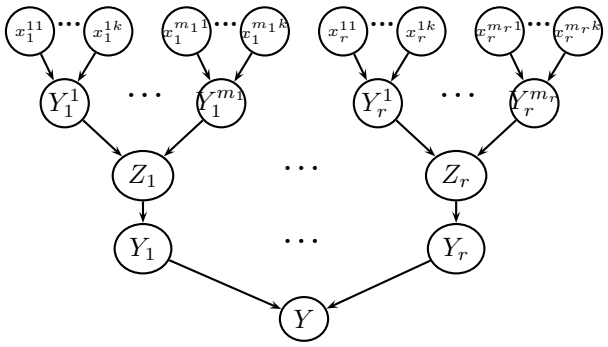

Figure 6: Value based Bayesian Network for the Noisy-OR combining rule. The nodes $Z_1 ... Z_r$ are distributed according to the mean of their parent distributions. $P(Y_i = 1|Z_i = 1) = q_i$ and $P(Y_i = 1|Z_i = 0) = 0$. Y is a deterministic OR function of $Y_1, ..., Y_r$.

3.4 Gradient Derivation for the Mean Squared Error for Noisy-OR

Intuitively, Noisy-OR models the case where there are multiple causes, any one of which can cause the target effect. However, this effect is disabled with some probability $(1 - q_i)$ independently of each other (see Figure 6). Thus, it is modeled by the following function, where all variables are binary. Z_i are the parent variables and Y is the child variable.

$$P(Y = 1 \mid Z_1, \ldots, Z_n) = 1 - \prod_i (1 - q_i)^{Z_i} \tag{16}$$

Here q_i is defined as $P(Y = 1|Z_i = 1, \forall j \neq i, Z_j = 0)$. If all Z_i's are 0, Y is zero. Otherwise, Y is 0 if all its 1-inputs are disabled, and 1 otherwise. Noisy-OR is equivalent to a network where the inputs $Z_1, \ldots, Z_r$ are transformed to $Y_1, \ldots, Y_r$, such that $P(Y_i = 1 \mid Z_i = 1) = q_i$, $P(Y_i = 1 \mid Z_i = 0) = 0$, and Y is a deterministic OR function of Y_i's. The distribution of the value at each Z_i is the mean of the distributions of the values of its parent nodes, $Y_i^1, \ldots, Y_i^k$.

We now derive the gradient equations for the mean-squared error function for the prediction of the target variable, when multiple FOCI-statements are combined by the Noisy-OR combining rule. Let the l^{th} training example e_l be denoted by $(\langle x_{l,1}^{1,1}, \ldots, x_{m_{l,r_l},k}^{l,r_l} \rangle, y_l)$. Recall that $x_{l,i}^{j,p}$ is the p^{th} input value of the j^{th} instance of the i^{th} rule of e_l. Here we exploit the Independence of Causal Influence (ICI) of noisy-OR. Since the output variable Y is 0 only when all its immediate inputs $Y_1, \ldots, Y_r$ are 0's, and their influences on Y are independent of each other, the predicted probability of class y on e_l can be decomposed as follows:

$$\begin{aligned} P(y = 1|e_l) &= 1 - \textstyle\prod_i^{r_l} \frac{1}{m_{l,i}} \sum_j^{m_{l,i}} \left[P_i(y = 0|\mathbf{x}_{l,i}^j) + (1 - q_i) P_i(y = 1|\mathbf{x}_{l,i}^j) \right] \\ &= 1 - \textstyle\prod_i^{r_l} \frac{1}{m_{l,i}} \sum_j^{m_{l,i}} \left[1 - q_i + q_i P_i(y = 0|\mathbf{x}_{l,i}^j) \right] \end{aligned} \tag{17}$$

Here 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

$$\begin{aligned} E &= \frac{1}{2}\sum_{l=1}^{n}\sum_{y}(I(y_l,y)-P(y|e_l))^2 \\ &= \frac{1}{2}\sum_{l=1}^{n}\left[(I(y_l,y=0)-P(y=0|e_l))^2+(I(y_l,y=1)-P(y=1|e_l))^2\right] \\ &= \frac{1}{2}\sum_{l=1}^{n}\left[(I(y_l,y=0)-P(y=0|e_l))^2+(I(y_l,y=1)-(1-P(y=0|e_l)))^2\right] \quad (18) \end{aligned}$$

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

$$\frac{-\partial E}{\partial \theta_{y|\mathbf{x}_i}} = \sum_{l=1}^{n}\left[(1+(I(y_l,y=0)-I(y_l,y=1))-2P(y=0|e_l))\delta(e_l)\right] \quad (19)$$

where,

$$\delta(e_l) = \left[q_i\frac{\#(\mathbf{x}_i|e_l)}{m_{l,i}}\prod_{i'\neq i}\left[\frac{1}{m_{l,i'}}\sum_{j}\left[1-q_i+q_iP_{i'}(y=0\mid \mathbf{x}^j_{l,i'})\right]\right]\right]$$

We present the gradients for the success probabilities q_i in the Appendix section.

3.5 Gradient Derivation for LogLikelihood for Noisy-OR

We now give the derivation of the gradient for loglikelihood with Noisy-OR as the combining rule. The loglikelihood LL is given by,

$$LL = \sum_{l}\log P(y_l|e_l). \quad (20)$$

where $P(y_l = 1|e_l)$ is given by,

$$P(y=1|e_l) = 1-\prod_{i}^{r_l}\frac{1}{m_{l,i}}\sum_{j}^{m_{l,i}}P_i(y=0|\mathbf{x}^j_{l,i})+(1-q_i)P_i(y=1|\mathbf{x}^j_{l,i})$$

$$P(y=1|e_l) = 1-\prod_{i}^{r_l}\frac{1}{m_{l,i}}\sum_{j}^{m_{l,i}}1-q_i+q_iP_i(y=0|\mathbf{x}^j_{l,i}). \quad (21)$$

Taking the derivative of the likelihood with respect to $P(y|\mathbf{x}_i) = \theta_{y|\mathbf{x}_i}$ we get,

$$\begin{aligned} \frac{\partial L}{\partial \theta_{y|\mathbf{x}_i}} &= \sum_{l}\frac{1}{P(y_l|e_l)}\frac{\partial P(y_l|e_l)}{\partial \theta_{y|\mathbf{x}_i}} \\ &= \sum_{l}\left[\frac{1}{P(y_l|e_l)}(-1)^y\delta(e_l)\right] \quad (22) \end{aligned}$$

where,

$$\delta(e_l) = \left[q_i \frac{\#(\mathbf{x}_i|e_l)}{m_{l,i}} \prod_{i' \neq i} \left[\frac{1}{m_{l,i'}} \sum_j \left[1 - q_i + q_i P_{i'}(y = 0 \mid \mathbf{x}^j_{l,i'}) \right] \right] \right]$$

The gradient in the above equation will have different signs corresponding to whether the final value of y is a 0 or an 1.

3.6 Expectation-Maximization for Weighted Mean

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 are computed based on the current expected values of the data. Expectation-Maximization avoids the slow update step of the gradient-based methods by directly maximizing the likelihood in each iteration. We first explain the EM algorithm for the case where we use the Weighted-Mean rule to combine the distributions due to different rules. Consider n rules with the same resultant. Accordingly, there will be n distributions that need to be combined via the Weighted-Mean. Let w_i be the weight for rule i, such that $\sum_i w_i = 1$.

The EM algorithm for parameter learning with mean and weighted mean combining rules is presented in Table 2. It is convenient to think of EM from the point of view of a value-based representation. Consider a random variable R which takes values from the indices of rules and instances, i and j. The interpretation of this random variable is that $R = (i, j)$ in example e_l, if the j^{th} instance of the i^{th} rule is responsible to generate the final y_l. In essence, R can be considered as a "multiplexer" node. Following (Hastie et al., 2001), we use call the probability $P(R = \{i, j\}|e_l)$ the "responsibility" $\gamma^j_{l,i}$. This would mean that the set of $\gamma^j_{l,i}$ over all (i, j) pairs for a given example e_l sum to 1.

In the *expectation* step, we compute the responsibilities, which reflect the distribution of the sources of the final value of Y. As we discussed in the value-based interpretation of the Bayesian network, we can think of the intermediate nodes in this network as channeling the value of one of the root (i, j) pairs to Y. The numerator represents the chances that the value of Y^j_i matches that of y_l on input $\mathbf{x}_i$, and the denominator represents the chances that the value of *some* $Y^{j'}_{i'}$ matches that of y_l. Their ratio represents the probability that the value y_l has originated at (i, j).

The goal of M-step is to find the maximum likelihood parameters that result in the expectations computed in the E-step. We do this by treating the expectations of the number of times the variable Y^j_i is inherited by the target variable Y when its corresponding input $\mathbf{x}^j_i$ takes the value $\mathbf{x}^i_{j,i}$, as a pseudocount denoted by $n(R = (i, j), Y_{i,j} = y, \mathbf{x}^j_i = \mathbf{v})$. The maximum likelihood estimates then correspond to the probability $P(Y_{i,j} = y|\mathbf{x}^j_i = \mathbf{x}^i_{j,i})$ being the $n(.)$ vector normalized over different values of y. The numerator of the equation in M-Step of Table 2 is the above expected count: $n(R = (i, j), Y_{i,j} = y, \mathbf{x}^j_i = \mathbf{x}^i_{j,i})$. The denominator is the normalizing constant.

Table 2: EM Algorithm for parameter learning in FOCIL

1. Initialize parameters θ and weights w_i arbitrarily.
2. Count N_{s_i} for each rule i over the training set.
3. Repeat until convergence
   - E Step: $\forall i$ and for each instantiation of each rule, compute the responsibilities

     $$\gamma^j_{l,i} = \frac{(w_i)\frac{1}{m_{l,i}}\theta_{y_l|\mathbf{x}^j_{l,i}}}{\sum_{l,i',j}(w_{i'})\frac{1}{m_{l,i'}}\theta_{y_l|\mathbf{x}^j_{l,i'}}}$$

     $n[i] := \sum_{l,j}\gamma^j_{l,i}$, where e_l has at least two applicable rules including i .
   - M Step: Compute the new parameters:

     $$\forall i, \mathbf{x}_i \quad \theta_{y|\mathbf{v}} = \frac{\sum_{l:y_l=y,j:\mathbf{x}^j_i=\mathbf{v}}\gamma^j_{l,i}}{\sum_{l,j':\mathbf{x}^j_i=\mathbf{v}}\gamma^{j'}_{l,i}}$$

   - Weight update:

     $t := 0$;
     Initialize w^0_i randomly for all i.
     Repeat
       For all rules i, $w^{t+1}_i := \frac{n[i]}{\sum_{S_i}\frac{n_{S_i}}{\sum_{j\in S_i} w^t_j}}$
       $t := t + 1$;
     until the weights converge

We have not found an analytical solution for the weights that maximize the likelihood. Hence, we resort to an iterative method to solve the search for weights that maximize likelihood. Let $n_{\{i,j\}}$ be the number of times rules i and j were applicable and $n^i_{\{i,j\}}$ be the number of times rule i was actually selected. Let the weights be $\langle w_1, w_2, ..., w_n\rangle$. For example, let the number of rules be 3. We will later generalize this. Let L denote the likelihood of the data. Then L is given by,

$$L = (w_1/(w_1+w_2))^{n^1_{\{1,2\}}} * (w_2/(w_1+w_2))^{n^2_{\{1,2\}}} * .... * (w_3/(w_1+w_2+w_3))^{n^3_{\{1,2,3\}}}$$

The loglikelihood (LL) is now given by,

$$\begin{aligned} LL &= n^1_{\{1,2\}}(\ln w_1 - \ln(w_1+w_2)) + n^2_{\{1,2\}}(\ln w_2 - \ln(w_1+w_2)) + \ldots + n^2_{\{2,3\}}(\ln w_2 - \ln(w_2+w_3)) + \\ &\quad n^3_{\{2,3\}}(\ln w_3 - \ln(w_2+w_3)) + n^1_{\{1,2,3\}}(\ln w_1 - \ln(w_1+w_2+w_3)) + \end{aligned}$$

$$
\begin{aligned}
& n^2_{\{1,2,3\}}(\ln w_2 - \ln(w_1 + w_2 + w_3)) + n^3_{\{1,2,3\}}(\ln w_3 - \ln(w_1 + w_2 + w_3)) \\
= \; & n[1] \ln w_1 + n[2] \ln w_2 + n[3] \ln w_3 - n_{\{1,2\}} \ln(w_1 + w_2) \\
& -n_{\{1,3\}} \ln(w_1 + w_3) - n_{\{2,3\}} \ln(w_2 + w_3) - n_{\{1,2,3\}} \ln(w_1 + w_2 + w_3) \qquad (23) \\
& \qquad (24)
\end{aligned}
$$

where $n[i]$ is the expected number of times rule i was selected over at least one other rule. Setting the partial derivative of LL with respect to w_1, w_2, and w_3 to 0's gives,

$$\frac{n[1]}{w_1} = \frac{n_{\{1,2\}}}{w_1 + w_2} + \frac{n_{\{1,3\}}}{w_1 + w_3} + \frac{n_{\{1,2,3\}}}{w_1 + w_2 + w_3}$$

etc. It is convenient to rewrite these equations as follows:

$$w_1 = \frac{n[1]}{\frac{n_{\{1,2\}}}{w_1+w_2} + \frac{n_{\{1,3\}}}{w_1+w_3} + \frac{n_{\{1,2,3\}}}{w_1+w_2+w_3}} \qquad (25)$$

Note that $n[1]$ is the number of times rule 1 was selected when there was at least one other rule that was instantiated for the current example. Hence, $n[1] = \sum_{l,j} \gamma^1_{l,j}(y_l)$, where l is indexed over all examples where there is at least one more rule applicable other than 1.

The other counts such as $n_{\{1,2\}}$ can be obtained from the data by looking at the number of examples in which rules 1 and 2 had at least one instantiation each.

We solve these equations by turning them into assignments and then updating all the weights through synchronous iteration. Generalizing the notation for arbitrary number of rules, let us use S_i to denote a subset of rules $\{1..r\}$ that includes i and at least one more rule. Then n_{S_i} denotes the number of instances in which exactly the rules in S_i are applicable with at least one instantiation. If we denote the weight for rule i at iteration t as w^t_i, we synchronously update the weights for all rules i at iteration $t+1$ using the weights at time t as in the following update equation.

$$w^{t+1}_i := \frac{n[i]}{\sum_{S_i} \frac{n_{S_i}}{\sum_{j \in S_i} w^t_j}} \qquad (26)$$

Though it might appear that the synchronous update would make EM converge slower, we have verified through experiments that it is not the case. The M-step update typically converges in a few iterations.

The responsibilities can be thought of as conditional probabilities of the hidden node values given the target variable and the inputs in the corresponding value-based Bayesian network. Note that in our problem formulation, there are two levels of combining rules. As we discussed earlier, we can think of the Mean combining rule as resulting from randomly inheriting a value of the parent node as its own value. The responsibility of a particular influent instance can be thought of as the probability of choosing that instance to transmit its value to the final target node. Similarly, the weight of each rule can be thought of as representing the probability of choosing its target value as the final value.

3.7 Expectation Maximization for Noisy-OR

Since Noisy-OR is asymmetric with respect to its values, and all its inputs are responsible for the output, we directly estimate the expectations of values of random variables of interest without using auxiliary random variables such as responsibilities. Previous work on implementing EM for Noisy-OR models includes Vomlel (2006) for propositional networks and Koller and Pfeffer (1997) for PRMs. Since our formulation involves multiple levels of combining rules, our approach is different from the prior work and is derived from first principles using the value-based network of Figure 6.

We now describe the EM procedure to learn the CPTs for Y_i^j's. In the expectation step of EM, we seek to estimate the value of Y_i^j for each i, j pair and each value of the target variable Y. First, let us consider the easy case of $Y = 0$. Since Y is a result of OR'ing its parents, each of Y's parents including Y_i must be 0.

$$
\begin{aligned}
&P(Y_i^j = 1 \mid Y = 0, \mathbf{x}) = P(Y_i^j = 1 \mid Y_i = 0, \mathbf{x}) = \alpha P(Y_i^j = 1 \mid \mathbf{x}) P(Y_i = 0 \mid Y_i^j = 1, \mathbf{x}) \\
&= \alpha P(Y_i^j = 1 \mid \mathbf{x}) \\
&\quad \Big(P(Y_i = 0 \mid Z_i = 0) P(Z_i = 0 \mid Y_i^j = 1, \mathbf{x}) + P(Y_i = 0 \mid Z_i = 1) P(Z_i = 1 \mid Y_i^j = 1, \mathbf{x}) \Big) \\
&= \alpha P(Y_i^j = 1 \mid \mathbf{x}) \frac{1}{m_i} \left((1 - q_i) + \sum_{1 \le j' \ne j}^{m_i} \left[P(Y_i^{j'} = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_i^{j'} = 1 \mid \mathbf{x}) \right] \right) \quad (27)
\end{aligned}
$$

The last line follows from expanding the Mean combining rule, noting that $P(Y_i = 0 \mid Z_i = 0) = 1$, $P(Y_i = 0 \mid Z_i = 1) = (1 - q_i)$, and $Y_i^j = 1$, and simplifying. Similarly, for $Y_i^j = 0$, we have:

$$
\begin{aligned}
&P(Y_i^j = 0 \mid Y = 0, \mathbf{x}) = P(Y_i^j = 0 \mid Y_i = 0, \mathbf{x}) = \alpha P(y_i^j = 0 \mid \mathbf{x}) P(Y_i = 0 \mid Y_i^j = 0, x) \\
&= \alpha P(Y_i^j = 0 \mid \mathbf{x}) \frac{1}{m_i} \left(1 + \sum_{1 \le j' \ne j}^{m_i} \left[P(Y_i^{j'} = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_i^{j'} = 1 \mid \mathbf{x}) \right] \right) \quad (28)
\end{aligned}
$$

We can determine α by normalizing Equations 27 and 28. Now we proceed to $Y = 1$, which is only slightly more complicated.

$$
\begin{aligned}
&P(Y_i^j = 1 \mid \mathbf{x}, Y = 1) = \alpha P(Y_i^j = 1, Y = 1 \mid \mathbf{x}) = \alpha P(Y_i^j = 1 \mid \mathbf{x}) P(Y = 1 \mid Y_i^j = 1, \mathbf{x}) \\
&= \alpha P(Y_i^j = 1 \mid \mathbf{x}) \Big(1 - P(Y = 0 \mid Y_i^j = 1, \mathbf{x}) \Big) \\
&= \alpha P(Y_i^j = 1 \mid \mathbf{x}) \left(1 - \prod_{1 \le i'}^{r} P(Y_{i'} = 0 \mid Y_i^j = 1, \mathbf{x}) \right) \\
&= \alpha P(Y_i^j = 1 \mid \mathbf{x}) \\
&\quad \left(1 - \frac{1}{m_i} \left((1 - q_i) + \sum_{1 \le j' \ne j}^{m_i} \left[P(Y_i^{j'} = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_i^{j'} = 1 \mid \mathbf{x}) \right] \right) \prod_{1 \le i' \ne i}^{r} P(Y_{i'} = 0 \mid \mathbf{x}) \right) \quad (29)
\end{aligned}
$$

The last but one step follows from the fact that the only way $Y = 0$ is if all its parents are $0'$s, and that they are all independent of each other given $\langle \mathbf{x}, Y_i^j \rangle$. The last step then expands $P(Y_i = 0 \mid Y_i^j = 1, \mathbf{x})$ by marginalizing over Z_i, expanding the Mean combining rule, and using the fact that $Y_{i'}$ is independent of Y_i^j for all $i' \neq i$.

Similarly, we estimate $P(Y_i^j = 0 \mid Y = 1, \mathbf{x})$ as follows:

$$
\begin{aligned}
&P(Y_i^j = 0 \mid \mathbf{x}, Y = 1) = \alpha P(Y_i^j = 0, Y = 1 \mid \mathbf{x}) = \alpha P(Y_i^j = 0 \mid \mathbf{x}) P(Y = 1 \mid Y_i^j = 0, \mathbf{x}) \\
&= \alpha P(Y_i^j = 0 \mid \mathbf{x}) \Big(1 - P(Y = 0 \mid Y_i^j = 0, \mathbf{x})\Big) \\
&= \alpha P(Y_i^j = 0 \mid \mathbf{x}) \\
&\quad \left(1 - \frac{1}{m_i}\left(1 + \sum_{1 \leq j' \neq j}^{m_i} \left[P(Y_i^{j'} = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_i^{j'} = 1 \mid \mathbf{x})\right]\right) \prod_{1 \leq i' \neq i}^{r} P(y_{i'} = 0 \mid \mathbf{x})\right)
\end{aligned}
\tag{30}
$$

We determine α by normalizing as usual. Given the distributions of all $P(Y_i^j \mid \mathbf{x}, Y)$, for all i, j over all examples, the maximization step is straightforward. First, we treat these probabilities as fractional counts and estimate the expected number of groundings j of rule i in which $\mathbf{X}_i^j = \mathbf{v}$ over all examples and out of these the number in which $Y_i^j = 1$. In other words, we estimate $n(Y_i^j = 1, \mathbf{x}_i^j = \mathbf{v}) = \sum_l \sum_{j:\mathbf{x}_{l,i}^j = \mathbf{v}} P(Y_i^j = 1 \mid \mathbf{x}_l, y_l)$. Here the first index l is over the examples, and the second index j is over different instances of rule i whose influents are grounded to the values $\mathbf{v}$. $\mathbf{x}_l$ and y_l respectively denote the input vector and the output label of the l^{th} example. Similarly, $n(\mathbf{x}_i^j = \mathbf{v}) = \sum_l \sum_{j:\mathbf{x}_{l,i}^j = \mathbf{v}} 1$. We then estimate $P(Y_i^j = 1 \mid \mathbf{x}_j^i = \mathbf{v})$ as $\frac{n(Y_i^j = 1, \mathbf{x}_j^i = \mathbf{v})}{n(\mathbf{x}_j^i = \mathbf{v})}$ for any instance j.

In the Appendix, we describe how to learn the parameters q_i's of noisy-OR using EM and the gradient descent.

4. Experiments and Results

In this section, we describe results on the 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 the experiments and compare them with propositional classifiers such as decision-tree learner and Naive Bayes. We do not evaluate the Noisy-OR algorithms in the folder data set as the target variable is not binary. Instead, we evaluate the Noisy-OR algorithms on a synthetic data set and present the results for the same. In this section, we first present the results on the synthetic data set for Noisy-OR and then follow it up with our experiments with the Weighted-Mean combining rule.

4.1 Synthetic Data set for Noisy-OR

To estimate the accuracy of the learned model using Noisy-OR, 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 different instances of the same rule are combined using mean and the different rules are combined using the Noisy-OR combining rule. The two influents in each rule have a range of 10 and 2 values respectively. The target attribute can take 2 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 probabilities are hard to predict and not too close to the default probability of $\frac{1}{2}$ each. 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.

The goal is to evaluate the different versions of the Noisy-OR learning algorithms on this dataset to determine the accuracy of the learned distributions. We trained the learning algorithms on 15 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 of the test examples and the predicted distribution was averaged over all the test examples. Since the gradient descent methods optimize the mean-squared error and the loglikelihood while the EM optimizes the loglikelihood, there is a need for comparing the different algorithms using the same performance metric. We have chosen the average absolute error for this purpose. We obtain the distribution over the target variable and present the results.

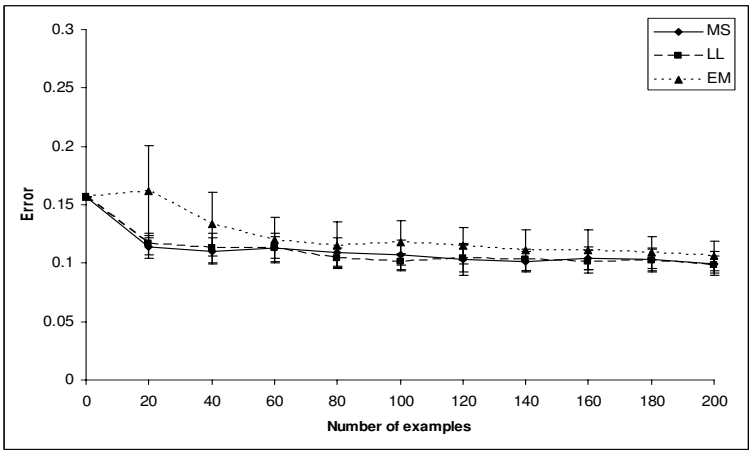

Figure 7: Learning curves for algorithms that use Noisy-OR on the synthetic data. EM: Expectation Maximization; MS: Gradient descent for mean squared error; LL: Gradient descent for log likelihood

The results are presented in Figure 7. The x-axis has the number of examples and y-axis has the average absolute error for the examples. As can be seen, all the algorithms eventually converge to almost the same error rate (there is no statistically significant difference in error rates). Initially, EM seems to perform worse, but with more training data achieves comparable performance to the gradient descent methods. We flattened the data set by using the counts of the instances of the parents as features and used Weka to run Naive

Bayes on this modified data set. The Naive Bayes algorithm had a very poor performance and a very high error rate of close to 0.42 even with about 2000 training examples. Since the performance is very poor, we omit the propositional classifier from the learning curves.

4.2 Folder prediction

We employed the two rules that were presented earlier.

```
WeightedMean{
  If {role(Doc,Role,Task)} then  Task.id, Role.id Qinf (Mean) Doc.folder.
  If {source(Src,Doc)} then Src.folder Qinf (Mean) Doc.folder.
  }
```

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 $\langle Task, Role \rangle$ pairs vary[3].

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 3, 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 (MRR) of the correct folder (the average of the reciprocals of the ranks). MRR is a standard performance metric used in information retrieval literature and it is the higher the better. It is 1 if the true folder is always ranked as the top choice. If the true folder is always ranked second, the mean reciprocal rank falls to 1/2. If it is always ranked n, the mean reciprocal rank is $1/n$. Thus the score decreases monotonically with the amount of misranking, but suffers more towards the top of the list than at the bottom.

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[4]. There is no statistically significant difference in the performance of the relational algorithms. 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 (the decision tree algorithm in weka) and Naive Bayes algorithms on this new dataset to predict the class probabilities. 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 4).

3. On average, each document participated in 2 $\langle Task, Role \rangle$ pairs, although a few documents participated in 5 to 6 $\langle Task, Role \rangle$ pairs.
4. If the algorithm were to rank the folders at random, the score would be around 0.2745.

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
MRR	0.8299	0.8325	0.8274	0.8279	0.7970

Table 3: 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.

To test the importance of learning the weights in the case with the Weighted-Mean, we altered the data set so that the folder names of all the sources were randomly chosen. As can be seen in Table 4, with this change, the source document rule is assigned low weight by the learning algorithms with a small loss in the score. In particular, the learning algorithms for the Weighted-Mean combining rule assigned a weight of 0.1 to the source rule and a weight of 0.9 to the task-role rule. The table shows that the three relational learning algorithms were able to recover from the irrelevant inputs. We then repeated the learning process on this data set, but we held the weights on the two rules fixed at 0.5. The results were much worse, because the algorithms were forced to combine the useless source probability distribution with the informative task-role distribution. The mean reciprocal rank had decreased to around 0.55 for all the learning algorithms.

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

Table 4: Results of learning the weights in the original data set and the modified data set.

4.3 Synthetic data set experiment for weighted mean

The folder prediction experiment demonstrates good performance on an interesting real-world task, but it does not tell us how close the predicted probability distribution is to the

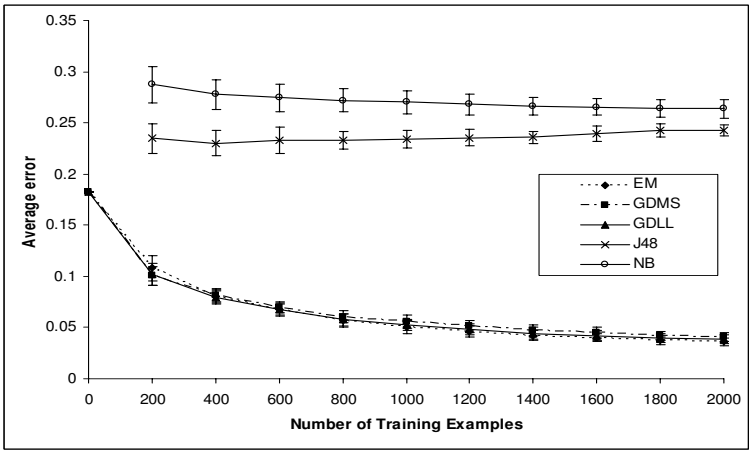

Figure 8: Learning curves for the synthetic data. EM: Expectation Maximization; GDMS: Gradient descent for mean squared error; GDLL: Gradient descent for log likelihood; J48: Decision tree; NB: Naive Bayes.

true distribution on the test 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 very similar to the one that was presented earlier. 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 different instances of the same rule are combined using mean and the different rules are combined using the Weighted-Mean combining rule. 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. As with previous experiment, this is to make sure that the probabilistic predictions on examples are hard to predict and not too close to the default probability of $\frac{1}{3}$ each. 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.

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 of the instances of the parents as features and used Weka to run J48 and Naive Bayes on this modified data set. We obtained the distribution over the target variable from Weka for these two algorithms.

The results are presented in Table 8. All three relational algorithms that use the correct model have a very low average absolute error between the true and the predicted distri-

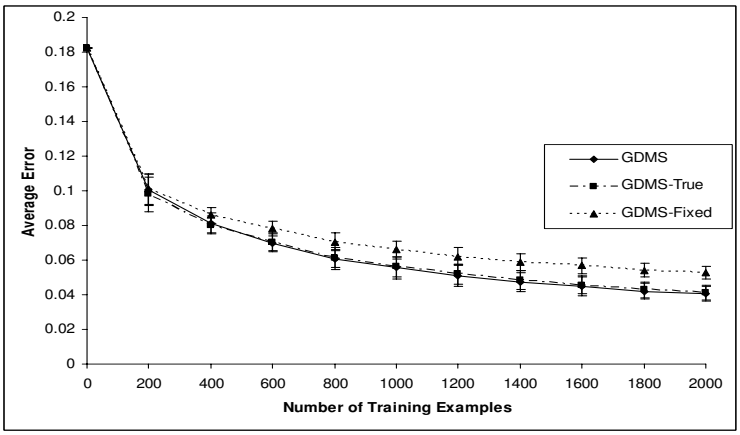

Figure 9: Learning curves for mean squared gradient descent on the synthetic data. GDMS: learning the weights; GDMS-True: gradient descent with true weights; GDMS-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

bution. 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. The performance of the relational algorithms is significantly better than the propositional classifiers. This demonstrates that though the propositional algorithms can perform reasonably well on a classification task, it is harder for them to learn the true distribution in a relational setting.

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 9, 10, and 11. 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 9,10,11). 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 with no additional cost.

In order to evaluate our algorithms on datasets with more than 2 rules, we created a synthetic dataset that is the extension of the earlier synthetic dataset. In this dataset, we used 3 rules with 2 influents each taking 10 and 3 values respectively. The weights were generated at random. Similar to the other dataset, each example matches a rule with probability 0.5, and when it does match, it generates a number of instances randomly chosen

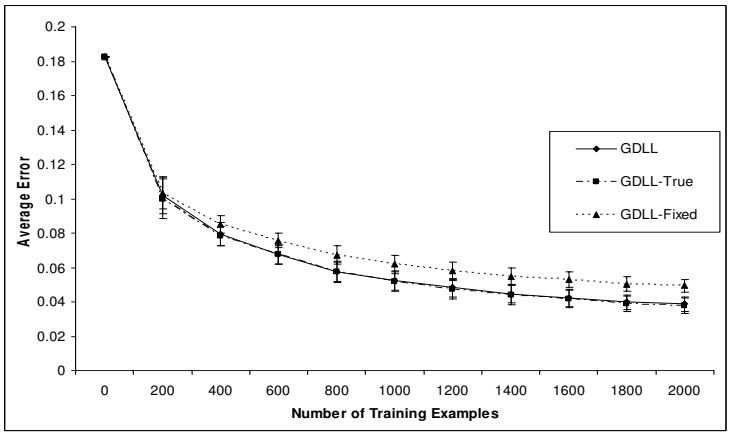

Figure 10: Learning curves for log-likelihood gradient descent on the synthetic data. GDLL: learning the weights; GDLL-True: gradient descent with true weights; GDLL-Fixed: gradient descent with weights fixed as $\langle 0.5, 0.5 \rangle$.

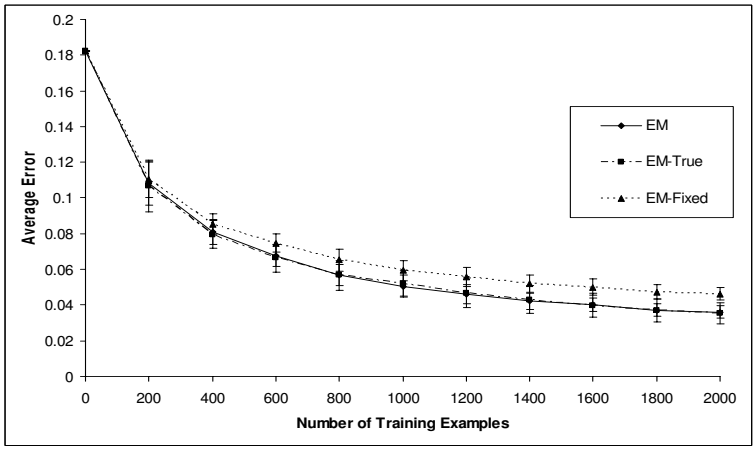

Figure 11: Learning curves for EM on the synthetic data. EM: learning the weights; EM-True: EM with true weights; EM-Fixed: EM with weights fixed as $\langle 0.5, 0.5 \rangle$.

between 3 and 10. We trained the algorithms on 30 training sets of 2000 examples each and tested them on a test set of 1000 examples.

The results of the experiment are presented in Figure 5. The absolute average error over all the test examples are averaged over the 30 datasets and presented. The results indicate that all the three algorithms have a very low average absolute error and are comparable in their performance. This demonstrates that the algorithms can be generalized to more than 2 rules effectively. The propositional classifiers have an average absolute error rate of

Algorithm	Avg Abs Error
EM	0.06985
GD-LL	0.07350
GD-MSE	0.07259

Table 5: Results of different algorithms on the 3-rule dataset.

0.32 in this domain. In our experiments, typically EM needs more iterations to converge but the gradient descent methods took more time to converge. There was no statistically significant difference in the performance of the relational algorithms.

5. Discussion and Future Work

Combining rules help exploit the independence of causal influences in Bayesian networks and make learning and inference more tractable in the propositional case (Heckerman and 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. Rather than thinking of combining rules as an alternative to aggregators, one could think of decomposable combining rules as a way of adding special structure to the aggregation problem that allows us to develop specialized methods that exploit the structure. While our gradient descent algorithms are derived directly from the combining rules perspective, our approach to the EM algorithm can be understood as aggregation-based, since it is explained in terms of the value-based network, where the aggregator function is deterministic, e.g., OR, or stochastic, e.g., random choice. Thus decomposable combining rules allow us to view the target distribution, both as a function of the input conditional distributions, and also as a consequence of decomposable causal structure in the corresponding value network. It is our hope that a wide variety of practical real world problems can be captured by a small number of simple decomposable combining rules, thus making arbitrarily complex aggregators unnecessary.

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 with the Weighted-Mean combining rule were quite similar on the folder data set. 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 to solve. 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. It would be interesting to extend this work to search over a set of combining rules to determine the one that best fits the data.

The combining rules are also well-explored in other relational probabilistic settings such as Probabilistic Horn Abduction (Poole, 1993) and Bayesian logic programs (Kersting and De Raedt, 2000). Indeed, the ability to tractably compose different influence statements or rules is necessary to build compact models. What is different in our work is that we learn the parameters in the presence of multiple influence statements each of which may have

multiple instantiations. We show that both gradient descent and EM can not only learn the CPTs of the rules but also their parameters.

There has been work on developing an EM-based learning algorithm for Noisy-OR in the propositional case (Vomlel, 2006). Diez and Galan provided a factorization for the more generalized Noisy-Max function and developed learning algorithms based on the factorization (Díez and Galán, 2003). Learning the parameters of the first-order clauses has been explored by previous researchers. Koller and Pfeffer (1997) investigated the use of EM algorithm to learn the parameters of the first-order clauses in the presence of combining rules. They use Knowledge-Based Model Construction (KBMC) to construct a ground network for each data case and employ the EM algorithm to learn the parameters of the rules.

More recently Jaeger considered a weighted combination or a nested combination of the combining rules and used a gradient ascent algorithm for optimizing the objective function (Jaeger, 2007). This technique has been applied to his formalism of Relational Bayesian Networks(RBNs)(Jaeger, 1997). The RBN is compiled into a likelihood graph that is then used for the various computations that are needed for the gradient ascent equation. The use of the likelihood graph greatly reduces the number of computations needed for the gradient computation.

This work can be naturally further extended to more general classes of combining rules and aggregators including tree-structured CPTs and noisy versions of other symmetric functions. 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, it is important to develop more compelling applications in knowledge-rich and structured domains that can benefit from the richness of the first-order probabilistic languages. Extending the SRL languages to dynamic domains with actions and utility makes them much more appropriate for compelling real-world applications. Some work has already begun in this direction (Natarajan et al., 2007).

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 and pointing out important corrections in the paper. We also thank Eric Altendorf and Angelo Restificar for their assistance in an earlier version of the paper.

References

Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from sparse data by exploiting monotonicity constraints. In *Proceedings of UAI 05*, 2005.

John Binder, Daphne Koller, Stuart Russell, and Keiji Kanazawa. Adaptive Probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997. ISSN 0885-6125.

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

F. J. Díez and S. F. Galán. Efficient computation for the noisy MAX. *International Journal of Approximate Reasoning*, 18:165–177, 2003.

Pedro Domingos and Mark Richardson. Markov logic: A unifying framework for statistical relational learning. In *Proceedings of the SRL Workshop in ICML*, 2004.

Anton N. Dragunov, Thomas G. Dietterich, Kevin Johnsrude, Matt McLaughlin, Lida Li, and Jon L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of IUI*, 2005.

Daan Fierens, Hendrik Blockeel, Maurice Bruynooghe, and Jan Ramon. Logical bayesian networks and their relation to other probabilistic logical models. In *Proceedings of ILP*, 2005.

Lise Getoor and John Grant. PRL: A probabilistic relational language. *Mach. Learn.*, 62 (1-2):7–31, 2006.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.

Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. *Invited contribution to the book Relational Data Mining, S. Dzeroski and N. Lavrac, Eds.*, 2001.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

David Heckerman and John S. Breese. Causal independence for probability assessment and inference using bayesian networks. Technical Report MSR-TR-94-08, Microsoft Research, 1994.

David Heckerman, Christopher Meek, and Daphne Koller. Probabilistic models for relational data. Technical Report MSR-TR-2004-30, March 2004.

Manfred Jaeger. Relational Bayesian networks. In *Proceedings of UAI-97*, 1997.

Manfred Jaeger. Parameter learning for relational bayesian networks. In *Proceedings of the International Conference in Machine Learning*, 2007.

Kristian Kersting and Luc De Raedt. Adaptive bayesian logic programs. In *Proceedings of the ILP '01*, pages 104–117, 2001.

Kristian Kersting and Luc De Raedt. Bayesian logic programs. In *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 2000.

Daphne Koller and Avi Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI*, pages 1316–1323, 1997.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

Kathryn Blackmond Laskey. MEBN: A language for first-order bayesian knowledge bases. *Artif. Intell.*, 172(2-3):140–178, 2008.

Stephen Muggleton. Stochastic logic programs. In *Advances in Inductive Logic Programming*, pages 254–264, 1996.

Sriraam Natarajan, Prasad Tadepalli, Eric Altendorf, Thomas G. Dietterich, Alan Fern, and Angelo Restificar. Learning first-order probabilistic models with combining rules. In *Proceedings of the International Conference in Machine Learning*, 2005.

Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. A relational hierarchical model for decision-theoretic assistance. In *Proceedings of 17th Annual International Conference on Inductive Logic Programming*, 2007.

Jennifer Neville, David Jensen, Lisa Friedland, and Michael Hay. Learning relational probability trees. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 625–630, 2003.

Liem Ngo and Peter Haddawy. Probabilistic logic programming and Bayesian networks. In *Proceedings ACSC95*, 1995.

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

Taisuke Sato and Yoshitaka Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, pages 391–454, 2001.

Jiří Vomlel. Noisy-or classifier: Research articles. *Int. J. Intell. Syst.*, 21(3):381–398, 2006.

Appendix

In this section, we show the equations for estimating the noisy-OR parameters, q_i. We now present the estimation through EM. First, we show the estimation step for $P(Y_i, Z_i = 1 \mid \mathbf{x}, Y)$ for different values of Y and Y_i. We first note the following:

$$P(Y_i, Z_i = 1 \mid \mathbf{x}, Y) = \frac{P(Y \mid Y_i, \mathbf{x}) P(Y_i \mid Z_i) P(Z_i = 1 \mid \mathbf{x})}{P(Y \mid \mathbf{x})} \tag{31}$$

$$P(Y = 0|\mathbf{x}) = \prod_i \frac{1}{m_i} \sum_{j=1}^{m_i} P(Y_i^j = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_i = 1 \mid \mathbf{x}) \tag{32}$$

$$P(Y = 1|\mathbf{x}) = 1 - P(Y = 0|\mathbf{x}) \tag{33}$$

Now we estimate the desired conditional distributions using the first equation above.

$$P(Y_i = 1, Z_i = 1 \mid \mathbf{x}, Y = 0) = 0 \tag{34}$$

$$P(Y_i = 1, Z_i = 1 \mid \mathbf{x}, Y = 1) = \frac{q_i}{P(Y = 1 \mid \mathbf{x})} \frac{q_i}{m_i} \sum_{j=1}^{m_i} P(Y_i^j = 1 \mid \mathbf{x}) \tag{35}$$

$$P(Y_i = 0, Z_i = 1 \mid \mathbf{x}, Y = 0) = \frac{1 - q_i}{P(Y = 0 \mid \mathbf{x})} \frac{1}{m_i} \sum_{j=1}^{m_i} P(Y_i^j = 1 \mid \mathbf{x})$$

$$\prod_{1 \leq i' \neq i} \frac{1}{m_{i'}} \sum_{j=1}^{m_{i'}} P(Y_{i'}^j = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_{i'} = 1 \mid \mathbf{x}) \tag{36}$$

$$P(Y_i = 0, Z_i = 1 \mid \mathbf{x}, Y = 1) = \frac{1 - q_i}{P(Y = 1 \mid \mathbf{x})} \frac{1}{m_i} \sum_{j=1}^{m_i} P(Y_i^j = 1 \mid \mathbf{x})$$

$$\left(1 - \prod_{1 \leq i' \neq i} \frac{1}{m_{i'}} \sum_{j=1}^{m_{i'}} P(Y_{i'}^j = 0 \mid \mathbf{x}) + (1 - q_i) P(Y_{i'} = 1 \mid \mathbf{x}) \right) \tag{37}$$

In the maximization step, we assign q_i to $\frac{n(Y_i=1, Z_i=1)}{n(Z_i=1)}$, where $n(Y_i = 1, Z_i = 1)$ is the expected number of examples where both Y_i and Z_i are 1 and $n(Z_i = 1)$ is the expected number of examples where $Z_i = 1$. These numbers are estimated as follows. Let $\mathbf{x}_l$ be the input vector of the l^{th} example and y_l be its Y-label.

$$n(Y_i = 1, Z_i = 1) = \sum_{l=1}^{n} P(Y_i = 1, Z_i = 1 \mid \mathbf{x}_l, y_l) \tag{38}$$

$$n(Z_i = 1) = \sum_{l=1}^{n} P(Y_i = 1, Z_i = 1 \mid \mathbf{x}_l, y_l) + P(Y_i = 0, Z_i = 1 \mid \mathbf{x}_l, y_l) \tag{39}$$

$$\tag{40}$$

Now, we present the gradients for the mean-squared error and loglikelihood with respect to q_i.

For the mean-squared error case, taking the derivative of Equation 18 with respect to q_i we get,

$$\frac{-\partial E}{\partial q_i} = \sum_{l=1}^{n} \left[(1 + (I(y_l, y = 0) - I(y_l, y = 1)) - 2P(y = 0 | e_l)) \delta_{q_i}(e_l) \right] \tag{41}$$

$\delta_{q_i}(e_l)$ is given by

$$\delta_{q_i}(e_l) = \left[\frac{\sum_j^{m_{l,i}} \left[P_i(y = 0 | x_{l,i}^j) - 1 \right]}{m_{l,i}} \Pi_{i' \neq i} \left[\frac{1}{m_{l,i'}} \sum_j \left[1 - q_i + q_i P_{i'}(y = 0 \mid x_{l,i'}^j) \right] \right] \right]$$

For the loglikelihood, taking the derivative of Equation 21 with respect to q_i we get,

$$\begin{aligned}
\frac{\partial L}{\partial q_i} &= \sum_l \frac{1}{P(y_l|e_l)} \frac{\partial P(y_l|e_l)}{\partial \theta_{y|\mathbf{x}_i}} \\
&= \sum_l \left[\frac{(-1)^y}{P(y_l|e_l)} \left[\frac{\sum_j^{m_{l,i}} \left[P_i(y=0|x_{l,i}^j) - 1 \right]}{m_{l,i}} \Pi_{i' \neq i} \left[\frac{1}{m_{l,i'}} \sum_j \left[1 - q_i + q_i P_{i'}(y=0 \mid x_{l,i'}^j) \right] \right] \right] \right]
\end{aligned}$$