Efficient Learning of Statistical Relational Models

by

Tushar Khot

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2014

Date of final oral examination: 08/18/2014

The dissertation is approved by the following members of the Final Oral Committee:
Jude Shavlik, Professor, Computer Sciences
Sriraam Natarajan, Assistant Professor, Informatics And Computing
David Page, Professor, Biostatistics and Medical Informatics
AnHai Doan, Professor, Computer Sciences
Mark Craven, Professor, Computer Sciences

To Aai and Baba

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Jude Shavlik, for mentoring and supporting me throughout my PhD. His guidance during my graduate career has been invaluable and helped me immensely to improve my technical knowledge and communication skills. I have learned a lot about my field as well as machine learning, in general, for which I will always be thankful.

I would also like to thank my co-advisor, Sriraam Natarajan, for patiently guiding me through my graduate career. His vast knowledge about the field has always inspired me to keep reading. He has been my friend and mentor, shaping me into the researcher I am today.

I would like to thank my committee members: David Page, AnHai Doan and Mark Craven for their inputs on my work. My first machine learning course was under David Page, which inspired my interest in first-order logic and eventually led to my work. I have learned a lot about various topics from my committee members through our discussions. I want to thank them for their feedback and providing a new perspective on my work.

My collaborators have made all of the work presented in this thesis possible and for that I am very thankful. The one collaborator who has been most influential and essential for my research is Kristian Kersting. Through many of our discussions in conferences and conference calls, I have learned a lot about the field, but more importantly about how to critically analyze problems and potential solutions. I would also like to thank Chris Ré for guiding and helping me through the problems with scaling our approaches, especially for the large-scale information-extraction task. His guidance during my prelims also helped forge the direction of my research in the final years. Although not directly involved in my graduate career, I want to thank Dr. Hemalatha Thiagarajan of NIT

Trichy for challenging me and inspiring me during my undergraduate career, which led me to pursue my PhD.

I am thankful to the smart people from the Machine Learning group in University of Wisconsin-Madison. Specifically, I want to thank my office mate, Gautam Kunapuli for the great technical discussions (and non-technical too). I would like to thank my friends in Madison, especially Mohit Saxena and Sabareesh Subramaniam, for supporting me during my graduate career. I am also grateful to my two room mates, Balaji Gopalan and Phillip Odom, for putting up with me. I would like to thank Indiana University, Bloomington for hosting me in my final year. I would also like to acknowledge my friends from Bloomington who made the last year of my PhD truly memorable.

Last, but not the least, I would like to thank my family for supporting me and motivating me. Aai and Baba have supported my every decision and I am grateful of their trust in me. Parna Tai and Dipali Tai have pushed me to try harder and kept me motivated, for that I will always be grateful.

I gratefully acknowledge the support of the funding agencies that have supported my education at University of Wisconsin-Madison. My research has been funded by the DARPA Machine Reading Program and Deep Exploration and Filtering of Text Program under the Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181 and FA8750-13-2-0039 respectively. I am also grateful to Dr. Tanimoto for sponsoring my Alumni Scholarship, which helped me in my early graduate career.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Machine Learning has been successfully applied to many prediction problems in varying domains such as molecular chemistry, medical diagnosis, social networks, and information extraction. However standard machine learning techniques assume that the examples or objects are independent of each other and have the same number of features or attributes. In many domains, the objects can be inter-related where predictions on one object can influence the predictions on related object (e.g., predicting hereditary diseases). Moreover the examples can have different number of features (e.g., patients with different number of tests) limiting applicability of the standard fixed-length feature vector representation.

To build probabilistic models over such structured data, Statistical Relational Learning (SRL) methods have been proposed, which combine first-order logic representation with probabilities. Most applications of SRL use an expert-defined model and learn the parameters from data. Since such a model may not be known or easy to specify, structure-learning approaches have been developed for SRL models. Due to the high expressivity of these models, there is a large space of possible structures and as a result, most structure-learning approaches can be computationally intensive.

I present a boosting-based approach that learns multiple weak rules of thumb instead of a single complex model. My approach, based on functional-gradient boosting, learns the structure as well as the parameters of the model simultaneously. I extend the basic relational functional-gradient boosting (RFGB) approach to two different SRL models: Relational Dependency Networks and Markov Logic Networks. I also experimentally demonstrate that my approach can learn more accurate models in a fraction of the time as compared to state-of-the-art learning approaches.

To further increase the applicability of my approach, I extend RFGB to handle missing data by deriving an Expectation-Maximization approach for relational models. The Expectation-step computes the expected values of missing data and the Maximization-step performs functional-gradient boosting to update the current model based on the expected values. I empirically demonstrate that the EM approach can learn more accurate model than the traditional closed-world assumption of assuming unobserved data to be false.

One of the issues with applying my approach to NLP tasks is the lack of negatively labeled examples, needing an approach for learning from only one class of examples (one-class classification). Since my EM approach can not handle completely missing values of one class, I present a non-parametric approach for relational one-class classification. I develop a tree-based relational distance measure that can be learned to directly maximize the one-class classification performance. My approach also has parallels to standard one-class classification techniques and can be used for propositionalization.

Apart from learning models, this thesis also explores knowledge representation in MLNs. As domain knowledge may be defined using combination functions, there is a need to be able to translate them into MLN rules to utilize this knowledge. I present an approach that can convert multilevel combination functions along with their corresponding parameters into four classes of MLN clauses. I present an algorithm for converting two combination functions: noisy-or and weighted-mean, and show the correctness of my transformation. Empirically, I demonstrate that combination functions can improve the accuracy of the model with a small number of rules.

Finally this thesis shows how RFGB can be used for Alzhiemer's disease prediction from MRI images as well as to augment expert rules for temporal relation extraction. I present my approach for a large-scale novel relation extraction task, where I process terabytes of streaming data to detect changes in extracted relations.

Overall, this thesis presents multiple structure-learning approaches for SRL, starting from a boosting-based algorithm, which is extended to handle missing values via EM. Next, I present a structure-learning approach for one-class classification by learning a relational distance metric. I present application of these structure-learning approach on multiple SRL datasets and real-world tasks. Apart from structure learning, this thesis presents an approach to convert knowledge from directed models to MLNs and presents my work on various challenging applications.

1 INTRODUCTION

Supervised Machine Learning (ML), very broadly, can be viewed as predicting values about instances given other instances with these values known (Mitchell, 1997). For example, predicting thunderstorms on a given day based on information from other days or predicting cardiac risks for a patient based on medical histories of other patients are supervised ML tasks. The instances in machine learning (a day or a patient) are generally called examples. The examples with the known target value (thunderstorms or cardiac risks) are called training examples and the examples for which the value is being predicted are called test examples. The task of supervised machine learning can be defined as:

Given: Training examples with known target values.
Do: Learn a model to predict the target values on unseen test examples.

Machine learning uses information about the training examples (called features) to learn a function from the feature values to the target value. For example to predict the cardiac risks for a patient, a ML model learns a function from features such as the age, weight, cholesterol levels and blood pressure to the probability of a cardiac arrest. This learned function can then be used on a different patient to predict his chance of a cardiac arrest. The supervised machine learning task can be further detailed as:

Given: Training examples with known target values and a set of features.
Do: Learn a function from the features to the target value as the model. Use the model to predict the target values on test example given its features.

Traditional statistical machine learning has been successfully applied to many real-world problems, such as medical diagnosis, document classification, and web search. Most of the approaches used to solve these problems assume the data is simple in its representation. Consider a sample university domain where we want to predict the satisfaction of every student. Table 1.1 shows the examples of students from such a domain. Each column corresponds to a feature of the student and we assume all students have the same features, albeit with possibly different values.

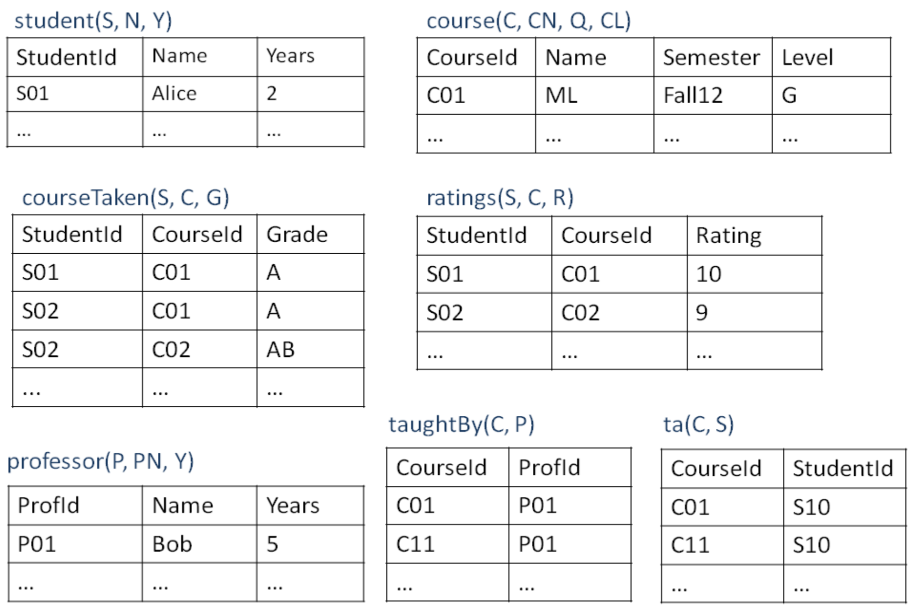
SID	CourseGrade1	CourseLevel1	CourseGrade2	CourseLevel2	...	Satisfied
S01	AB	Grad	A	UnderGrad	...	True
S02	AB	Grad	B	Grad	...	True
S03	AB	UnderGrad	C	Grad	...	False

...

Table 1.1: Traditional example representation for machine learning. Each row shows the levels and grades of the courses taken by a given student as well as whether the student is satisfied with his or her education.

However generally data is not available as fixed-length vectors as shown in the table. Students take different numbers of courses and courses have different numbers of properties. Figure 1.1 illustrates the complexity of real-world data in an university domain. As shown in this figure, data is stored across multiple relations or tables and conversion of such data into a fixed-length vectors may be non-trivial. Given such data with multiple relations and inter-related instances, the problem statement needs to be changed to

Given:	Training examples with known target values *and the relationships between the objects in the domain*.
Do:	Learn a function from the features of the example and *its related objects* to the target value. Use the model to predict the target value on the test example given the features of the example *and its related objects*.



student(S, N, Y)

StudentId	Name	Years
S01	Alice	2
...	...	...

course(C, CN, Q, CL)

CourseId	Name	Semester	Level
C01	ML	Fall12	G
...	...	...	...

courseTaken(S, C, G)

StudentId	CourseId	Grade
S01	C01	A
S02	C01	A
S02	C02	AB
...	...	...

ratings(S, C, R)

StudentId	CourseId	Rating
S01	C01	10
S02	C02	9
...	...	...

taughtBy(C, P)

CourseId	ProfId
C01	P01
C11	P01
...	...

ta(C, S)

CourseId	StudentId
C01	S10
C11	S10
...	...

professor(P, PN, Y)

ProfId	Name	Years
P01	Bob	5
...	...	...

Figure 1.1: Multi-relational data in an university domain. Students take different number of courses and professors teach different number of courses. Each relation or table is represented here as a first-order logic literal shown above each table.

First-order logic can handle such multi-relational data, for example, by converting each relation into a predicate. In Figure 1.1, the predicates corresponding to each relation are shown above the tables. The problem of learning in relational data can be viewed as learning a first-order hypothesis that would entail most of the positive examples and few of the negative examples. The hypothesis contains first-order logic rules such as $student(S, N, Y), courseTaken(S, C, \text{“A+”}) \rightarrow satisfied(S)$ which predicts that a student is satisfied if he gets an “A+” grade in some course. The field of Inductive Logic Programming (ILP) has developed various techniques to learn such hypothesis (Muggleton and Raedt, 1994). Hence learning in ILP can be viewed as learning the *structure* of the rules to predict the target relation whereas most traditional machine learning methods (e.g., logistic regression) learn the *weights* corresponding to each feature in the feature vector.

A drawback of traditional ILP approaches is that the clauses in a learned model always return Boolean predictions. Typically, there are no confidence measures associated with the learned rules or the examples during inference. This problem is addressed by introducing probabilities in first-order logic, which is one view of the field of Statistical Relational Learning (SRL; Getoor and Taskar 2007). By associating weights with each rule to indicate the confidence measure associated with that rule, one can now compute the probability of an example being true. Similar to ILP, the problem of learning models in SRL then can be viewed as learning the rules, but now also includes learning the weights of the rules in such models.

Another view of SRL is that it introduces first-order logic to statistical machine learning. For example, consider a Bayesian Network (Pearl, 1988) with each node being a predicate. The problem of learning in such models can be viewed as learning the parents for each node (*structure*) and the conditional distributions on each node (*parameters*).

Additionally, SRL models can jointly infer the class of a set of examples rather than assuming independence across examples. On tasks such as webpage or paper classification, this property of SRL allows the model to ensure that webpages or papers belong to a class similar to the ones they link or cite. It has been shown that this form of *collective classification* (classifying a set of examples collectively) helps improve the predictive accuracy of the model (Neville and Jensen, 2007). Although I do not discuss collective classification further, it is an important property of SRL models that is worth noting.

1.1 Boosted Statistical Relational Learning

SRL models allow us to compactly model multi-relational datasets using first-order logic. However, these models typically have a much more complex structure-learning task than propositional models. For example, if we are learning the structure for a Markov network with two features a and b, there is one possible factor[1] connecting a and b with four possible parameters (one for every Boolean assignment). On the other hand, learning the structure with two predicates $a(X)$ and $b(X)$ could result in two factors : $< a(X), b(X) >$ and $< a(X), b(Y) >$. One can see that the number of possible factors grows when we have more than one argument for a predicate (e.g., $a(X), b(Y, Z)$), allow constants in the factors (e.g., $a(X), b(Y, c1)$), and when we start considering existential conditions in the factor (e.g., $\exists Y a(X), b(Y, c1)$).

Recently, there have been some advances in structure learning for SRL, especially in the case of Markov Logic networks (Mihalkova and Mooney, 2007; Kok and Domingos, 2009, 2010; Khot et al., 2011). Due to the large space of possible structures, these approaches use greedy search techniques to learn the model. For every candidate clause that is considered for addition to the current model, these approaches re-learn the parameters of the entire model and then decide whether adding the candidate clause improves the score being optimized. This thesis presents our boosting-based structure-learning technique that can learn the structure as well as the parameters simultaneously. We learn a set of relational regression trees which have a closed-form solution for the parameters at each leaf and do not require any re-learning of the parameters. We show how this technique can be used to efficiently learn the structure for two popular SRL models. We also extend this approach to handle missing data using structural EM.

[1]A factor is a function over a set of variables that returns a real number for every possible set of values taken by the variables.

1.2 Thesis Statement

My thesis investigates the claim:

> *Using boosting to learn the structure of statistical relational models will produce more accurate models in a fraction of the time than the current state of the art. Sequential update of the models can also efficiently handle missing values and learning from only examples of one class.*

Specifically, I present a functional-gradient boosting (Friedman, 2001) approach for learning the structure of two relational models. I investigate the predictive performance of our approach on various datasets and the applicability of our approach on real-world tasks ranging from information extraction to Alzheimer's prediction. Since our boosting approach sequentially updates the model, our approach can be extended to handle missing data by updating the structure based on the expected values of the missing data. For learning from examples with only one class label, I present an approach which sequentially updates a distance measure so as to spread out the examples while keeping examples of the same class close to each other.

1.3 Thesis Outline

My thesis is organized as follows:

- **Chapter 2** provides technical background necessary to present my work. It begins with a brief introduction to first-order logic and probabilistic graphical models in Section 2.1. This section sets up the basics for presenting SRL models in the subsequent section. Following it, I present the two major learning problems in SRL: parameter and structure learning as well as the associated challenges and related work. I explain Friedman's functional-gradient boosting (FGB; Friedman 2001), which forms the basis of most of my work, in Section 2.4. In the final section, I present details about the evaluation approach used in this thesis and few commonly used datasets.

- **Chapter 3** describes the basic algorithm of our relational functional-gradient boosting (RFGB) approach, which uses FGB to learn structure of relational models. I then show how RFGB can be applied to learn the structure of Relational Dependency Networks (Section 3.3) and Markov Logic Networks (Section 3.4), two popular SRL models. Finally, I present experimental results of these two approaches on various standard SRL datasets on tasks ranging from link prediction to information extraction.

- **Chapter 4** presents the extension of our boosting approach to the task of learning in presence of missing data (missing labels of ground atoms). I derive a structural-EM algorithm where every E-step calculates the expected values for the missing data and the M-step updates the structure using functional-gradients. Since this approach is derived using the basic RFGB algorithm, we can handle missing data for learning any models that use RFGB. I show how this approach can be adapted to learn structure of RDNs, MLNs, and relational policies. I show experimentally that the EM approach can learn more accurate models than the traditional closed-world assumption of assuming everything that is unobserved to be false.

- **Chaper 5** describes a non-parametric approach developed for learning relational models using examples of only one class label. Information extraction tasks generally only have annotated positive examples and assume all the others to be negative. Standard propositional approaches for one-class classification rely on a distance measure to classify all the examples near the labeled examples as also belonging to the labeled class. I define a tree-based relational distance measure that is learned specifically for the task of one-class classification. I show how distances from multiple trees can be combined to get the final distance measure and how can new trees be added to the current distance measure to maximize the one-class classification performance. Our approach can also be viewed as a propositional one-class classification techniques such as kernel density estimation (Parzen, 1962) and SVDD (Tax and Duin, 1999) using the learned relational distance measure.

- **Chapter 6** presents a technique to transform knowledge represented as combining rules from directed models into MLNs. Although the conditional distributions in directed models can be represented using MLNs, directed relational models also use combining rules to merge multiple conditional distributions. I present an algorithm that can convert combining rules such as noisy-or and weighted-mean into MLN rules by using four special classes of clauses. I also show how to transform the parameters of the combining rules into weights of the MLN clauses in closed form.

- **Chapter 7** presents various other explorations performed using RFGB and otherwise. I present our work on Alzhiemer's disease prediction using RFGB via segmentation of MRI images. I present our approach for temporal relation extraction from text where we utilized expert-defined rules and features. I present our approach for a large-scale relation extraction task, namely Knowledge Base Acceleration (KBA), where we process terabytes of streaming data to detect changes in extracted relations using the Elementary system (Niu et al., 2012b).

- **Chapter 8** concludes my thesis with a review of the contributions made by my work and promising future research directions.

2 BACKGROUND

This chapter presents background on SRL models on which my work is based. I start with a brief technical background on first-order logic and graphical models. In Section 2.2, I present an overview of SRL models, followed by details on two popular SRL models. I then present the learning challenges in these models and the approaches taken to solve them in literature. In Section 2.4, I present functional-gradient boosting, an ensemble approach[1], which forms the basis of my learning approaches. Finally, I present details about the evaluation metrics and datasets I used.

2.1 Technical Background

I first define some notation that is used throughout this thesis. I use capital letters such as X, Y, Z to represent variables and small letters such as x, y, z to represent values taken by the variables. I use bold-faced letters to represents sets. Bold letters such as $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ represent sets of variables and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ represent sets of values. I use $\mathbf{z}_{-z}$ to denote $\mathbf{z} \setminus z$, i.e., every element from $\mathbf{z}$ except z. Similarly $\mathbf{x}_{-i}$ is used to represent $\mathbf{x} \setminus x_i$.

2.1.1 Representation: First-order logic

A simplistic view of first-order logic (FOL) is that it generalizes propositional logic by introducing variables as arguments to propositions (p to $p(\mathsf{X})$) which can be used to make logical statements about all objects in the domain (Russell and Norvig, 2003). To avoid confusion with random variables, I use sans-serif capital letters X, Y, and Z to represent logical variables. I use lower-case sans-serif letters such as x, y and z to represent

[1]Ensemble methods learn multiple models instead of one (Bishop, 2006).

values taken by logical variables i.e. objects in the domain. The common logical operators and quantifiers used in this thesis are:

- $\wedge$: AND operator. E.g. $p(X) \wedge q(X)$ implies both p & q are true for X.
- $\vee$: OR operator. E.g. $p(X) \vee q(X)$ implies either p or q is true for X.
- $\Rightarrow$: Condition implies the consequence. E.g., $p(X) \Rightarrow q(X)$ implies that if $p(X)$ is true, $q(X)$ has to be true.
- $\forall$: True for all values. E.g., $\forall X, p(X)$ implies $p(X)$ is true for all values of X.
- $\exists$: True for at least one value. E.g., $\exists X, p(X)$ implies $p(X)$ is true for at least one value of X.

All logical variables are implicitly universally quantified (i.e. $\forall$) unless explicitly existentially quantified. Table 2.1 presents definitions of standard first-order logic terms.

2.1.2 Uncertainty: Graphical models

Graphical models (Koller and Friedman, 2009) represent conditional dependence among random variables which can then be used to factor the joint distribution over these variables. The factored distribution also reduces the number of parameters needed to model the joint distribution. Figure 2.1 shows sample graphical models over three variables. Undirected models such as Markov networks (Kindermann and Snell, 1980) factor the joint distribution as the product over potentials defined over cliques in the graph (subject to a normalization term). The potentials are shown using the function ϕ and the normalization term using Z.

Directed models such as Bayesian networks (Pearl, 1988) represent the joint distributions as a product of conditional distributions for each variable given the parents of the variable (e.g., C is the parent of B in the

Constants	Represent objects in the domain. E.g., anna, bob.
Variables	A variable can be assigned a value from a range of constants. E.g., Variable X may take a value from $\{\mathsf{anna}, \mathsf{bob}\}$.
Predicate	Represents relations between objects in the domain. E.g., the $\mathsf{Friends}$ predicate captures the friendship relation.
Atom	A predicate along with its arguments. E.g., $\mathsf{Friends}(X, Y)$, $\mathsf{Father}(\mathsf{bob}, \mathsf{anna})$.
Literal	An atom or its negation E.g., $\mathsf{Friends}(\mathsf{anna}, \mathsf{bob})$, $\neg\mathsf{Father}(X, Y)$.
Grounding	Substituting a variable with a constant. E.g., A possible grounding of $\mathsf{Father}(X, Y)$ is $\mathsf{Father}(\mathsf{bob}, \mathsf{anna})$.
Ground atom literal	An atom/literal without any variables. E.g., $\mathsf{Friend}(\mathsf{bob}, \mathsf{anna})$.
Clause	A disjunction (i.e., OR) of literals E.g., $\mathsf{Friend}(X, Y) \vee \mathsf{Father}(X, Y)$ states that either X is a friend of Y *OR* X is the father of Y.
Horn Clause	A clause with only one positive literal commonly represented with an implication ($\mathsf{Body} \Rightarrow \mathsf{Head}$) having one literal in the head. E.g., $\mathsf{Friend}(X, Y) \wedge \mathsf{Smokes}(Y) \Rightarrow \mathsf{Smokes}(X)$, i.e., $\neg\mathsf{Friend}(X, Y) \vee \neg\mathsf{Smokes}(Y) \vee \mathsf{Smokes}(X)$.

Table 2.1: First-order logic terminology.

figure). To ensure the product of conditional distributions represents the joint distribution, directed models require the model to be acyclic. As can be seen here, the factored distribution needs five independent parameters (two for $P(B|C)$ and $P(C|A)$ each and one for $P(A)$) as compared to the seven independent parameters needed for the joint distribution ($2^3 - 1$).

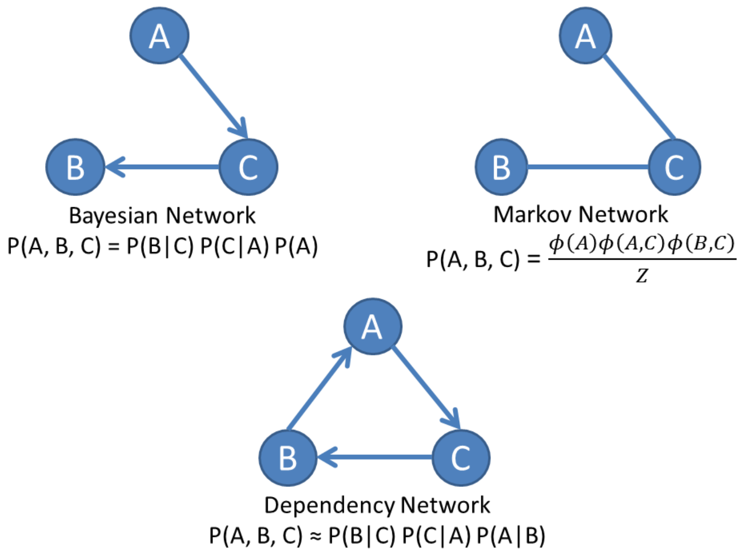

A
B
C
Bayesian Network
P(A, B, C) = P(B|C) P(C|A) P(A)
A
B
C
Markov Network
P(A, B, C) = $\frac{\phi(A)\phi(A,C)\phi(B,C)}{Z}$
A
B
C
Dependency Network
P(A, B, C) ≈ P(B|C) P(C|A) P(A|B)

Figure 2.1: Sample probabilistic graphical models over three variables. Note that Markov networks do not need to have a potential ϕ defined for every clique (assumed to be 1, if undefined). The dependency network has a cycle among variables A, B and C.

Dependency networks (Heckerman et al., 2001) are directed graphical models that remove this acyclicity condition (e.g., in the figure A, B and C have a cycle) thereby allowing for faster learning of these models. But the product of the conditional distributions may not produce a coherent joint distribution [2]. I discuss dependency networks in more detail in Section 2.2.1.

2.2 Statistical Relational Learning Models

As mentioned in the previous chapter, Statistical Relational Learning (SRL) models combine probabilistic models and first-order logic to model uncertain multi-relational data. Although SRL has recently surged in popularity

[2] $P(A, B) \neq P(A|B) \times P(B|A)$.

(with annual workshops since 2000), the idea of incorporating probabilities with logic was first considered in 1980s. Nilsson (1986) used probabilities on logic statements to compute probabilities of logical entailments and thereby constrain the space of consistent distributions. Knowledge Base Model Construction (KBMC; Wellman et al. 1992) builds a decision model based on the current problem instance (e.g., FOL query) and knowledge base (e.g., FOL rules). Although these models did not have any learning methods, they set the framework for the learnable models that followed.

One such approach was proposed by Haddawy (1994), where the logical statements are used to construct a Bayesian Network. For example, the definite clause $A, B \Rightarrow C$ can be used to introduce A and B as the parents of C in a Bayesian network. This led to the work by Ngo and Haddawy (1996) on Probabilistic Logic Programs (PLP), which used definite FOL clauses and a corresponding probability distribution to define the structure of a Bayesian Network. PLP allowed multiple FOL clauses for a predicate and used combining rules to merge the distributions from each clause. This work was later simplified as Bayesian Logic Programs (BLP; Kersting and De Raedt 2007) by using Bayesian clauses that separate the logical (conditions under which the rules apply) and probabilistic (the conditional distribution) components of the FOL clauses. Object-Oriented Bayesian Networks (OOBN; Koller and Pfeffer 1997b) and Probabilistic Relational Models (PRMs; Getoor et al. 2001) defined a Bayesian network over variables in classes and attributes in a relational schema respectively, which can then be grounded over all the objects in the domain to generate a ground Bayesian network.

Rather than introducing relations to graphical models, Poole's Probabilistic Horn Abduction (PHA; Poole 1993) introduces probabilistic atoms to deterministic Horn clauses to abduce the probabilities. Due to the independence assumptions made by PHA, the probabilities can be computed as a sum of products. Stochastic Logic Programs (SLP; Muggleton 1996), on the other hand, extend stochastic grammars to handle Horn clauses as a grammar rule and use SLD resolution to prove the queries. These models illustrate two views of SRL models: 1) introducing relations in propositional graphical models or 2) introducing weights or probabilities to first-order logic.

Adding relations to propositional models Models such as PRMs, BLPs and Relational Bayesian Networks (RBN; Jaeger 1997) can be viewed as introducing relations to Bayesian Networks. Similarly Markov Logic Networks (MLN; Domingos and Lowd 2009) and Relational Markov Networks (RMN; Taskar et al. 2002) can be viewed as adding first-order logic to Markov networks. Relational Dependency Networks (RDN; Neville and Jensen 2007) do the same for Dependency Networks. Section 2.2.1 presents more details about RDNs from this perspective.

Softening first-order logic Alternatively, models such as Problog (Raedt et al., 2007) and SLP can be viewed as introducing probabilities to FOL rules. MLNs and BLPs can also be viewed as assigning weights and probabilities to first-order logic rules respectively. While SLP and Problog use proof theory (think theorem-proving in first-order logic or derivation in grammars) for computing probabilities, BLP and MLN probabilities are computed based on propositional networks generated from grounding FOL statements. Section 2.2.2 presents MLNs from this perspective.

2.2.1 Relational Dependency Networks

Dependency networks (DNs; Heckerman et al. 2001) are graphical models that approximate a joint probability distribution as a product of conditional probability distributions (CPDs) ($P(\mathbf{X}) \approx \prod_i P(X_i \mid Pa(X_i))$). Unlike Bayesian Networks, DNs allow cycles in the graphical model, as a result the joint distribution is approximated. For example, DN can have an edge from A to B and vice versa. Applying the chain rule on the product of the conditional distributions $P(A \mid B)P(B \mid A)$ in such a graph does not necessarily match the joint distribution $P(A, B)$, hence the approximated joint distribution.

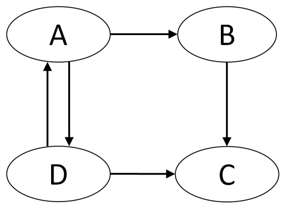


Figure 2.2: A dependency network.

Since DNs may contain cycles, each conditional distribution can be learned independently, which makes learning DNs much faster. Figure 2.2 shows a sample dependency network where $P(A, B, C, D)$ is approximated by the product $P(B|A)P(C|B, D)P(D|A)P(A|D)$. Heckerman et al. (2001) have shown that *ordered pseudo-Gibbs sampling* can be used to recover the full joint distribution from these conditional distributions as long as each conditional distribution is consistent, i.e., it can be obtained from the true joint distribution. For the proof and further details, please refer to Heckerman et al. (2001).

Relational Dependency Networks (RDNs) are a relational extension of DNs. RDNs are dependency networks where each node is a (first-order) predicate and the CPDs capture the conditional distribution of a predicate given a subset of all the other predicates. Similar to DNs, the network in RDNs can have cycles and hence approximate the joint distribution. Each predicate has an associated CPD conditioned on the value of its parents. Each CPD can be compactly represented using models such as Relational Probability Trees (RPT; Neville et al. 2003a) or Relational Bayesian Classifiers (RBC; Neville et al. 2003b).

An example RDN is presented in Figure 2.3 for an university domain. The ovals indicate predicates, while the dotted boxes represent the objects in the domain. As can be seen, there are *professor*, *student* and *course* objects with *taughtBy* and *takes* as the relations among them. The nodes *avgSGrade* and *avgCGrade* are aggregator functions over grades on students

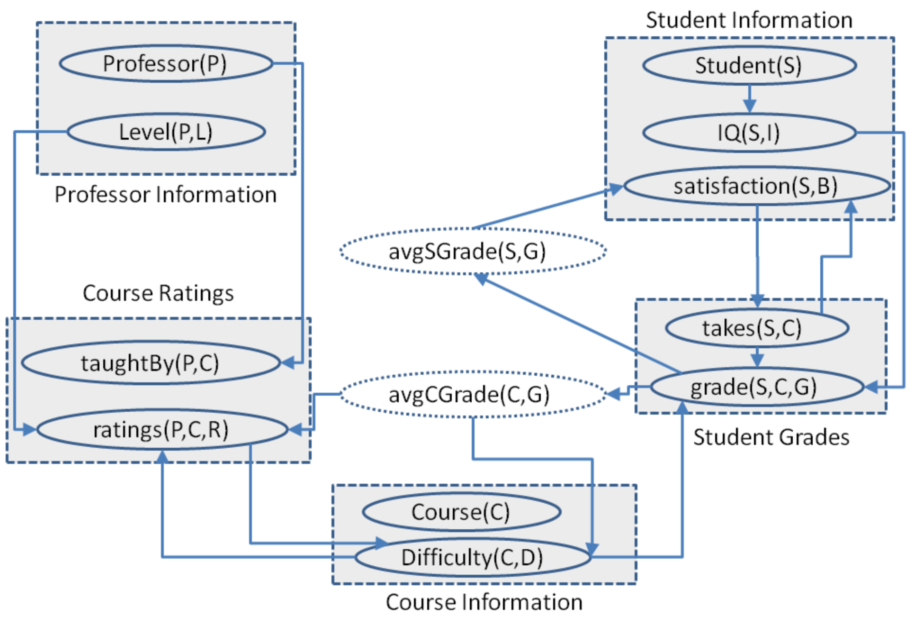


Figure 2.3: A relational dependency network.

and courses respectively. The arrows indicate the probabilistic (or possibly deterministic) dependencies among the predicates. For example, the predicate *grade* has *difficulty*, *takes*, and *IQ* as its parents. Also note that there is a bidirectional relationship between *satisfaction* and *takes*. Given the structure along with the conditional distributions, we can now use ordered pseudo-Gibbs sampling (Heckerman et al., 2001) to answer queries such as the satisfaction of a specific student.

2.2.2 Markov Logic Networks

Markov Logic Networks (MLNs) are relational models represented using weighted first-order logic rules. These rules provide a template for generating a Markov network by grounding the variables to all the constants[3]

[3]Most work assumes a finite set of constants, as do I in this document. For MLNs for an infinite domain, refer to Singla and Domingos (2007).

in the first-order logic rules. Each rule f_i forms a clique in the ground network and its weight w_i determine the potential for each clique. Figure 2.4 shows a MLN rule from a simple cancer domain (adopted from Domingos and Lowd 2009). The corresponding ground Markov network generated from these rules for a domain with two constants $X, Y \in \{a, b\}$ is shown in Figure 2.5.

$$\text{Weight=1.1} \quad \mathsf{Friends}(X, Y) \wedge \mathsf{Smokes}(Y) \rightarrow \mathsf{Smokes}(X)$$

Figure 2.4: Sample MLN rule.

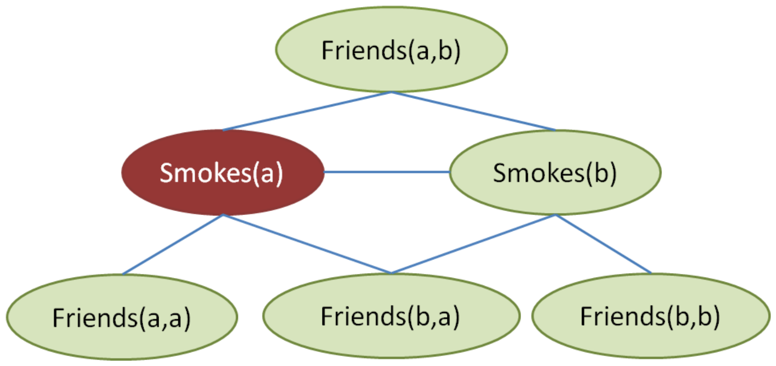


Figure 2.5: A ground Markov network for the rule in Figure 2.4 where $X, Y \in \{a, b\}$. Each node in the ground network is a ground atom. Red (dark) nodes indicate ground atoms that are false whereas green (light) nodes are true.

The joint probability distribution in a MLN is given by the product of the potentials on each clique, similar to Markov networks. Each clique potential is defined as the exponentiated weight of the grounded clause. For a given world state (truth value assignment to all ground atoms), the clique potential function returns e^{w_i} if the ground clause is true, otherwise it returns 1. Since all the cliques generated by grounding the same clause have the same weight, the probability of a given world state can be calculated using the number of true groundings of each clause. Hence the probability of the data is given by:

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

where $n_i(\mathbf{x})$ is the number of times the i^{th} formula is satisfied by the world $\mathbf{x}$ and Z is a normalization constant (as in Markov networks). In Figure 2.5, the sample MLN clause : $\neg Friends(X, Y) \vee \neg Smokes(Y) \vee Smokes(X)$ is only false for the grounding $\{X = a, Y = b\}$ and true for the remaining three groundings. So in this given world state, $n_i(\mathbf{x}) = 3$.

Similar to Markov networks, the normalization term is expensive to compute since its size is exponential in the number of features. In relational models, this problem becomes worse since Z is exponential in the number of ground atoms, which unlike the features grows with the dataset size. For example in the sample MLN, the $Friends$ predicate would have $O(n^2)$ groundings, where n is the number of people in the dataset. As a result, most learning methods approximate the likelihood (given above) with the pseudo-likelihood (PL):

$$PL(\mathbf{X} = \mathbf{x}) = \prod_{X_i \in \mathbf{X}} P(X_i = x_i | MB(X_i))$$

where $MB(X_i)$ corresponds to the Markov blanket[4] of the ground atom, X_i in the ground Markov network. The pseudo-likelihood term is similar to the probability distribution of RDNs, which is also defined as a product of the conditional distributions.

[4]The Markov blanket of a node x_i is all the direct neighbors of x_i in the ground Markov network.

2.3 Learning in SRL Models

SRL Models, and graphical models in general, are specified in terms of the *structure* of the model and the *parameters* defined over this structure. The structure defines the relations among the variables and the parameters defined the degree of the relationship. For Dependency networks, the structure of the model are the edges in the network and the parameters are the conditional distributions defined for each node based on this structure. In RDNs, the structure is also defined in terms of the edges but the nodes in the network are first-order logic predicates. The parameters in the model are relational conditional distributions (e.g. $P(difficulty(C, D)|avgCGrade(C, G), ratings(P, C, R))$. In MLNs, the structure are the first-order logic rules and the parameters are the weights of the rules. Next I present prior work on parameter learning in SRL followed by the same for structure learning.

2.3.1 Parameter learning

Since the parameters of a model are defined with respect to a model structure, the parameter-learning approaches assume that the structure is already provided. The parameter-learning problem also depends on the type of the underlying model, namely directed or undirected models.

Directed models

In case of directed models such as PRMs, BLPs and RDNs, it is assumed that the parents of every logical predicate is known. Similar to Bayesian networks, the problem of parameter learning in these models can be viewed as learning the conditional distributions for each predicate. In propositional models, the conditional distributions are commonly represented using a conditional probability tables (CPT) for every instantiation of the parent variables. The entries in the table can be calculated in closed-form by using the counts of the variable instantiations in the training data (Koller and Friedman, 2009). A similar approach is taken in relational models such as PRMs and BLPs where the counts are computed over all ground atoms to estimate a relational CPD (Getoor et al., 2001; Kersting and De Raedt, 2007). Since RDNs also learn conditional distributions, a similar approach can be taken. Natarajan et al. (2005; 2008) present a general approach for parameter learning in directed models and derive an EM approach to handle combination functions in these models.

Undirected models

For parameter learning in undirected models, the structure of the model (which is the cliques in the network) is assumed to be given and the parameter-learning problem corresponds to learning the clique potentials. In MLNs, since the first-order logic rules specify the cliques in the network, the parameter-learning problem corresponds to learning the weights of the rules. The earliest approaches for learning the weights in MLNs used gradient descent (Singla and Domingos, 2005) where the gradients for the weight of the clause f_i is given by:

$$\Delta_{w_i} = n_i(\mathbf{x}) - E_w[n_i]$$

where $n_i(\mathbf{x})$ is the number of times clause f_i is true in the data and E_w calculates the expected number of times it would be true given the current weights. The expectation term can be calculated as $E_w[n_i] = \sum_{\mathbf{x}'} P(\mathbf{x}'; w) \cdot n_i(\mathbf{x}')$. Since calculating $P(\mathbf{x}'; w)$ requires inference to be performed using the current weights, inference needs to be performed for every gradient step. As a result instead of computing the expectation, most approaches use the counts from the maximum a posteriori [5] (MAP) estimate ($E_w[n_i] \approx n_i(\mathbf{mx})$ where $\mathbf{mx} = \arg\max_{\mathbf{x}'} P(\mathbf{x}'; w)$). Following this

[5]Unlike parameter estimation, MAP estimate defines most likely assignment or most probable explanation in inference.

work, second-order gradient descent approach using diagonal Newton and scaled conjugate gradients have been developed for MLNs (Lowd and Domingos, 2007). There have also been margin-based parameter-learning approaches developed for MLNs (Huynh and Mooney, 2009, 2011). But all of these approaches perform iterative updates to the weights where each update computation needs to perform inference. As a result, even parameter learning in MLNs can be computationally intensive.

2.3.2 Structure learning

Unlike parameter learning, structure-learning approaches search over the space of possible structures for a model. Generally structure-learning approaches use a scoring function to evaluate a structure, a hypothesis space of valid structures to search over and a search strategy to search within the hypothesis space. Since the number of possible structures for relational models can be very large (structure learning is NP-Hard even in propositional models (Chickering, 1996)), most approaches use a greedy search strategy in the hypothesis space. Since the predictive performance of a structure (standard scoring function) depends on the parameters too, the search procedure needs to learn the parameters for every candidate structure, increasing the computational complexity of structure learning.

Due to the relational nature of data, potentially multiple objects or attributes might influence a ground atom. For example in a domain with three predicates, $A(X), B(X,Y)$ and $C(Y)$, suppose the structure-learning approach selects $B(X,Y)$ to be the parent of $A(X)$. So a grounding of $A(X)$, say $A(x)$ has the parents: $\{B(x,y) : y \in Y\}$. Since each grounding of $A(X)$ has $|Y|$ parents, using traditional conditional distribution tables requires $2^{|Y|}$ parameters. Hence relational models commonly use *aggregators* such as $\mathtt{count}$, max, $average$, $\mathtt{exists}$, etc. to combine multiple parents into a single parent to prevent the exponential number of parameters (Getoor and Taskar, 2007). For example, the $\mathtt{exists}$ aggregator function can be computed using $\exists Y, B(X,Y) \rightarrow \mathtt{Exists}(B(X))$ and then one can use $\mathtt{Exists}(B(X))$ as the parent of $A(X)$. Similarly if Y is a numeric or ordinal variable, one can calculate $MaxY(B(x,Y))$ as the maximum value of Y among all the groundings of $B(x,Y)$. Searching over the space of possible aggregators further increases the cost of learning the structure of SRL models. I first present prior work on structure learning for directed models followed by the same for undirected models.

Directed models

BLPs (Kersting and De Raedt, 2007) and PRMs (Getoor et al., 2001) use a greedy hill-climbing approach based on operators over the structure similar to the structure-learning approaches for Bayesian networks. BLPs define operators such as adding or removing literals from clauses, replacing variables by constants or vice versa, and adding or removing clauses. PRMs, on the other hand, define a set of potential parents for every target attribute and only considers adding or removing a parent from this set during the greedy search. To score a candidate structure, both the models first calculate the parameters based on counts in the data as mentioned before. The candidate structure with the highest score (calculated based on the likelihood of the training data) is accepted and the process continues.

A similar approach is taken by the "Score As You Use" (SAYU; Davis et al. 2007) algorithm, where clauses are proposed using the Aleph (Srinivasan, 2004) ILP engine. The clauses are used to create features for a naive Bayes and scored based on the improvement in area under precision-recall curve. The learned rules are then used to create new views, i.e., for predicate invention, which are used as features in a tree-augmented naive Bayes classifier (Friedman et al., 1997).

To learn the structure of RDNs, Neville and Jensen (2007) propose learning the conditional models directly for every predicate. To model the conditional distributions, they use two models: Relational Probability

Trees (RPT; Neville et al. 2003a) and Relational Bayesian Classifiers (RBC; Neville et al. 2003b). Since RDNs allow cycles, they can learn models for each predicate independently with no constraints on the parents. The conditional models for each predicate can be used to derive the structure of the RDN, i.e., predicates appearing in the model are the parents in the RDN. To begin with, they assume all the other predicates to be parents of a target predicate, i.e., a completely connected graph. When the conditional models are represented using RBCs, only the parameters of the classifiers are learned without any selection of parents (Neville et al., 2003b). When RPTs are used, the tree-learning approach automatically selects the parents from all the other predicates while learning the tree (Neville et al., 2003a).

Undirected models

For undirected models, since most of the prior work as well as my work focuses on MLNs, I present structure-learning approaches for MLNs. Structure learning in MLNs corresponds to learning the clauses along with the weights of these clauses. The initial approach to learn MLN structure avoided learning parameters for every structure by learning the structure, i.e., the clauses of the model first and then learning the parameters (Richardson and Domingos, 2004). They used CLAUDIEN (Van Laer et al., 1994), a first-order logic clause learner, to first learn the rules and then learned the weights of the rules. But this approach does not take the potential parameters into account before scoring the clauses and as a result can be sub-optimal. Following this work, Kok and Domingos (2005) developed a structure-learning approach that searched over the space of clauses and learned the weights for scoring each candidate structure. As shown by them experimentally, learning the structure along with the weights has a better accuracy than just learning the clauses and then learning the weights. As the search over the space of possible clauses can be slow, approaches were developed to discover templates from the ground atoms to construct fewer candidate clauses. I briefly describe three structure-learning approaches below.

Bottom-up structure learning (BUSL; Mihalkova and Mooney 2007). BUSL starts by creating template nodes (TNodes) using true ground atoms in the data. For each ground atom of a target predicate, they find atoms that share a constant and then variablize the atoms. To variablize the ground atoms, they replace the constants with a variable where the shared constants use the same variable . For example, for a query atom actor(brando), workedUnder(brando, coppola) shares the constant brando. The TNodes created from these atoms are actor(X) and workedUnder(X, Y). The TNodes are generated for all the ground atoms to create one set of features in a feature vector. For every ground atom, the TNode feature is set to true, if there exists some grounding that is true for that atom. For example, actor(brando) would set the features actor(X) and workedUnder(X, Y) to true in the example above. Given these feature vectors for every ground atom, they use the Grow-Shrink Markov Network (GSMN) learning algorithm to learn a propositional Markov network. The cliques in this Markov network are used to create candidate clauses by combining the TNodes in a clique. Only the candidate clauses are considered for addition to the MLN thereby reducing the learning time.

Learning via hypergraph lifting (LHL; Kok and Domingos 2009). LHL begins by creating a hypergraph from the data where the constants or objects in the domain form the nodes and the true relations form the hyperedges. Hyperedges can connect more than two nodes when relations have more than two arguments. A sample hypergraph is shown in Figure 2.6 on an university domain. Professors, p1 and p2; students s1 and s2; and courses c1, c2, and c3 form the nodes in the graph. Each relation is shown by a different edge type. For example, course c1 is taught by professor p1 and student s2 has taken the course c3. The constants are then clustered using a MLN which prefers clustering constants having the same relation with the another cluster. Relational path-finding on this lifted hypergraph is used to obtain candidate MLN clauses. Path-finding traverses the graph in a depth-first manner where every traversed hyperedge adds the relation to the candidate clause. Similar to BUSL, only the candidate clauses are considered for addition to the MLN in the greedy search.

Learning using Structural Motif (LSM; Kok and Domingos 2010).
Similar to LHL, LSM starts with a ground hypergraph constructed from the training data. Instead of clustering based on just the neighbors, LSM performs N random walks starting from each node. If the distribution of the paths in the random walks is similar for two nodes, they are clustered together. This process is repeated till no two clusters are similar to create the lifted hypergraph. Again path-finding over the lifted hypergraph is used to create the candidate clauses. But instead of using a greedy search as used by the earlier methods, all the clauses are added to the MLN and clauses with low weights are dropped.

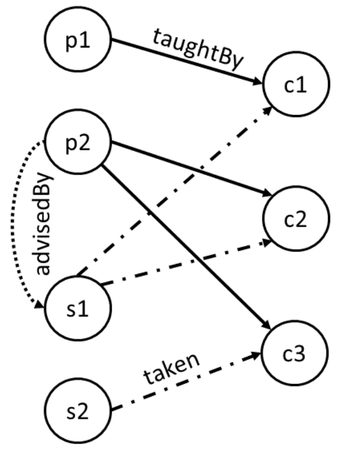


Figure 2.6: Example hypergraph for an university domain. The constants (professors, students and courses) form the nodes and the relations (advisedBy, taughtBy and taken) form the edges between the constants.

2.3.3 Learning trade-offs

Traditionally artificial intelligence relied on
the structure as well as the parameters of the models being specified by an expert. With the availability of training data, it became possible to learn

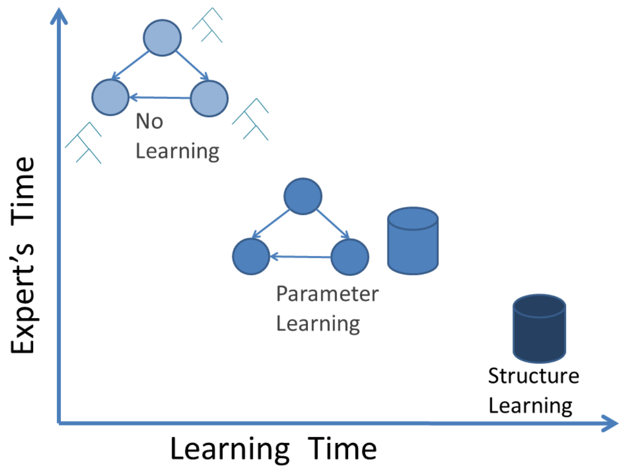


Figure 2.7: Trade-offs between expert's time and learning time for learning problems in SRL models.

the parameters of the model using training data while using the structure defined from the expert.

Parameter learning reduced the amount of effort needed from the expert and potentially improved the accuracy of the model (by relying on data to correct mistakes made by experts), but also increased the computational time. For some domains, the structure of the model may be non-trivial, not known or insufficient. As a result, both the structure and parameters of the model need to be learned from the data.

Although *structure learning* reduces the expert's effort, it can be computationally intensive due to the large space of possible structures while including parameter learning as a sub-task. Figure 2.7 shows this trade-off between the computation cost and expert's effort. My research has focused on reducing the learning time while improving the accuracy of structure learning in SRL models. To achieve this goal, I learn multiple weakly predictive relational models using functional-gradient boosting (FGB). I present details about FGB for propositional domains next.

2.4 Functional-Gradient Boosting

Most machine learning approaches use a parametric model that optimizes a specific loss function. For example, the logistic regression model uses a weight parameter w, and uses gradient descent to find the best parameters that maximize the likelihood of the data. Let $\{x_1, \dots, x_n\}$ be the set of examples and $\{y_1, \dots, y_n\}$ be their corresponding binary labels (represented as 1 and -1). In a logistic regression model, the probability of a label for a given example is given by $P(y_i|x_i; w) = 1/(1 + e^{-y_i w^T x_i})$. Assuming the examples are independent, the log-likelihood (LL) of the full dataset is given by

$$LL(\mathbf{y}, \mathbf{x}; w) = \sum_{x_i \in \mathbf{x}} \log P(y_i|x_i; w)$$

The standard method of learning in these models is based on gradient descent where the learning algorithm starts with initial parameters w_0 and computes the gradient of the log-likelihood (LL) function. The gradient during the m^{th} iteration is given by

$$\Delta_m = \frac{\partial LL(\mathbf{y}, \mathbf{x}; w_{m-1})}{\partial w_{m-1}}$$

and the weight parameter at the end of m iterations is given by

$$w_m = w_0 + \Delta_1 + \dots + \Delta_m$$

Friedman (2001) suggested that instead of using a parametric approach, apply the numeric optimization in the function space. For example, the probability of an example can be defined to be $P(y_i|x_i; \psi) = 1/(1 + e^{-y_i \psi(x_i)})$ and the gradients can be computed with respect to the function ψ. Similar to parametric gradient descent, we start with an initial function

ψ_0 and compute the gradients with respect to the function ψ:

$$\Delta_m = E_{x,y}\left[\frac{\partial LL(y, x; \psi_{m-1})}{\partial \psi_{m-1}}\right]$$

The ψ function at the end of m iterations is given by

$$\psi_m = \psi_0 + \Delta_1 + \ldots + \Delta_m$$

Since we only have a finite set of examples, rather than computing the gradients over the entire space of possible examples, Friedman suggests calculating the gradient for each training example. The gradient for an example x_i is given by $\Delta_m(x_i) = \partial LL(\mathbf{y}, \mathbf{x}; \psi_{m-1})/\partial \psi_{m-1}(x_i)$. We can then fit a regression function, $\hat{h}(x_i)$ to these gradients, $\Delta_m(x_i)$. Most functional-gradient approaches learn a regression tree to represent $\hat{h}$ and minimize the least-square error:

$$\hat{h}_m = \arg\min_h \sum_{x_i} [h(x_i) - \Delta_m(x_i)]^2$$

Since we approximated the gradients (Δ_m) using a regression function ($\hat{h}_m$), the potential function ψ after the m^{th} iteration is given by:

$$\psi_m = \psi_0 + \hat{h}_1 + \ldots + \hat{h}_m$$

Standard boosting approach (Freund and Schapire, 1996) learns a sequence of models where the weight on the examples (think importance of the examples) is updated after every iteration to increase the weight on incorrectly classified examples. As a result, every subsequent model attempts to correct the mistakes in the current model. FGB also learns a sequence of models ($\hat{h}_i$ in this case) where every subsequent model focuses on the incorrectly classified examples (due to the example's higher regression values), hence the *boosting* in its name.

#	a	b	**c**
1	1	0	1
2	0	0	0
3	1	1	0

Table 2.2: Training data with class label **c**.

#	a	b	Δ_1
1	1	0	0.50
2	0	0	-0.50
3	1	1	-0.50

Table 2.3: Initial regression dataset.

#	a	b	Δ_2
1	1	0	0.50
2	0	0	-0.37
3	1	1	-0.50

Table 2.4: Regression values after tree 1.

Consider the small dataset with three examples shown in Table 2.2 with two features (a and b) and the class label c. If we assume $P(y_i = 1|x_i; \psi) = 1/(1 + e^{-\psi(x_i)})$, the gradients can be shown to be $\Delta_m(x) = I(y_i = \hat{y}_i) - P(y_i = \hat{y}_i|x_i; \psi_m)$, where $I(y_i = \hat{y}_i)$ is the indicator function and $\hat{y}_i$ is the true label of x_i in the training data. Let us assume the initial prior ψ_0 returns 0 for every example i.e., $\psi_0(x) = 0, \ \forall x$. Given this initial prior, all the examples would have the predicted probability of $P(y_i = 1|x_i; \psi_0) = 0.5$, based on the current model.

The gradient Δ for the positive example is 0.5 whereas for the negative examples is -0.5, as shown in Table 2.3. Hence the gradients for the positive examples are pushing the ψ function for those examples to a higher value, thereby pushing the predicted probability closer to 1. Let us assume that we learn a regression tree with only one node that tests for a being true or not. The left (true) branch would contain two examples (#1 and #3) and the right (false) branch would contain only one example (# 2). Since the mean of the regression values of examples #1 and #3 is zero, the left leaf would return zero. On the other hand, the right branch would return -0.50 as the regression value. So the learned regression function $\hat{h}_1$ to fit the Δ values shown in Table 2.3 is:

$$\begin{aligned} \hat{h}_1(x) &= \quad 0.0 \qquad \text{if } a = 1 \\ &= -0.5 \qquad \text{if } a = 0 \end{aligned}$$

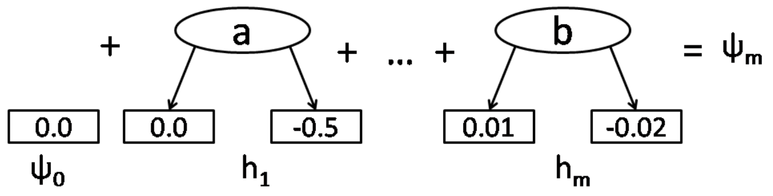


Figure 2.8: A sample model for predicting class label c after m iterations.

Since our ψ_0 function returned zero for all the examples, adding $\hat{h}_1$ to ψ_0 would give us $\psi_1 = \psi_0 + \hat{h}_1 = \hat{h}_1$ as our new current model. Given this model, we can compute the probabilities for the training examples. Since examples #1 and #3 have $\psi_1(x) = 0$, they still have the same gradients, but the gradient for example #2 has reduced to -0.37, as shown in Table 2.4. The next tree learned on this regression dataset would split on feature b and reduce the gradient on example #3, similar to what we observed in the previous step with example #2. Hence with each iteration, the gradients on the examples move closer to zero and our predicted probabilities would move closer to the observed values in the training data. Figure 2.8 shows a sample model ψ_m after m iterations of boosting.

2.5 Evaluation Approach

Before presenting our work on learning SRL models, I present details about the evaluation approach used. To show that our approach learns a more accurate model, we compare the accuracy of the probabilistic predictions made by the models using three evaluation measures commonly used in literature. We use the areas under Precision-Recall curve (AUC-PR) and Receiver Operating Characteristic curve (AUC-ROC; Davis and Goadrich 2006). We also use Conditional Log Likelihood (CLL) [6] which evaluates how close are the probability values to the true label.

[6] $CLL = \Sigma_i \log P(y_i = \hat{y}_i | \mathbf{x}_{-i})$

The goal in machine learning is to generalize from the training examples and not to overfit on the training or test examples. Evaluating on the training set or repeated evaluation on the same test set could lead to these overfitting issues. To evaluate the generalization performance of the classifiers, a common approach used in machine learning is n-fold cross-validation. In cross-validation, the dataset is first partitioned into n bins, $\{S_1, \ldots, S_n\}$. The classifier is trained on a train fold contain $n-1$ bins (S_{-i}) and tested on the remaining bin (S_i). This process is repeated n times for different train and test folds ($i = 1, \ldots, n$) and the results are averaged across folds.

In propositional datasets, since the examples are independent of each other, they can be partitioned into n bins easily (by placing $1/n$ examples in every bin). When dealing with relational datasets, the objects or examples are dependent on each other. Partitioning the data could result in breaking relationships between the examples in different bins. To avoid this, each network of connected objects, also known as a "mega-example", is kept in one bin. For example, a university is a mega-example with inter-connected student examples. The number of mega-examples in the dataset are generally used as the number of folds for cross-validation. As a result, each dataset may use a different number of cross-validation folds for evaluation. Next, I briefly describe commonly used SRL datasets in literature that I used in my work.

UW-CSE

The UW-CSE dataset (Richardson and Domingos, 2006) was creating from University of Washington's Computer Science and Engineering Department's student database (hence the name). The data set consists of details about professors, students and courses from five different sub-areas of computer science (AI, programming languages, theory, system and graphics). The goal is to predict whether a student is advised by a professor using the other predicates. For further details, refer to Section 3.5.2.

Cora

The Cora dataset, now a standard dataset for citation matching, was first created by McCallum et al. (2000), and later segmented by Bilenko and Mooney (2003). The dataset was later converted into relational format by Poon and Domingos (2007). In citation matching, the task is to identify citations that refer to the same paper, which as a sub-task may include matching the author, title and venue of citations.

For each citation we have information about the various fields using predicates such as author, title, venue, $\mathsf{hasWordAuthor}$, $\mathsf{hasWordTitle}$, and $\mathsf{hasWordVenue}$. This task has multiple target predicates ($\mathsf{sameAuthor}$, $\mathsf{sameVenue}$, $\mathsf{sameTitle}$, and $\mathsf{sameBib}$) for identifying matching authors, venues, titles and the complete citation.

IMDB

This dataset was created by Mihalkova and Mooney (2007) from IMDB.com and contains information about actors, movies, directors and the relationships between them. The predicates in this dataset are: actor, $\mathsf{director}$, $\mathsf{workedUnder}$, genre, and gender. The actor and $\mathsf{director}$ predicates are mutually exclusive predicates (i.e., $\mathsf{actor}(X) \Leftrightarrow \neg\mathsf{director}(X)$) that provide type information for the people in the domain.

WebKB

The WebKB dataset was first created by Craven et al. (1998) and contains information about department webpages and the links between them. It also contains the categories for each webpage and the words within each page. This dataset was converted by Mihalkova and Mooney (2007) to contain only the category of each webpage and links between these pages. The textual information was ignored.

Citeseer

Similar to the Cora dataset, the *Citeseer* dataset was created by Poon and Domingos (2007) for performing *information extraction* using the dataset from Lawrence et al. (1999). This dataset has 1563 citations and 906 clusters. It consists of four sections, each on a different topic. The dataset contains predicates such as the tokens at each position in a citation (`token`) and attributes about each token (`isDate, isDigit`, etc.). Given these facts, the goal is to predict the field type (`Title, Author, Venue`) for each position in the citation.

MovieLens

Xu et al. (2009) created the MovieLens dataset by randomly selecting a subset of 100 users and 603 movies from the ratings submitted by users on movielens.org. The task is to predict the preferences of the users on movies. The users have attributes `age, gender`, and `occupation`, while the movies have released `year` and `genre` attributes. The ratings of the movie by the user were in a 5-star scale. Typically, every user rated $(30-400)$ movies for a total number of 78,445 user-movie ratings.

Adverse Drug Reactions

The Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated datasets[7] that are modeled after real observational data sources, but contain hypothetical people with fictional drug exposure and health condition occurrence. We

[7] http://omopcup.orwik.com

use the OMOP simulator to generate a dataset of $10,000$ patients that includes record of drugs and diagnoses (conditions) with dates. The goal is to predict drug use based on the conditions.

NFL

We create a text-based dataset from the National Football League (NFL) corpus from the Linguistic Data Consortium[8] (LDC). This corpus consists of articles of NFL games over the past two decades. The idea is to read the articles and identify concepts such as *score*, *team* and *game* in the text. For example, in the text, "Green Bay defeated Dallas $28-14$ in Saturday's Superbowl game," the goal is to identify *Green Bay Packers* and *Dallas Cowboys* as the teams, and 28 and 14 as their respective scores.

We use the Stanford NLP toolkit[9] to create features from the parse trees obtained from the parser, the tags obtained from the part-of-speech tagger and the named entity-recognizer, etc. The features were constructed at different levels: *word-level*, *sentence-level*, *paragraph-level* and *article-level*.

Cancer

The cancer domain is a popular synthetic task used to evaluate MLNs (Kersting et al., 2009; Domingos and Lowd, 2009). The dataset used in my work has a friend network with 500 people, where each person has three attributes: `stress(X)`, `cancer(X)`, and `smokes(X)`. I created a friend network using a symmetric predicate, `friends(X,Y)`. The `stress` attribute for each person is set using a Bernoulli distribution. A person is more likely to smoke if he is stressed or has friends who smoke. Similarly, a person is likely to have cancer if he smokes or he has a lot of friends who smoke.

[8]http://www.ldc.upenn.edu

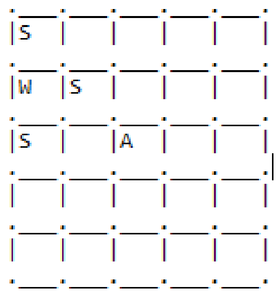
[9]http://nlp.stanford.edu/software/index.shtml

Heart

The Heart dataset [10] is a multivariate data set with 13 attributes. The task is to predict the presence of heart disease in patients given features such as age, gender, cholesterol level, and rest electrocardiograph.

Wumpus world

In imitation learning task (Ratliff et al., 2006), the goal is to learn a policy that can predict the next optimal action given the current state by *imitating* an expert's actions. To evaluate imitation learning on a relational domain, I created a simple version of the Wumpus task (Russell and Norvig, 2003). I use a 5x5 grid with a wumpus placed at a random location. The wumpus is always surrounded by stench on all four sides. I did not have any pits or breezes in our task.



```
.___.___.___.___.___.
|S  |   |   |   |   |
.___.___.___.___.___.
|W  |S  |   |   |   |
.___.___.___.___.___.
|S  |   |A  |   |   |
.___.___.___.___.___.|
|   |   |   |   |   |
.___.___.___.___.___.
|   |   |   |   |   |
.___.___.___.___.___.
```

Figure 2.9: The text UI for the simple Wumpus World. *W* indicates the wumpus location, *S* indicates the stench location, and *A* is the agent.

Figure 2.9 shows one instantiation of the initial grid locations. The agent can perform eight possible actions: four move actions in each direction and four shoot actions in each direction. The agent's task is to move to a cell such that he can fire an arrow to kill the wumpus. The dataset contains predicates for each cell such as `cellAt, cellRight`, and `cellAbove` and obstacle locations such as `wumpus` and `stench`.

[10]http://archive.ics.uci.edu/ml/datasets/Heart+Disease

3 LEARNING STRUCTURE FOR RELATIONAL MODELS

As described in Section 2.3.2, the problem of structure learning involves learning the dependencies among the predicates defining the task's data, along with the parameters associated with these dependencies. Due to the large space of possible structures, the task of learning the structure for relational models is a challenging task which has received attention (Mihalkova and Mooney, 2007; Biba et al., 2008a; Kok and Domingos, 2009, 2010) lately. As mentioned earlier, most of these structure-learning approaches first learn the structure for the model and then learn the parameters for the candidate structures. Based on the success of boosting-based approaches (Freund and Schapire, 1996; Friedman, 2001; Viola and Jones, 2001) where finding many rough rules of thumb is a lot easier and accurate than finding a single, highly accurate model; I propose to use functional-gradient boosting to learn the structure and parameter for relational models simultaneously. Our work presented in this chapter has been published in *Machine Learning* journal (Natarajan et al., 2012) and *IEEE International Conference on Data Mining* (Khot et al., 2011).

3.1 Introduction

Triggered by the intuition that learning many weakly predictive models can be faster and more accurate than finding a single, highly accurate local model, I propose to turn the problem of learning relational models into a series of relational function approximation problems using functional gradient-based boosting. I represent each conditional probability distribution as a sum of regression models grown incrementally. Instead of representing the conditional distribution as a single relational probability tree, I propose to use a set of relational regression trees (Blockeel and Raedt, 1998).

Using functional-gradient boosting for learning the structure has several advantages. First, being a non-parametric approach the number of parameters grows with every boosting iteration. Due to the incremental updates of the structure and greedy tree-learning, predicates are introduced only as needed; as a result the potentially large search space is not explicitly considered. Second, such an approach can take advantage of off-the-shelf regression-tree learners. Moreover, advances made in the tree learners, such as being able to handle continuous features, can be utilized easily. Third, the use of functional-gradient boosting makes it possible to learn the structure and parameters simultaneously, which is an attractive feature as structure learning in SRL models is computationally quite expensive. Finally, given the success of ensemble methods in machine learning (Bell et al., 2007; Viola and Jones, 2001) , it can be expected (and also shown empirically later in this chapter) that our approach is superior in predictive performance compared to other6 learning methods.

The chapter is organized as follows. In Section 3.2, I present the basic approach of Relational Functional Gradient Boosting (RFGB), followed by the adaptations of RFGB to RDNs and MLNs. In Section 3.5, I present the experimental results for both these adaptations on various tasks and compare them to state-of-the-art structure learning approaches.

3.2 Relational Functional Gradient Boosting (RFGB)

Similar to previous approaches in functional-gradient boosting, we use the sigmoid function ($\frac{e^x}{e^x+1}$) to represent the probability distribution of each example, x_i, i.e.,

$$\begin{aligned} P(x_i = \mathtt{true}|\mathbf{Ne}(x_i)) &= \frac{e^{\psi(x_i;\mathbf{Ne}(x_i))}}{e^{\psi(x_i;\mathbf{Ne}(x_i))}+1} \\ \log P(x_i = \mathtt{true}|\mathbf{Ne}(x_i)) &= \psi(x_i;\mathbf{Ne}(x_i)) - \log\left(e^{\psi(x_i;\mathbf{Ne}(x_i))}+1\right) \end{aligned} \tag{3.1}$$

where $\mathbf{Ne}(x_i)$ corresponds to the neighbors of x_i that influence x_i. In directed graphs, $\mathbf{Ne}(x_i)$ is the parents of the variable, whereas for a Markov network it is the Markov blanket.

For our approach, we define the joint probability distribution as a product of conditional distributions. This definition of the joint distribution can be applied to multiple relational models. As shown in Section 2.2.1, RDNs approximate the joint model as a product of conditional models. Although by definition, MLNs define the joint distribution as a product of potentials normalized over all the world states, optimizing this joint distribution is computationally expensive due to the exponential number of the world states. Hence learning approaches maximize the pseudo-likelihood (Domingos and Lowd, 2009; Kok and Domingos, 2009, 2010) which is also a product of conditional distributions.

The pseudo-log-likelihood (PLL) for $\mathbf{x}$ is defined as:

$$PLL(\mathbf{x}) \equiv \log P(\mathbf{X} = \mathbf{x}) = \log \prod_{x_i \in \mathbf{x}} P(x_i|\mathbf{Ne}(x_i)) = \sum_{x_i \in \mathbf{x}} \log P(x_i|\mathbf{Ne}(x_i))$$

As described in Section 2.4, functional-gradient boosting first computes the functional gradients ($\frac{\partial}{\partial\psi(x)}$) of the score that we wish to maximize. In our case, we wish to learn a model that maximizes the PLL of the examples in the training data. Hence, we calculate the functional gradient of PLL for every example.

$$
\begin{aligned}
\frac{\partial \log P(\mathbf{X}=\mathbf{x})}{\partial \psi(x_i;\mathbf{Ne}(x_i))} &= \frac{\partial \log P(x_i|\mathbf{Ne}(x_i))}{\partial \psi(x_i;\mathbf{Ne}(x_i))} \\
&= \frac{\partial \left(\psi(x_i;\mathbf{Ne}(x_i)) - \log\left(e^{\psi(x_i;\mathbf{Ne}(x_i))}+1\right)\right)}{\partial \psi(x_i;\mathbf{Ne}(x_i))} \\
&= I(x_i = true) - \frac{1}{e^{\psi(x_i;\mathbf{Ne}(x_i))}+1} \frac{\partial \left(e^{\psi(x_i;\mathbf{Ne}(x_i))}+1\right)}{\partial \psi(x_i;\mathbf{Ne}(x_i))} \\
&= I(x_i = true) - \frac{e^{\psi(x_i;\mathbf{Ne}(x_i)))}}{e^{\psi(x_i;\mathbf{Ne}(x_i))}+1} \\
&= I(x_i = true) - P(x_i = true;\mathbf{Ne}(x_i)) = \Delta(x_i) \qquad (3.2)
\end{aligned}
$$

where I is the indicator function, which returns 1 if x_i is a positive example here, otherwise returns 0. The gradient at each example ($\Delta(x_i)$) is now simply the adjustment required for the probabilities to match the observed value for that example. If x_i is a negative example, the gradient for $x_i \leqslant 0$, thereby pushing the ψ value closer to $-\infty$ and the predicted probability of the example closer to 0. On the other hand if x_i is a positive example, the gradients push the probabilities closer to 1.

Functional-gradient boosting can also be used to learn from probabilistic examples, i.e., examples with associated probabilities of being true instead of Boolean values. To handle probabilistic examples, instead of maximizing the log-likelihood, we minimize the KL-divergence between the true example probabilities ($\hat{P}$) and the current model's predictions (P), given by

$$
\begin{aligned}
D_{KL}(\hat{P}||P) = \sum_{x_i \in \mathbf{x}} \Bigg[& \hat{P}(x_i) \log \frac{\hat{P}(x_i)}{P(x_i|\mathbf{Ne}(x_i))} \\
& + (1-\hat{P}(x_i)) \log \frac{1-\hat{P}(x_i)}{1-P(x_i|\mathbf{Ne}(x_i))} \Bigg]
\end{aligned}
$$

For simplicity, we use $P(x_i)$ to represent $P(x_i = true)$ here. Minimizing the KL-divergence is equivalent to maximizing the negative KL-divergence. Hence, we calculate the functional gradients, as before, on the negative KL-divergence.

$$\begin{aligned}
\frac{\partial - D_{KL}(\hat{P}||P)}{\partial\psi(x_i;\mathbf{Ne}(x_i))} &= -\frac{\partial}{\partial\psi(x_i;\mathbf{Ne}(x_i))}\Bigg[\hat{P}(x_i)\log\frac{\hat{P}(x_i)}{P(x_i|\mathbf{Ne}(x_i))} \\
&\qquad + (1-\hat{P}(x_i))\log\frac{1-\hat{P}(x_i)}{1-P(x_i|\mathbf{Ne}(x_i))}\Bigg] \\
&= \hat{P}(x_i)\frac{\partial\log P(x_i|\mathbf{Ne}(x_i))}{\partial\psi(x_i;\mathbf{Ne}(x_i))} + (1-\hat{P}(x_i))\frac{\partial\log(1-P(x_i|\mathbf{Ne}(x_i)))}{\partial\psi(x_i;\mathbf{Ne}(x_i))}
\end{aligned}$$

The first gradient term ($\frac{\partial\log P(x_i|\mathbf{Ne}(x_i))}{\partial\psi(x_i;\mathbf{Ne}(x_i))}$) is same as the gradients calculated for positive examples, i.e., $1-P(x_i|\mathbf{Ne}(x_i))$. The second gradient term is the gradient for negative example, i.e., $0-P(x_i|\mathbf{Ne}(x_i))$. The gradient is now simplified to:

$$\begin{aligned}
&= \hat{P}(x_i)(1-P(x_i|\mathbf{Ne}(x_i))) + (1-\hat{P}(x_i))(0-P(x_i|\mathbf{Ne}(x_i))) \\
&= \hat{P}(x_i) - \hat{P}(x_i)P(x_i|\mathbf{Ne}(x_i))) - P(x_i|\mathbf{Ne}(x_i)) + \hat{P}(x_i))P(x_i|\mathbf{Ne}(x_i)) \\
&= \hat{P}(x_i) - P(x_i|\mathbf{Ne}(x_i))
\end{aligned}$$

The gradient for probabilistic examples is the difference between the true probability and the predicted probability. Since the log-likelihood term can be re-written as the negative KL-divergence[1], the gradients for negative KL-divergence match the gradients derived earlier for log-likelihood ($\hat{P}(x_i) = 1$ for positive examples and $\hat{P}(x_i) = 0$ for negative examples).

Figure 3.1 shows our relational functional-gradient boosting (RFGB) approach. Algorithm 3.1 presents the pseudo-code for the same. For each predicate k, we first generate the examples for a regression-tree learner,

[1]For a positive example, the second term in D_{KL} is zero and for a negative example, the first term in D_{KL} is zero since $p\log p \to 0$ as $p \to 0$.

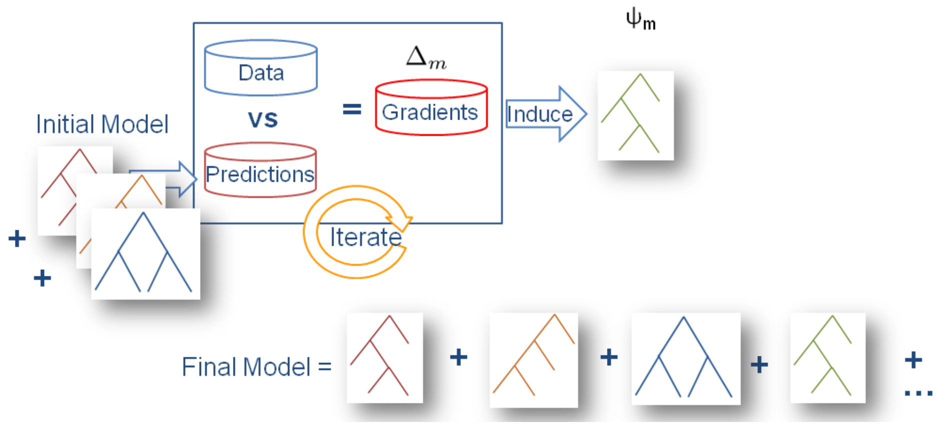


Figure 3.1: Relational Functional Gradient Boosting. This is similar to the standard FGB where trees are induced in stage-wise manner; the key difference being that the trees are relational regression trees. To compute the predictions, a query x, is applied to each tree in turn, and the numerical values at the leaf reached in each tree are summed to obtain $\psi(x)$.

then we call function *FitRelRegressTree* to get the new regression tree and add it to the current model (F^k_{m-1}). This is repeated up to a pre-set number of iterations M (in our experiments, typically, $M = 20$). Alternatively, we can use a stopping criteria to reduce over-fitting such as a) the average change in the gradient value between iterations being less than δ or b) for atleast $1 - \epsilon$ fraction of examples the change in gradient being less than δ. Early stopping (Mitchell, 1997) is another common approach to reduce over-fitting, where we evaluate the accuracy of the model on a held-aside validation set and stop adding trees when the accuracy decreases.

Note that after m steps, the current model F^k_m will have m regression trees, each of which approximates the corresponding gradient for the predicate k. These regression trees serve as the individual components ($\Delta_m(k)$) of the final potential function ψ. A key point about our regression trees is that they are not large trees, which reduces over-fitting while lowering the learning time. Typically, in our experiments, we limit the

Algorithm 3.1 RFGB(Data):
Relational Functional Gradient Boosting algorithm.

```
1: for $1 \leqslant k \leqslant K$ do    ▷ Iterate through K predicates
2:   for $1 \leqslant m \leqslant M$ do    ▷ Iterate through M gradient steps
3:     $S_k := \mathtt{GenExamples}(k; \mathtt{Data}; F^k_{m-1})$    ▷ Generate examples
4:     ▷ Fit trees to functional gradient
5:     $\Delta_m(k) := \mathtt{FitRelRegressTree}(S_k, k)$
6:     $F^k_m := F^k_{m-1} + \Delta_m(k)$    ▷ Update model
7:   end for
8:   $P(X_k = x_k | \mathbf{Ne}(x_k)) \propto F^k_M(x_k)$
9: end for
10: return
```

Algorithm 3.2 GenExamples(k, Data, F):
Generate regression examples from Data.

```
1: $S := \emptyset$
2: for $1 \leqslant i \leqslant N_k$ do    ▷ Iterate over all examples
3:   Compute $P(x^i_k = \mathtt{true} | \mathbf{Pa}(x^i_k))$    ▷ Probability of an example being true
4:   $\Delta(x^i_k) := I(x^i_k = \mathtt{true}) - P(x^i_k = \mathtt{true} | \mathbf{Pa}(x^i_k))$    ▷ Compute Gradient
5:   $S := S \cup [(x^i_k), \Delta(x^i_k))]$    ▷ Update relational regression examples
6: end for
7: return $S$    ▷ Return regression examples
```

depth of the trees to be 3 and the number of leaves in each tree is restricted to be about 8 (the parameter L in *FitRelRegressTree*). The initial potential F^1_0 is set to capture the uniform distribution in all our experiments. However, it is possible to use more informative initial potentials that can encode domain knowledge or a prior probability about the target.

The function *GenExamples*, shown in Algorithm 3.5, is the function that generates the examples for the regression-tree learner. As can be seen, it takes as input the current predicate index k, the data, and the current model F. The function iterates over all the examples and for each example, computes its probability and gradient. Recall that for computing the probability of y_i, we consider all the trees learned for Y_i. For each tree, we compute the regression value based on the path taken by the example in the tree. The sum of regression values from all the trees is used to compute the probability of the example. The gradient (computed in line 4 in Algorithm 3.5) is the target regression value for the tree-learning step.

3.3 Adapting RFGB for RDNs

When applying functional gradient boosting to RDNs, the probability of each example is computed conditioned on its parents. Hence, the gradient of each example is given by Equation 3.2 with the $Ne(x_i)$ replaced with the parents of x_i, $Pa(x_i)$.

$$\begin{aligned} P(x_i|\mathbf{Pa}(x_i)) &= \frac{e^{\psi(x_i;\mathbf{Pa}(x_i))}}{e^{\psi(x_i;\mathbf{Pa}(x_i))}+1} \\ \Delta(x_i) &= I(x_i = true; \mathbf{Pa}(x_i)) - P(x_i = true \mid \mathbf{Pa}(x_i)) \end{aligned}$$

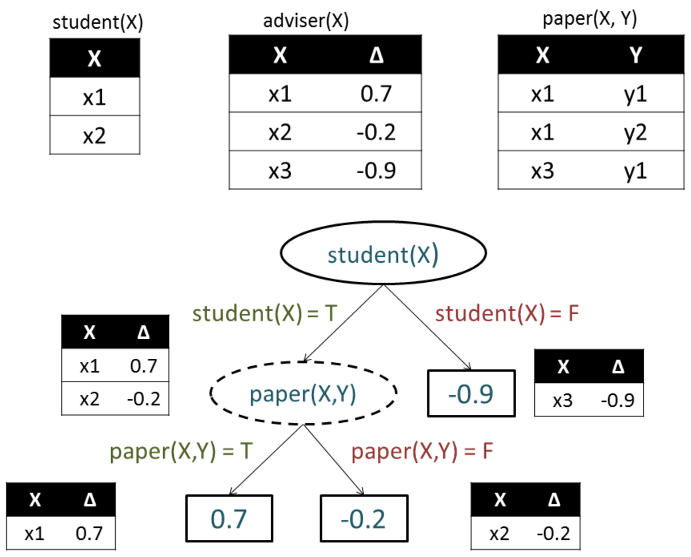
Representation of Functional Gradients for RDNs As mentioned above, we represent the function ψ using relational regression trees. Similar to decision trees, relational trees have tests on each node, but the node tests can contain a conjunction of literals and share any variable defined by their ancestor, except under one condition. The variables introduced by a node can not be used in the subtree under the false branch of that node. This follows from the way the test conditions are interpreted in relational trees. In a relational tree for $q(X)$, if a node contains the literal $p(X, Y)$, then the left or true branch corresponds to the examples $\{X : p(X, Y) = true\}$ and the right or false branch corresponds to the examples $\{X : \forall\ Y p(X, Y) = false\}$. This definition ensures that any example can take exactly one branch and thereby partitions the space. Since the false branch does not enforce a specific set of values that Y should take to follow that branch, Y can not be used in that subtree. Each example can take exactly one path in the relational trees and return the regression value at the leaf as the ψ function value for that example. (It is possible to also convert the relational trees into a decision list and return the regression value corresponding to the first satisfied clause in the list.)

At a fairly high level, the learning of relational regression tree proceeds as follows: The learning algorithm starts with an empty tree and repeatedly searches for the best test at a node according to a split criterion. Next, the examples in the node are split into *success* and *failure* according to the test. For each split, the procedure is recursively applied, further obtaining subtrees for the splits. We use weighted variance[2] as the split criterion and the average gradient of examples reaching a leaf as its regression value. These two conditions ensure that we greedily minimize the squared error at each node. We augment the relational regression-tree learner with the aggregation functions such as *count, max, average* in the inner nodes, thus making it possible to learn complex features for a given target.

An example relational regression tree is shown in Figure 3.2. The predicates $student$ and $paper$ are available as evidence (shown at the top) and the regression examples are created for the target predicate, $adviser$. The current node being scored is shown with the dotted oval, i.e., $paper(X, Y)$. Since $target(x1)$ and $target(x2)$ are the only examples reaching the node, introducing $paper(X, Y)$ only affects their regression values and only these two examples are used to score the node. Based on evidence, $paper(x1, Y)$ is true for $Y = y1$, and so $adviser(x1)$ takes the true branch. On the other hand, $paper(x2, Y)$ is false for all values of Y, and it takes the false branch. The values on the leaves are set to the mean of the regression values reaching the node and the variance computed based on the leaf values. The variance is computed for every possible test in this node and the test with the lowest variance is selected.

Since we can learn the models for each predicate independently in RDNs, we learn all our boosted trees for each predicate simultaneously.

[2] $\sigma_w = \sum_i (\Delta_i - \mu_1)^2 + \sum_j (\Delta_j - \mu_2)^2$ where Δ_i (and Δ_j) are the gradients of examples on the true (and false) branch and μ_1 (and μ_2) is the average gradient of these examples.



student(X)
X
x1
x2

adviser(X)	
X	Δ
x1	0.7
x2	-0.2
x3	-0.9

paper(X, Y)	
X	Y
x1	y1
x1	y2
x3	y1

X	Δ
x1	0.7
x2	-0.2

X	Δ
x3	-0.9

X	Δ
x1	0.7

X	Δ
x2	-0.2

Figure 3.2: Example of learning a relational tree from regression examples.

Hence, we swap lines 1 and 3 in Algorithm 3.1 and learn all the trees for one predicate before going on to the next predicate.

3.4 Adapting RFGB for MLNs

While RDNs do not use the number of groundings to compute the probability distributions, MLN probability distribution depends on the groundings of the clause. Hence, all the examples that reach the same leaf may not have the same distribution due to each example potentially having different number of groundings of the clause corresponding to the path taken.

Derivation of the Functional Gradient In undirected models, the probability of an example is independent of the other examples given its Markov blanket (MB). Hence, $Ne(x_i)$ in Equation 3.1 and 3.2 can be replaced with

$\mathrm{MB}(x_i)$ to obtain the gradient for each example, i.e.,

$$P(x_i|\mathbf{MB}(x_i)) = \frac{\exp(\psi(x_i;\mathbf{MB}(x_i)))}{\exp(\psi(x_i;\mathbf{MB}(x_i))) + 1} \tag{3.3}$$
$$\Delta(x_i) = I(x_i = true) - P(x_i = true \mid \mathbf{MB}(x_i))$$

One of the primary differences between applying functional gradient boosting to MLNs and other relational models lies in the definition of the ψ function derived below. In MLNs, the conditional probability of an example given its Markov blanket is represented as:

$$\begin{aligned} P(x_i = true|\mathbf{MB}(x_i)) &= \frac{\exp\left(\sum_j w_j n_j(x_i = true; \mathrm{MB}(x_i))\right)}{\sum_{x'} \exp\left(\sum_j w_j n_j(x_i = x'; \mathrm{MB}(x_i))\right)} \\ &= \frac{\exp\left(\sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))\right)}{\exp\left(\sum_j w_j nt_j(x_i; \mathbf{MB}(x_i))\right) + 1} \end{aligned} \tag{3.4}$$

where we defined

$$nt_j(x_i;\mathbf{MB}(x_i)) \equiv n_j(x_i = true;\mathbf{MB}(x_i)) - n_j(x_i = false;\mathbf{MB}(x_i)) \tag{3.5}$$

$n_j(\mathbf{x})$ is the number of true groundings of clause C_j in the ground Markov network of $\mathbf{x}$. Given our definitions of the probability of an example in Equations 3.3 and 3.4, we can derive the function ψ.

$$P(x_i|\mathbf{MB}(x_i)) = \frac{\exp(\psi(x_i;\mathbf{MB}(x_i)))}{\exp(\psi(x_i;\mathbf{MB}(x_i))) + 1} = \frac{\exp\left(\sum_j w_j nt_j(x_i;\mathbf{MB}(x_i))\right)}{\exp\left(\sum_j w_j nt_j(x_i;\mathbf{MB}(x_i))\right) + 1}$$
$$\Rightarrow \psi(x_i;\mathbf{MB}(x_i)) \equiv \sum_j w_j \; nt_j(x_i;\mathbf{MB}(x_i)) \tag{3.6}$$

$nt_j(x_i)$ is the difference between the number of true groundings of the clause C_j when an example is true and the groundings when it is false. If x_i appears in the head of a Horn clause C_j, $nt_j(x_i)$ also corresponds to the number of *non-trivial groundings* of x_i (Shavlik and Natarajan, 2009).

E.g, $\mathtt{target}(c1)$ only appears in the head of the clause $p(A, B) \wedge q(B, C) \rightarrow \mathtt{target}(A)$ and hence $nt_j(\mathtt{target}(c1))$ corresponds to the number of non-trivial groundings of $\mathtt{target}(c1)$.

I now explain non-trivial groundings in terms of the trivial groundings of an example. The trivial groundings are defined as the groundings of a clause that remain true irrespective of the truth value of the example. E.g., groundings of $\neg p(c1, B) \vee \neg q(B, C) \vee \mathtt{target}(c1)$ where $\neg p(c1, B) \vee \neg q(B, C) = \mathtt{true}$ satisfy the clause irrespective of the truth value of $\mathtt{target}(c1)$. So the non-trivial groundings of $\mathtt{target}(c1)$ that satisfy the clause correspond to the groundings where $p(c1, B) \wedge q(B, C) = \mathtt{true}$ (obtained by negating the condition for trivial groundings). In general, the non-trivial groundings of an example correspond to the groundings of a clause where the body of the clause is true. For a more detailed discussion of non-trivial groundings, see Shavlik and Natarajan (2009).

Representation of Functional Gradients for MLNs To apply functional gradient boosting, we need to find $\hat{\psi}$ such that the squared error between $\hat{\psi}$ and the functional gradient is minimized over all examples, i.e.,

$$\arg\min_{\hat{\psi}} \sum_{i=1}^{n} (\hat{\psi}(x_i; MB(x_i)) - \Delta(x_i))^2 \tag{3.7}$$

We consider two representations of $\hat{\psi}$s: *trees* and *clauses*.

Tree Representation

For our first representation, we use a relational regression-tree learner to fit the gradients on each example. Each path from the root to a leaf can be seen as a clause and the weight at the leaf corresponds to the weight of the clause. As an example, let us consider adding the literal $q(X, Y)$ to the tree at a node N. Let the current clause formed by the path from the root to the node N be $p(X) \rightarrow \mathtt{target}(X)$. So adding $q(X, Y)$ splits the current clause to two clauses,

$$\begin{aligned} C_1: \quad & p(X) \wedge q(X,Y) && \rightarrow target(X) \\ C_2: \quad & p(X) \wedge \forall\, Y \neg q(X,Y) && \rightarrow target(X) \end{aligned}$$

For all the examples that reach node N (i.e., p(X)=true), let $\mathcal{I}$ be the set of examples that satisfy $q(X,Y)$ and $\mathcal{J}$ be the ones that do not. Let w_1 and w_2 be the regression values that would be assigned to C_1 and C_2 respectively. Let $n_{x,1}$ and $n_{x,2}$ be the number of non-trivial groundings for an example x with clauses C_1 and C_2. The regression value returned for an example now depends on whether it belongs to $\mathcal{I}$ or $\mathcal{J}$. The regression function for the examples reaching node N can be defined as

$$\hat{\psi}(x_i) = n_{x_i,1} \cdot w_1 \cdot I(x_i \in \mathcal{I}) + n_{x_i,2} \cdot w_2 \cdot I(x_i \in \mathcal{J}) \tag{3.8}$$

and the squared error (SE) is

$$SE = \sum_{x \in \mathcal{I}} [n_{x,1} \cdot w_1 - \Delta_x]^2 + \sum_{x \in \mathcal{J}} [n_{x,2} \cdot w_2 - \Delta_x]^2$$

Taking the derivative w.r.t. the weights, we can calculate the optimum value for the weights at the leaf for a given split.

$$\frac{\partial}{\partial w_1} SE = \sum_{x \in \mathcal{I}} 2\,[n_{x,1} w_1 - \Delta_x]\, n_{x,1} + 0 = 0 \Rightarrow w_1 = \frac{\sum_{x \in \mathcal{I}} \Delta_x \cdot n_{x,1}}{\sum_{x \in \mathcal{I}} n_{x,1}^2} \tag{3.9}$$

Since the optimum weights have a closed-form solution, we can compute the weights easily for possible literals that can be added at each node. We greedily search for the literal that minimizes the squared error with the optimum weights.

Figure 3.3 gives the sample regression tree for $target(X)$ for the example considered above. Assume that the algorithm selected $p(X)$ to be the best split with weight w_3 on its false branch ($\mathrm{p}(\mathsf{X}) = \mathtt{false}$) and it is scoring $\mathrm{q}(\mathsf{X}, \mathsf{Y})$ as the next split for the true branch. As described above, $\mathcal{I}$ contains all examples that have at least one grounding for $\mathrm{q}(\mathsf{X}, \mathsf{Y})$ and $\mathcal{J}$ contains the rest; $\mathtt{target}(x_1)$ is in $\mathcal{I}$ if $\mathrm{p}(x_1) \wedge \exists \mathsf{Y} \mathrm{q}(x_1, \mathsf{Y})$ is true and $\mathtt{target}(x_2)$ is in $\mathcal{J}$, if $\mathrm{p}(x_2) \wedge (\forall \, \mathsf{Y}, \neg \mathrm{q}(x_2, \mathsf{Y}))$ is true. Given the number of groundings and gradients of examples in $\mathcal{I}$, we can now compute the weight w_1 on the left leaf using Equation 3.9 and similarly compute w_2 on the right leaf using $\mathcal{J}$. The MLN generated from this tree is:

$$\begin{aligned} w_1 &: \mathrm{p}(\mathsf{X}) \wedge \mathrm{q}(\mathsf{X}, \mathsf{Y}) \rightarrow \mathtt{target}(\mathsf{X}) \\ w_2 &: \mathrm{p}(\mathsf{X}) \wedge \forall \, \mathsf{Y} \neg \mathrm{q}(\mathsf{X}, \mathsf{Y}) \rightarrow \mathtt{target}(\mathsf{X}) \\ w_3 &: \neg \mathrm{p}(\mathsf{X}) \rightarrow \mathtt{target}(\mathsf{X}) \end{aligned} \tag{3.10}$$

Note that the examples reaching the leaf with weight w_1, namely $\mathcal{I}$, satisfy the body of the first clause. Also, by construction, these examples do not satisfy the body of the remaining two clauses. As mentioned before the non-trivial groundings of the clause correspond to the groundings which satisfy the body of the clause. Therefore, the set of examples $\mathcal{I}$ have zero non-trivial groundings for the last two clauses. Given our definition of ψ in Equation 3.6, we can see that the weights of these clauses do not effect the regression value returned for examples in $\mathcal{I}$ (since $nt_j(x; \mathbf{MB}(x)) = 0$). Hence, we use the examples from $\mathcal{I}$ to only compute the weight for the first clause (where $nt_j(x; \mathbf{MB}(x)) > 0$). Similar to decision trees, we partition the examples into mutually exclusive sets and only use the examples reaching the leaf

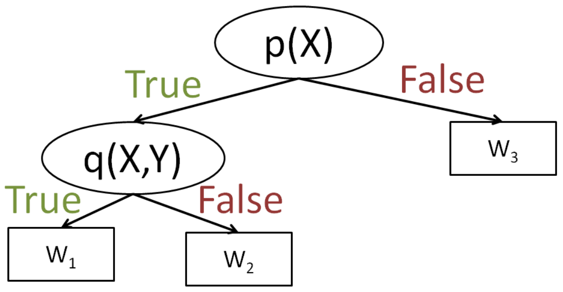


Figure 3.3: Sample tree for $\mathtt{target}(\mathsf{X})$.

to efficiently calculate the weight at that leaf.

We can efficiently compute the regression value returned for an example by using an ordered-list of clauses. In an ordered list, one would return the weight for the first satisfied clause in the given list. For the clauses shown above, we return the weight w_1 if the example satisfies the first clause. If it does not satisfy the first clause, we check for the second clause in the next step and so on. For the second clause in Equation 3.10, we do not need to check for $\forall\ Y \neg q(X, Y)$ any more. If there exists some Y such that $q(x_1, Y)$ returns true, we would have returned the weight w_1 for example $target(x_1)$ in the previous step. In general, we can remove the test conditions in the false branch in the ordered-list representation. We can rewrite the set of clauses in Equation 3.10 using the ordered-list approach:

$$\begin{aligned} w_1 &: p(X) \wedge q(X, Y) \rightarrow target(X) \\ w_2 &: p(X) \rightarrow target(X) \\ w_3 &: target(X) \end{aligned} \tag{3.11}$$

This ordered-list representation can only be used by our implementation and is only accurate if we have a single target predicate (explained below). Hence in all the scenarios where the ordered list can not be used, we revert to the unordered set of clauses shown in Equation 3.10. Unfortunately the false test conditions needed in the unordered set of clauses may result in large cliques in the ground Markov Network. For example, our second clause in 3.10 when converted to its conjunctive normal form (used to create the ground network) becomes $\neg p(X) \vee \exists Y q(X, Y) \vee target(X)$. MLNs assume that all existentially quantified variables have a finite domain. Thereby, a clause with existential variables in MLNs can be handled by replacing it with a disjunction over all the groundings of the existential variable. In this case, we will create a clause $\neg p(X) \vee q(X, y_1) \vee \ldots q(X, y_n) \vee$

$\mathtt{target}(X)$, which results in a large clique ($O(|Y|)$) in the ground network.

Clause Representation

Due to the potentially large clique generated by using regression trees, we define a second representation for the regression function, namely a set of Horn clauses. To learn this clausal representation, we ignore the false branch, i.e., set the weights on the false branch (w_2 and w_3 in the previous example) to zero. We learn Horn clauses by using a beam search that adds literals to clauses that reduce the squared error. We maintain a (beam-size limited) list of clauses ordered by their squared error and keep expanding clauses from the top of the list. We add literals to clauses as long as their lengths do not exceed a threshold and there are clauses in the stored list yet to be expanded. We recommend using the clausal representation when the existential variables introduced by the trees would make the inference step too slow.

In this version of our algorithm, we learn a set of clauses independently at each gradient step. Since we do not have two branches when every new condition is added, the error function becomes:

$$SE = \sum_{x \in \mathcal{I}} [n_{x,1} \cdot w - \Delta_x]^2 + \sum_{x \in \mathcal{J}} \Delta_x^2 \Longrightarrow w = \frac{\sum_{x \in \mathcal{I}} \Delta_x \cdot n_{x,1}}{\sum_{x \in \mathcal{I}} n_{x,1}^2}$$

Note that the key change is that we do not split the nodes and instead just search for new literals to add to the current set of clauses. Hence, instead of an ordered-list of clauses, we learn a single clause obtained by taking the true branch at every node. We repeat this process to obtain a pre-set number of clauses (set to 3 in our experiments) within each gradient step.

Algorithm 3.3 presents the pseudocode for learning relational regression trees for MLNs. The function $\mathtt{FitRelRegressionTree}(S, P)$ corresponds to the function used in Algorithm 3.1. We limit our trees to have maximum

Algorithm 3.3 FitRelRegressionTree(S, P):
Fit relational regression trees to the regression dataset, S.

```
1: Tree := createTree($P(X)$)
2: Beam := {root(Tree)}
3: L := 8   ▷ Maximum leaves
4: while $\mathtt{numLeaves}(\mathtt{Tree}) \leqslant L$ do
5:     Node := popBack(Beam)   ▷ Node with worst score
6:     C := createChildren(Node)   ▷ Create children
7:     BN := popFront(Sort(C))   ▷ Node with best score
8:     addNode(Tree, Node, BN)   ▷ Replace Node with BN
9:     insert(Beam, BN.left, BN.left.score)
10:    insert(Beam, BN.right, BN.right.score)
11: end while
12: return Tree
```

L leaves and greedily pick the worst node to expand (to reduce the error in that node). In FitRelRegressionTree, we begin with an empty tree that returns a constant value. We use the background predicate definitions to create the potential literals that can be added (createChildren). We pick the best scoring node (based on square error) and replace the current leaf node with the new node (addNode). Then both the left and right branch of the new node are added to the potential list of nodes to expand. To avoid overfitting, we only insert and hence expand nodes that have at least 6 examples. We pick the node with the worst score and repeat the process.

FitRelRegressionClause(S, P) is called instead of FitRelRegressionTree when learning clauses. FitRelRegressionClause uses the maximum clause length as the parameter N (we set this to 3) and beam size B (we set this to 10). We greedily try to find the best scoring clause (BC) with $\mathtt{length} \leqslant N$. In every iteration, we pick the current best performing clause from a queue for expansion. We add all the clauses that improve the score to our queue, while only keeping the best B clauses in the queue and ignoring the rest. We repeat this process till no expansions of clauses with $\mathtt{length} \leqslant N$ are possible. This method only learns one clause at a time. Hence for learning

Algorithm 3.4 FitRelRegressionClause(S, P):
Fit relational regression clauses to the dataset, S.

```
1: Beam := {P(X)}  ▷ Initialize with the target predicate
2: BestClause := P(X)
3: N := 3  ▷ Maximum clause length
4: B:= 10  ▷ Beam width
5: while ¬empty(Beam) do
6:   Clause := popFront(Beam)  ▷ Pick the best scoring clause to expand
7:   if length(Clause) ⩾ N then
8:     continue  ▷ Clause too long to be expanded
9:   end if
10:  C := addLiterals(Clause)  ▷ Possible expansions of the base clause
11:  for c ∈ C do
12:    c.score = SE(c)  ▷ Score of the expanded clause c
13:    if c.score ⩾ Clause.score then
14:      insert(Beam, c, c.score)  ▷ Add the expansion if it is better than the
         base clause
15:    end if
16:    if c.score ⩾ BC.score then
17:      BestClause := c  ▷ Update best scoring clause
18:    end if
19:  end for
20:  while length(Beam) ⩾ B do  ▷ Consider the best B clauses to expand
21:    pop(Beam)
22:  end while
23: end while
24: return BC
```

multiple clauses, we call this function multiple times during one gradient step and update the gradients before each call based on the currently learned clauses. In all our experiments we learn a maximum of 3 clauses in a single gradient step.

Learning Joint Models One of the key features of SRL methods is the ability to learn and reason about predicates and examples jointly. To handle multiple target predicates, we learn a joint model by learning trees or clauses for each predicate in turn. We use the MLN's learned for all the predicates prior to the current iteration to calculate the regression value for the examples. For example while learning the joint model for three predicates p, q and r, we iterate through each predicate and learn one tree at a time. Let us assume we iterate through the predicates in the order $\{p, q, r\}$ and we have learned the k^{th} tree for q. When computing the gradients for predicate r, we use the $k - 1$ trees learned for r along with the k trees learned for p and q. Also for efficiency, while learning a tree for the target predicate r, we do not consider the influence of that new tree on other target predicates p and q.

While learning a model with a single target predicate, all the Horn clauses in that model have the target predicate as the head of the clause. I have shown above that $nt_j(x)$ corresponds to the true groundings of the body of such Horn clauses which is also the number of non-trivial groundings of the clause. But if we are learning a joint model, the target predicate may appear in the body of a clause. I will show that for such clauses, $nt_j(x)$ corresponds to the negative of the number of non-trivial groundings of the clause. Computing $nt_j(x_i)$ this way allows us to compute the ψ values for every example quickly without grounding the entire MLN at every iteration.

Proposition:

$nt_j(x)$ corresponds to the negative of the number of non-trivial groundings of the clause.

Proof:

For simplicity, assume that the clause $C_j : p(X), q(X, Y) \to target(X)$ is learned for $target(X)$ in the first iteration. The proof presented can be generalized to any Horn clause. If we are learning a joint model over $target$ and p, this clause will be used to compute the regression value for $p(x)$ in the next iteration. In its CNF form, this clause can be written as

$C_j : \neg p(X) \vee \neg q(X,Y) \vee \mathtt{target}(X)$. When $p(x) = \mathtt{false}$, all the groundings of Y satisfy the clause, i.e.,

$\mathbf{n_j}(\mathbf{p}(\mathbf{x}) = \mathbf{false}) = |Y|$

When $p(x) = \mathtt{true}$, only the groundings of Y where $\neg q(x,Y) \vee \mathtt{target}(x) = \mathtt{true}$ satisfy the clause, i.e.,

$\mathbf{n_j}(\mathbf{p}(\mathbf{x}) = \mathbf{true}) = |\{y \in Y : \neg q(x,y) \vee \mathtt{target}(x)\}|$

Also, these groundings satisfy the clause irrespective of the value of $p(x)$, i.e., these are the trivial groundings of the clause. The difference between these two set of groundings are the groundings that do not satisfy the condition $\neg q(x,Y) \vee \mathtt{target}(x) = \mathtt{true}$, i.e.,

$\mathbf{n_j}(\mathbf{p}(\mathbf{x}) = \mathbf{false}) - \mathbf{n_j}(\mathbf{p}(\mathbf{x}) = \mathbf{true}) = |\{y \in Y : q(x,y) \wedge \neg\mathtt{target}(x)\}|$

These groundings are also the non-trivial groundings of the clause, as these are all the groundings of Y except the trivial groundings. We defined $nt_j(p(x))$ as $n_j(p(x) = \mathtt{true}) - n_j(p(x) = \mathtt{false})$, which is the negative of the number of non-trivial groundings of the clause.

3.5 Experiments for RFGB

I present the results of our structure-learning approaches on both RDNs and MLNs in this section. I denote the results for our boosted-RDNs algorithm as RDN-B and boosted-MLNs as MLN-BT (tree representation) and MLN-BC (clause representation). I also *italicized* the name of our methods in the result tables. I also present the results for our structure-learning approach where we learn only one tree (RDN), i.e., we do not perform boosting, but learn a single larger tree instead.

I evaluate our structure learning approaches for RDNs against the state-of-the-art structure-learning approach for RDNs - Proximity [3]. I compare our boosted structure learning approaches for MLNs against previous structure learning approaches for MLNs, namely LHL (Kok and

[3]http://kdl.cs.umass.edu/software/proximity.html

Domingos, 2009), Motif (Kok and Domingos, 2010) and BUSL (Mihalkova and Mooney, 2007). I present results for two variations of Motif - with short rules (Motif-S) and long rules (Motif-L). When a hand-written MLN model is available, I also present results for weight learning on this model.

I use the two-tailed Student's t-test (Hastie et al., 2001) to evaluate statistical significance (at $p = 0.05$) in our experiments. When our approach is statistically significantly better than the baseline approaches, I use **bold** letters. When one of our approach is statistically significantly better than the baseline and our other approaches, I use ***bold-italicized*** letters. I first show the results on three commonly used SRL datasets: WebKB, UW-CSE and IMDB. I follow it with additional experimental results in Section 3.5.4.

3.5.1 Evaluating the boosted RDN approach: WebKB

I use the WebKB dataset (Craven et al., 1998) to compare our boosting approach for RDNs against Proximity. I used this particular dataset since Proximity provides a query model needed by it for this dataset and it runs out of memory (on a machine with 8 GB RAM) for other standard datasets. The WebKB data set consists of web pages with their most commonly occurring words. It also contains the links between these web pages. The goal is to classify these web pages into multiple categories. We only present the results of binary classification of web pages into *Student* or *Non-student* pages. Following standard procedure, we create one fold for every school in the data set (Wisconsin, Washington, Cornell and Texas).

This dataset is provided by the Proximity software package, which also includes the query model required by Proximity as part of the package. Thus we did not have to make any transformations to the data set for Proximity. Proximity uses the query model to generate the set of possible parents. While learning the relational trees, Proximity searches for the best split from these set of parents using the *chi-square* test (Neville et al., 2003a). Our approach, on the other hand, grows the parents set as it grows the trees and uses *weighted variance* score to pick the best split.

In Table 3.1, I present the results on the Area Under ROC curve (AUC-ROC) and Area under Precision-Recall curve (AUC-PR; Davis and Goadrich 2006) averaged over the four folds. We ignore the predictions made by Proximity on other categories apart from Student to calculate their AUC since we are interested in the binary classification problem. Our RDN learning algorithms perform statistically-significantly better (at $p = 0.05$) than the Proximity learning algorithm. We tried increasing the depth of the trees in Proximity, but it did not improve its results. There is no statistically-significant difference between the AUC values of RDN-B and RDN.

Algorithm	AUC-ROC	AUC-PR
RDN-B	$\mathbf{0.980 \pm 0.018}$	$\mathbf{0.965 \pm 0.032}$
RDN	$\mathbf{0.980 \pm 0.020}$	$\mathbf{0.956 \pm 0.050}$
Proximity	0.753 ± 0.020	0.648 ± 0.028

Table 3.1: Cross-validation results on the WebKB data set.

3.5.2 Evaluating the boosted MLN approaches: UW-CSE

I next compare our two boosting algorithms for MLNs using the UW-CSE dataset. The UW-CSE dataset (Richardson and Domingos, 2006) was creating from University of Washington's Computer Science and Engineering department's student database (hence the name). The data set consists of details about professors, students and courses from 5 different sub-areas of computer science (AI, programming languages, theory, system and graphics). The dataset includes predicates such as $\mathit{professor}$, $\mathit{student}$, $\mathit{publication}$, $\mathit{advisedBy}$, $\mathit{hasPosition}$, $\mathit{projectMember}$, $\mathit{yearsInProgram}$, $\mathit{courseLevel}$, $\mathit{taughtBy}$, and $\mathit{teachingAssistant}$ and equality predicates such as $\mathit{samePerson}$, $\mathit{sameCourse}$ etc. For the UW-CSE data set, the goal is to predict the $advisedBy$ relationship between a student and a professor using the other predicates. There are 4,106,841 possible $advisedBy$ relations out of which 3380 relations are true. Since the dataset consists of five areas (or mega-examples), we performed five-fold cross-validation.

I compare our two boosting algorithms - tree-based (MLN-BT) and clause-based (MLN-BC) to state-of-the-art MLN structure-learning methods: LHL, Motif-S (short rules) and Motif-L (long rules). In addition to the methods describe above, we also compared against the handcoded MLN available on Alchemy's website with discriminative weight learning (denoted as *A-D*, which is short for Alchemy-Discriminative). We set the maximum number of leaves in MLN-BT to 8 and maximum number of clauses to 3 in MLN-BC. The beam width was set to 10 and maximum clause length was set to 3 for MLN-BC. We used 20 gradient steps (and hence learned 20 trees) on all our boosting approaches.

Table 3.2 presents the AUC-PR and CLL values, along with the training time taken by each method averaged over five folds. As can be seen, for the complete dataset both boosting approaches (MLN-BT and MLN-BC) perform significantly better than other MLN structure learning techniques on the AUC-PR values. Current MLN learning algorithms on the other hand are able to achieve lower CLL values over the complete dataset by pushing the probabilities to 0, but are not able to differentiate between positive and negative examples, as shown by the low AUC-PR values.

Since most relations such as `taughtBy, advisedBy` etc. are false in the real-world, the number of negatives can be order of magnitude more than the number of positives. In these cases, simply measuring CLL over the entire data set can be misleading. Predicting all the examples as the majority class (when the number of examples in one class are far greater than the other) can have a good CLL value, but a very low AUC-PR value (nearly 0). For example, consider a test set with ten negative and one positive example. A model that returns a probability of 0.1 for every example has a CLL of -0.14 and AUC-PR of 0.1. On the other hand, a model, which returns a probability of 0.5 on every negative example and 0.6 for the positive example, has a CLL of -0.29 and AUC-PR of 1.0. The first model can not differentiate between the positive and negative examples (hence AUC-PR=0), but it has a better CLL (lower is better) than the second one. Hence, we also present results with a lower skew, i.e., where the number of negative examples is twice the number of positives.

When we reduce the negative examples in the test set to twice the number of positives, our boosting techniques dominate on both the AUC-PR and CLL values, while the other techniques, which cannot differentiate between the examples, have poor CLL values. There is no significant difference between learning trees or clauses in the case of boosting MLNs.

Algo	All negatives		2X negatives		Training
	AUC-PR	CLL	AUC-PR	CLL	Time
MLN-BT	**0.21 ± 0.17**	-0.46 ± 0.36	**0.94 ± 0.06**	**-0.52 ± 0.45**	18.4 sec
MLN-BC	**0.22 ± 0.17**	-0.47 ± 0.14	**0.95 ± 0.05**	**-0.30 ± 0.06**	33.3 sec
Motif-S	0.01 ± 0.00	-0.06 ± 0.03	0.43 ± 0.03	-3.23 ± 0.78	1.8 hrs
Motif-L	0.01 ± 0.00	-0.07 ± 0.02	0.27 ± 0.06	-3.60 ± 0.56	10.1 hrs
A-D	0.01 ± 0.00	-0.08 ± 0.02	0.31 ± 0.10	-3.90 ± 0.41	7.1 hrs
LHL	0.01 ± 0.01	-0.06 ± 0.02	0.42 ± 0.10	-2.94 ± 0.31	37.2 sec

Table 3.2: Cross-validation results on the UW data set. MLN-BT = Boosting with trees, MLN-BC=Boosting with clauses, Motif-S=Motif with short rules, Motif-L=Motif long rules, A-D=Hand-coded rules with discriminative learning, LHL=Lifted Hypergraph learning.

3.5.3 Evaluating the boosted RDN and MLN methods: IMDB

We use the IMDB dataset to compare our boosted MLN approaches (MLN-BT and MLN-BC) and our boosted RDN approach (RDN-B) against other structure-learning approaches, namely LHL, BUSL, Motif-S (short rules) and Motif-L (long rules). The IMDB dataset was first used by Mihalkova and Mooney (2007) and contains five predicates: `actor`, `director`, `genre`, `gender`, and `workedUnder`. We do not evaluate the `actor` and `director` predicates because they are mutually exclusive facts in this dataset and easy to learn for all the methods. Also since gender can take only two values, we convert the `gender(person,gender)` predicate to a single argument predicate `female_gender(person)`. Following Kok and Domingos (2009), we omitted the four equality predicates. We performed five-fold cross-validation using the folds generated by Mihalkova and Mooney (2007) and averaged the results across all the folds. We perform inference over each predicate given all other predicates as evidence.

Algorithm	workedUnder	genre	gender
MLN-BT	0.90 ± 0.07	0.94 ± 0.08	0.45 ± 0.06
MLN-BC	1.00 ± 0.00	1.00 ± 0.00	0.39 ± 0.07
RDN-B	0.99 ± 0.02	0.91 ± 0.12	0.46 ± 0.18
BUSL	0.89 ± 0.11	0.94 ± 0.08	0.44 ± 0.08
LHL	1.00 ± 0.00	0.37 ± 0.09	0.39 ± 0.12
Motif-S	0.56 ± 0.16	0.52 ± 0.29	0.48 ± 0.08
Motif-L	0.48 ± 0.27	0.39 ± 0.03	0.46 ± 0.08

Table 3.3: AUC PR values on the IMDB data set.

Table 3.3 and 3.4 show the AUC-PR and CLL values respectively for the three query predicates: `workedUnder`, `genre`, and `gender`. Our boosting approaches perform better on average, on both the AUC-PR and CLL values, than the other methods. The BUSL method exhibits the best performance of the prior structure-learning methods in this domain. Our boosting algorithms are comparable or better than BUSL on all the predicates. For `workedUnder`, LHL has comparable AUC-PR values to our boosting approaches, while it is clearly worse on the other predicates. There is no

Algorithm	workedUnder	genre	gender
MLN-BT	-0.18 ± 0.06	-0.20 ± 0.09	-0.62 ± 0.05
MLN-BC	-0.11 ± 0.04	-0.12 ± 0.08	-0.84 ± 0.21
RDN-B	-0.88 ± 0.20	-0.25 ± 0.22	-0.76 ± 0.16
BUSL	-0.56 ± 0.05	-0.27 ± 0.09	-0.69 ± 0.01
LHL	-0.02 ± 0.01	-1.13 ± 0.23	-0.73 ± 0.05
Motif-S	-2.73 ± 1.66	-3.99 ± 2.70	-0.71 ± 0.08
Motif-L	-2.30 ± 1.16	-2.32 ± 1.15	-0.69 ± 0.06

Table 3.4: CLL values on the IMDB data set.

significant difference between the two versions of our boosting algorithms. As can be seen from Table 3.4, the MLN-based methods are marginally better than our boosted RDNs for predicting `workedUnder` predicate, while comparable for others.

3.5.4 Additional experiments

Next, I present additional experimental results on other domains for boosted RDNs and boosted MLNs. The results are similar to the three experiments presented above.

RDN: Movie Lens data set

We evaluate RDNs using the Movie Lens data set (Xu et al., 2009). Xu et al. created this dataset by randomly selecting a subset of 100 users and 603 movies from the ratings submitted by users on movielens.org. The task is to predict the preferences of the users on movies. The users have attributes `age`, `gender`, and `occupation`, while the movies have released `year` and `genre` attributes. Since we are interested in predicting the preference of the user, we created a new predicate, *likes* for every user-movie pair that is set to *true* if the user likes the movie and *false* otherwise. Originally, the ratings of the movie by the user were on a 5-star scale. We set the *likes* predicate to *true* if the rating of a movie by an user is greater than the average rating of all the movies by the same user. Typically, every user rated $(30 - 400)$ movies with a total number of 78,445 user-movie ratings. We performed 5-fold cross validation on the data by first splitting the data into five bins and then choosing four bins for training and fifth for testing. Hence, we use 80% of the data as the training set and evaluate on the other 20%.

Algorithm	AUC-ROC	AUC-PR	Training Time
RDN-B	0.611 ± 0.016	***0.602 ± 0.015***	332 s
RDN	0.587 ± 0.011	0.587 ± 0.011	6.25 s

Table 3.5: Cross-validation results on the Movie Lens data set.

Since this domain involved complex interaction between attributes, we introduced four aggregators for both RDN methods: (a) count of movies rated by the user, (b) count of ratings for a movie, (c) count of ratings of movies of a genre by the user and (d) count of the movies that the user likes in a genre. From Table 3.5, it can be observed that *RDN-B* is marginally better than RDN (statistically significant results in AUC-PR with p-value=0.023). As expected the time taken for boosting is higher when compared to learning a large single tree.

We attempted to use Proximity [4] (the default package for RDNs), but ran out of memory. If we restrict the search space to only use attributes of objects one relation away (e.g., only attributes of friends but not friends of friends) results were close to random. This is the key reason to use the TILDE (Blockeel and Raedt, 1998) implementation of learning trees (which uses clause learning techniques to incrementally introduce literals in trees) for learning RDN and not Proximity's RPT learning approach

[4]http://kdl.cs.umass.edu/proximity

(which first generates all the potential combination of literals as features before learning the trees).

Both the methods are significantly better than MLNs, where we used the hypergraph lifting algorithm in Alchemy to learn the structure (Kok and Domingos, 2009). Alchemy was not able to learn any meaningful structure (even with the aggregated predicates) and hence did not learn any useful model. We do not present Alchemy results here as all the examples are predicted to be *false*.

The results of *RDN-B* are quite similar to the best results reported using multi-relational Gaussian Processes (Xu et al., 2009), where the AUC for ROC was 0.627 for the same experimental setup.

RDN: Predicting adverse drug reactions

The next problem we considered is the prediction of adverse drug reactions on patients. The Observational Medical Outcomes Partnership (OMOP) designed and developed an automated procedure to construct simulated datasets[5] that are modeled after real observational data sources, but contain hypothetical people with fictional drug exposure and health condition occurrence. We used the OMOP simulator to generate a dataset of $10,000$ patients that included record of drugs and diagnoses (conditions) with dates. The goal is to predict drug use based on the conditions. For training, 75% of the data was sub-sampled, while the remaining 25% was used for testing. The test was conducted on 5 drugs with a training set of **1950** patients on the drug and a test set of **630** patients. predicting true if the predicted probability is > 0.5 and false otherwise.

The results are presented in Table 3.6. As can be seen, in this domain our approach *RDN-B* is significantly better than RDN in all the metrics except training time. This is due to the fact that in this domain, there were several different weak predictors of the class.

[5]http://omopcup.orwik.com

Algorithm	AUC-ROC	AUC-PR	Training Time
RDN-B	***0.824 ± 0.04***	***0.839 ± 0.04***	497.8 s
RDN	0.738 ± 0.04	0.736 ± 0.04	39.4 s
Noisy-Or	0.420 ± 0.08	0.582 ± 0.07	-

Table 3.6: Cross-validation results on the OMOP data set.

The third row of the table (Noisy-Or) is a relational method where we used Aleph (Srinivasan, 2004) to learn ILP rules. For each drug, we learned 10 rules, which are essentially Horn clauses with the target in the head, using Aleph. Some examples of the rules are:

```
on_drug(A) :-
   condition_occurrence(B,C,A,D,E,3450,F,G,H).
on_drug(A) :-
   condition_occurrence(B,C,A,D,E,140,F,G,H),
   condition_occurrence(I,J,A,K,L,1487,M,N,O).
```

The first rule identifies condition 3450 as interesting, while the second rule identifies two other conditions as interesting when predicting whether person A was on the current drug. Note that in first-order logic, the result is the disjunction of these rules (i.e., each rule is evaluated and the resulting concept is true if any of those rules are true). In SRL, we soften these rules using probabilities. Associated with each rule is a conditional distribution $P(head|body)$ and since all the rules have the same head, these rules are combined using the Noisy-Or combining rule[6] (Heckerman and Breese, 1994).

We have to learn two sets of parameters for combining the ILP rules using Noisy-Or. For every rule obtained from Aleph, we learn one parameter to capture the conditional probability distribution for the rule being true,

[6]$P(y = 1|\mathbf{x}) = 1 - \prod_i (1 - q_i P(y = 1|x_i))$ where q_i is the noise parameter and $\mathbf{x}$ are the parents of y.

$\mathsf{P(head|body)}$. We also learn one inhibition probability parameter (also the noise parameter) for the Noisy-Or combining rule. These parameters are learned using the EM algorithm presented in Natarajan et al. (2008), where it is derived for learning the parameters of the distributions and the noise parameters simultaneously. We run the EM algorithm for 50 iterations. Our current approach is significantly better than the ILP-SRL combination in all the evaluation metrics. We are unable to get Alchemy to learn rules for this data set due to the prohibitively large number of groundings in the data.

In this data set, boosting greatly helps in performance on almost all the drugs. While the non-boosted RDN is better than the ILP+combining-rule algorithm, RDN-B dominates clearly in this domain.

RDN: Cora dataset

We also evaluated our RDN approach on the Cora dataset (McCallum et al., 2000), a standard dataset for citation matching. Cora dataset was first created by Andrew McCallum, and later segmented by Bilenko and Mooney (2003). The dataset was later converted into relational format by Poon and Domingos (2007). In citation matching, the task is to identify citations that refer to the same paper, which as a sub-task may include matching the author, title and venue of citations.

For each citation we have information about the various fields using predicates such as author, title, venue, $\mathit{hasWordAuthor}$, $\mathit{hasWordTitle}$, and $\mathit{hasWordVenue}$. This task has multiple target predicates ($\mathit{sameAuthor}$, $\mathit{sameVenue}$, $\mathit{sameTitle}$, and $\mathit{sameBib}$) for identifying matching authors, venues, titles and the complete citation.

For a baseline approach, we used the B+N+C+T MLN presented in Singla and Domingos (2006) and available on the Alchemy website to compare against the boosted and non-boosted versions of RDN. Note that we are **not** learning the structure of MLN, but merely learn the weights of the MLN clauses. In the case of RDN and RDN-B, we learn the structure and parameters of the model.

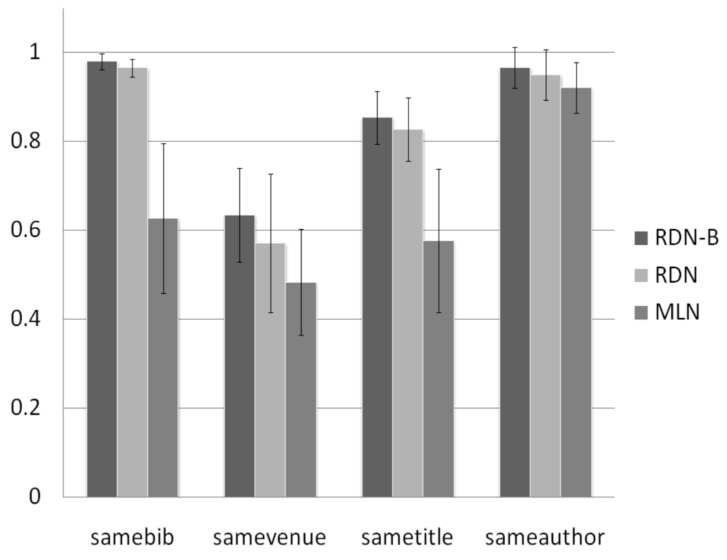
The area under curves of the PR curves for the entity-resolution task is presented in Figure 3.4a. The results are averaged over 5-folds for the four predicates that we mentioned earlier. As can be seen, RDN-B dominates the other methods consistently on all the four different predicates. MLNs exhibit good performance in the case of *SameAuthor* predicate, but are outperformed by the RDN methods in the other predicates. RDN learning that does not use boosting performs reasonably well, but is still outperformed by RDN-B in all the predicates.

The results are similar for the ROC curves and are presented in Figure 3.4b. As with the PR curves, these results are obtained over five-folds. RDN-B dominates for all the predicates in this case as well and is statistically significantly better on *SameBib* predicate.

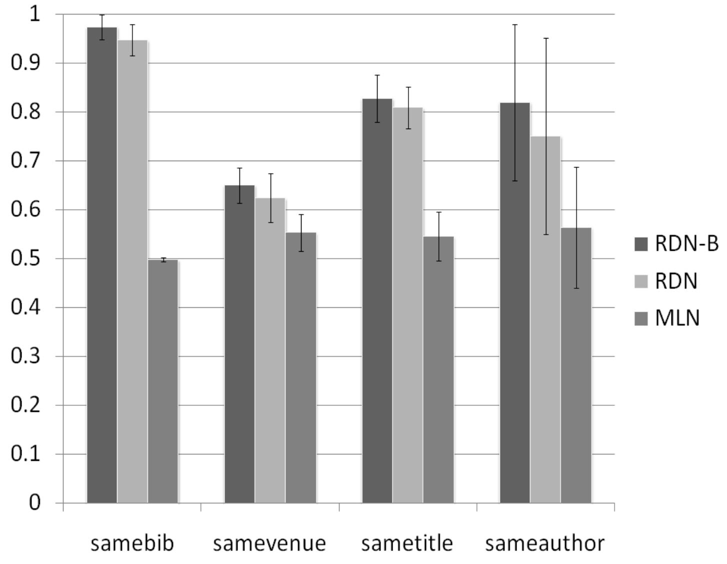
RDN: Citeseer dataset

Similar to the Cora dataset, we used the *Citeseer* dataset created by Poon and Domingos (2007) for performing *information extraction*. This dataset was first created by Lawrence et al. (1999). This dataset has 1563 citations and 906 clusters. It consists of four sections, each on a different topic. Over two-thirds of the clusters are singletons and the largest contains 21 citations (Poon and Domingos, 2007).

The Citeseer task can be viewed as an input to the Cora domain where the goal in Citeseer is to extract the fields from the citations. The extracted fields can then be used to perform entity resolution (identify citation clusters) using the extracted fields, as done in the Cora domain. Citeseer contains predicates such as the tokens at each position in a citation (`token`) and attributes about each token (`isDate, isDigit`, etc.). Given these facts, we try to predict the field type (`Title, Author, Venue`) for each position in the citation. For learning, the field types for each position are known;



(a) AUC-PR for Cora.



(b) AUC-ROC for Cora.

Figure 3.4: Area under curves for the entity-resolution task in Cora dataset. The error bars show one standard deviation from the mean.

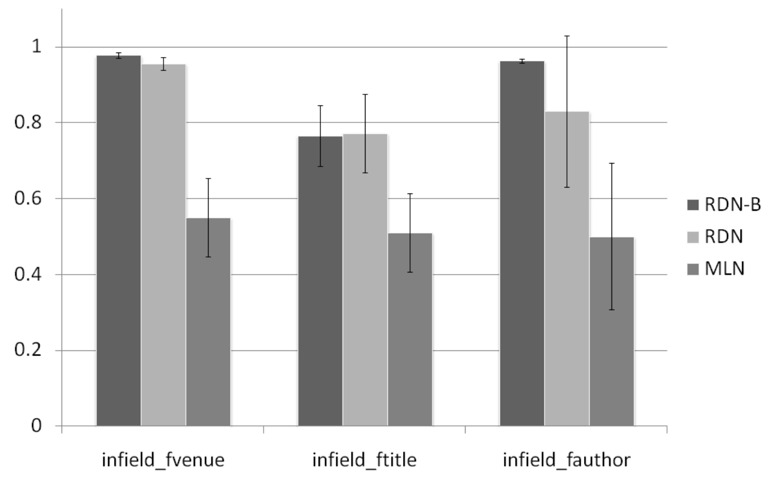
for inference, none of the field types are known and need to be inferred jointly.

We compare against the MLNs used by Poon et al. (2007). While their work performed entity resolution and information extraction jointly, we only use the features and rules specific to information extraction. Following the methodology used in prior work (Poon and Domingos, 2007), we create four-folds using the four sections. As with the previous experiment, we compare our learning method against the standard RDN learning and MLN presented in Poon and Domingos (2007). We learn the weights of the MLN clauses using Alchemy and discriminative weight learning setting.

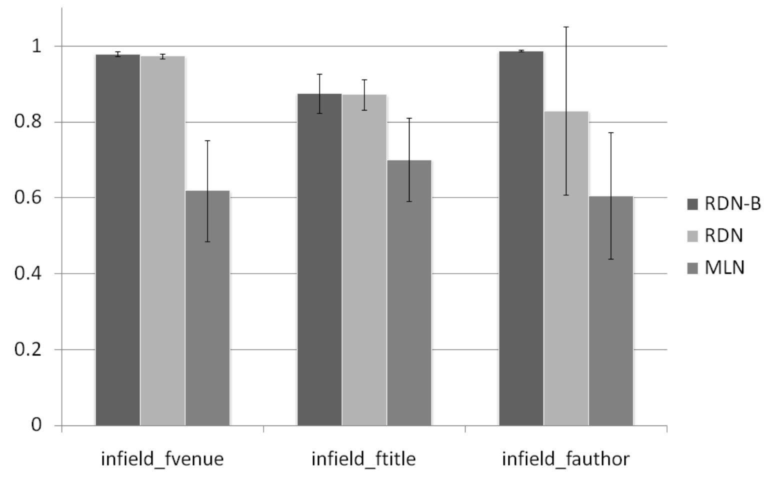
For our boosting methods, instead of using two arguments $\langle \texttt{Bib}, \texttt{Pos} \rangle$ in every predicate to indicate a particular position in the citation, we created objects of type BibPos. Hence, position P0001 in citation B1000 corresponds to a BibPos object, B1000_P0001. Since citations are of varying length, this creation of new objects allows us to avoid predicting labels for positions that do not exist. Also since our implementation can handle only binary classes, we changed *InField(Bib, Field, Pos)* to three different predicates: *infield_Field(BibPos)* where *Field={Fauthor, Ftitle, Fvenue}*.

The RDN learned using the boosting algorithm (RDN-B) is presented in Figure 3.6. The three predicates that are queried jointly in this dataset, viz., *Infield_fauthor, Infield_ftitle,* and *Infield_fvenue* are showed in shaded (dark) ovals. Recall that we created such predicates in the Cora dataset as well. The rest of the predicates are observed in the dataset and hence we do not learn the models for those predicates. This RDN is produced by collecting all the predicates that appear in different trees for a target predicate and making those predicates as the parents of the target predicate.

The average area under curves for PR-curves over five folds for the information extraction task are presented in Figure 3.5a. We compared RDN-B and RDN against MLNs, where we learn the weights using generative weight learning. We could not get discriminative learning of MLN



(a) AUC-PR



(b) AUC-ROC

Figure 3.5: Area under curves for the information extraction task in Citeseer dataset. The error bars show one standard deviation from the mean.

weights working as Alchemy seemed to run out of memory. The results are presented for the three `Infield` predicates. As can be seen, RDN-B

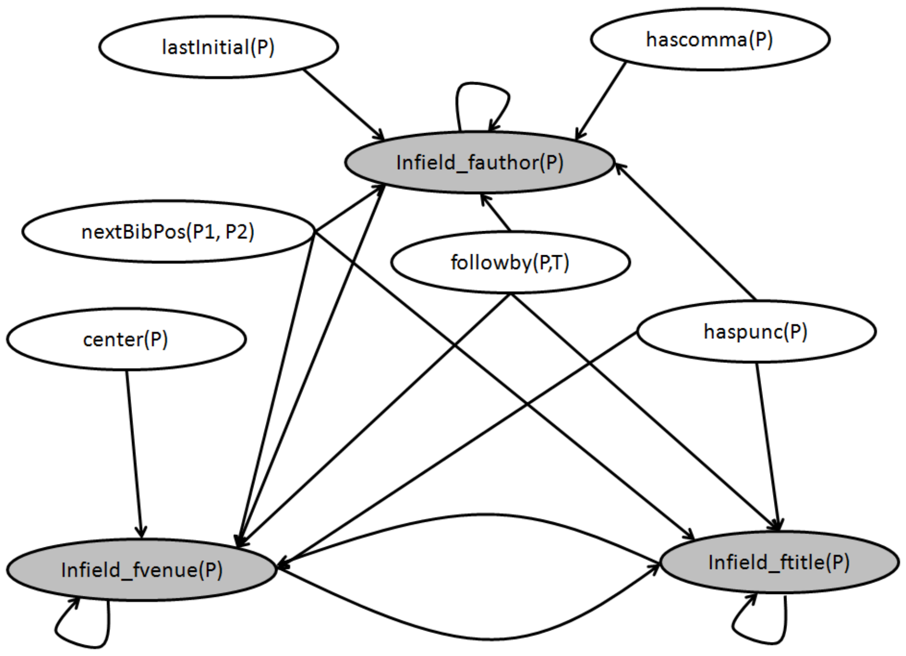


Figure 3.6: RDN learned using Boosting for the Citeseer dataset. The nodes that are shaded are the query nodes for which the models have been learned. The set of the nodes that are the parents for the query nodes are the set of all nodes that appear in the different regression trees.

greatly dominates MLNs on all the three predicates. On the other hand, RDNs without boosting are comparable in the *title* field to that of RDN-B. But for the other two fields namely, *author* and *venue*, the performance of RDN-B is significantly better than RDNs without boosting. The results are very similar in the case of ROC curves as well as shown in Figure 3.5b. RDN-B dominates MLNs in all the predicates while dominating RDNs significantly in the *author* predicate.

RDN: NFL dataset

We evaluate our method on a text-based dataset: the National Football League (NFL) dataset from the Linguistic Data Consortium[7] (LDC). This

[7]http://www.ldc.upenn.edu

dataset consists of articles of NFL games over the past two decades. The idea is to read the texts and identify concepts such as *score*, *team* and *game* in the text. As an easy example, consider the text, “Green Bay defeated Dallas $28 - 14$ in Saturday’s Superbowl game." Then, the goal is to identify *Green Bay Packers* and *Dallas Cowboys* as the teams, and 28 and 14 as their respective scores and the game to be a *Superbowl* game. There are cases in which the scores may not be directly specified in the text. The text could specify that “There were three touchdowns in the game" and the score must be inferred from this to be 21.

In addition to using the features from the annotated text, we also use the Stanford NLP toolkit[8] to create more features. These features are obtained from the parse trees constructed using the parser, the tags from the part-of-speech tagger, the named entity-recognizer, etc. The features were constructed at different levels: *word-level*, *sentence-level*, *paragraph-level* and *article-level*. Hence, the data set consisted of the annotations and linguistic information from the NLP parser. These features were provided as inputs to the different learning algorithms.

An alternate ensemble approach used commonly in literature is *bagging* (Breiman, 1996), where multiple models are learned over different subsamples of the examples. Unlike boosting, the models are not learned sequentially and can be even learned in parallel. The goal of this experiment is to empirically compare these two different ensemble approaches as well as a combination of the two approaches. Bagging is generally a variance-reducing technique whereas boosting is primarily a bias-reduction approach. By combining the two approaches, we can potentially benefit from both the techniques.

We compare four methods: RDNs, Bagging (random forests), Boosted RDNs, and Bagging of Boosted RDNs. The key idea in the *bagging+boosting* method is that we run the boosting algorithm for a few gradient-steps

[8]http://nlp.stanford.edu/software/index.shtml

and collect the regression trees and repeat the procedure 100 times and the gradient is averaged over these 100 sets of different trees. We perform five-fold cross validation to predict the above concepts.

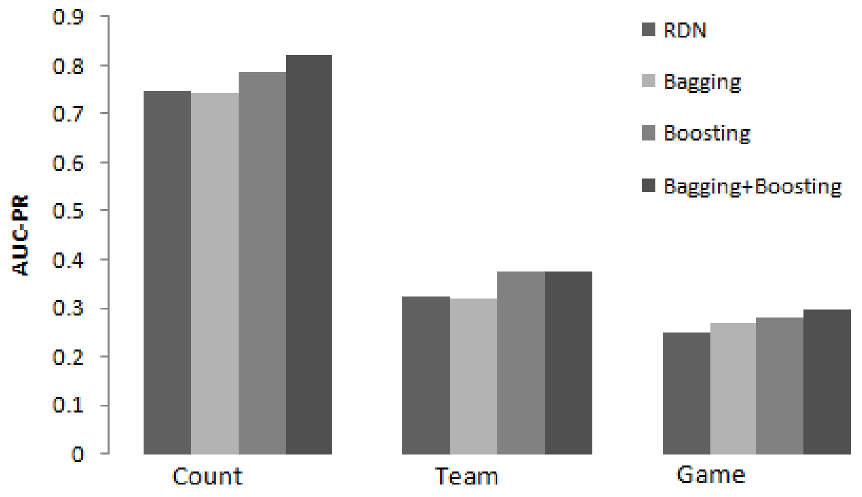


Figure 3.7: Precision-Recall values for the NFL corpora. The results are presented for RDNs with a single RRT, Bagged RDNs, Boosted RDNs and Bagged, Boosted RDNs.

The area under curve for precision-recall ($AUCPR$) curves of the different methods are presented in Figure 3.7. There are three different sets of graphs corresponding to the three concepts: *score(count)*, *game*, and *team*. As can be seen, for all the concepts, our boosted RDN method outperforms standard RDN learning method and the random forests (bagging) method. Interestingly, the method that uses both bagging and boosting helps in one concept (*count*), but is not statistically significantly better in other concepts.

As can be seen from the figure, while the concept of *score* is easier to learn, there is a lot of room for improvement for identifying the teams and the game. The key reason is that the natural text is quite ambiguous. For example, one article might mention the team as "San Francisco," the other article might mention them as “49ers" and the third article might use “SF

49ers." Hence, there is a need to perform co-reference resolution across articles.

MLN: Cora entity resolution

To evaluate our boosted MLN approach, we learn a joint model over `SameBib`, `SameVenue`, `SameTitle` and `SameAuthor` on the Cora dataset described earlier. Since this dataset is large, to speed up learning we sample 25% of the examples during every gradient step for MLN-BT. Similar to the UW dataset, we use a handcoded MLN(B+N+C+T) for Cora presented by Singla et al. (2006). We evaluate all the models jointly over the four target predicates given the evidence predicates. We use the queryEvidence flag for Alchemy weight learning and inference.

Algorithm	SameBib	SameVenue	SameTitle	SameAuthor
MLN-BT	$\mathbf{0.96 \pm 0.02}$	0.56 ± 0.17	0.71 ± 0.20	0.96 ± 0.04
MLN-BC	$\mathbf{0.96 \pm 0.02}$	0.68 ± 0.09	$\boldsymbol{0.82 \pm 0.13}$	0.98 ± 0.02
Alch-G	0.63 ± 0.17	0.45 ± 0.11	0.54 ± 0.14	0.90 ± 0.05
Alch-D	0.63 ± 0.17	0.48 ± 0.12	0.58 ± 0.16	0.92 ± 0.06
Motif-S	0.63 ± 0.16	0.45 ± 0.10	0.61 ± 0.17	0.93 ± 0.09
LHL	0.63 ± 0.17	0.45 ± 0.10	0.52 ± 0.15	0.91 ± 0.04

Table 3.7: AUC PR on the Cora testbed.

We perform five-fold cross-validation and average the results over all the folds. The AUC-PR and CLL values are presented in Table 3.7 and 3.8 respectively. MLN-BT has a slightly lower performance compared to MLN-BC since we need longer rules to accurately cluster entities. The entity-resolution task requires rules such as `Title(B1,T1)`, `Title(B2,T2)`, `SameBib(B1,B2)` $\rightarrow$ `SameTitle(T1,T2)`, which the greedy approach used in boosting may never find. Since any subset of the given rule would have little impact on the squared error, MLN-BT never learns such rules. MLN-BT scores two literals at a time for a given node and as

Algorithm	SameBib	SameVenue	SameTitle	SameAuthor
MLN-BT	$\mathbf{-0.39 \pm 0.04}$	-5.32 ± 1.88	-8.09 ± 2.97	-0.29 ± 0.14
MLN-BC	$\mathbf{-0.33 \pm 0.06}$	-5.12 ± 3.86	-11.18 ± 7.28	-0.60 ± 0.39
Alch-G	-5.58 ± 1.49	-4.27 ± 0.96	-5.14 ± 1.39	-8.87 ± 0.37
Alch-D	-4.95 ± 0.06	-4.08 ± 1.14	-4.34 ± 0.82	-3.32 ± 1.82
Motif-S	-2.54 ± 1.45	-1.80 ± 1.57	-2.79 ± 1.36	-1.57 ± 1.63
LHL	-5.99 ± 1.60	-4.20 ± 0.97	-5.11 ± 1.41	-8.80 ± 0.34

Table 3.8: CLL values on the Cora testbed.

a result learns short rules that only capture common words between the titles. Increasing the number of literals within each node to three increases the learning time drastically since every node in the tree searches over three literals. MLN-BC searches for clauses of length 3 but is computationally feasible since it learns fewer clauses than the nodes in a tree. Nevertheless, both methods are significantly better than other MLN learning methods. While structural Motifs and LHL methods are comparable when predicting the `SameAuthor` relationship, our boosting-based methods are significantly better for all the other relationships.

MLN: WebKB

The WebKB dataset was first created by Craven et al. (1998) and contains information about department webpages and the links between them. It also contains the categories for each webpage and the words within each page. This dataset was converted by Mihalkova and Mooney (2007) to contain only the category of each webpage and links between these pages. They created the following predicates: `Student(A)`, `Faculty(A)`, `CourseTA(C, A)`,`CourseProf(C, A)`, `Project(P, A)` and `SamePerson(A, B)` from these webpages. The textual information was ignored. We removed the `SamePerson(A, B)` predicate as it only had groundings with both the arguments being exactly same (i.e., `SamePerson(A,A)`). We evaluated all

Algo	AUC-PR		CLL	
	courseTA	courseProf	courseTA	courseProf
MLN-BT	0.426 ± 0.027	0.738 ± 0.034	$\mathbf{-0.603 \pm 0.057}$	$\mathbf{-0.406 \pm 0.050}$
MLN-BC	0.379 ± 0.031	0.750 ± 0.110	$\mathbf{-0.656 \pm 0.012}$	$\mathbf{-0.357 \pm 0.045}$
LHL	0.350 ± 0.046	0.460 ± 0.036	-2.274 ± 0.102	-2.243 ± 0.104
Motif-L	0.332 ± 0.014	0.637 ± 0.219	-2.282 ± 0.110	-2.198 ± 0.105

Table 3.9: Results on WebKB data set with twice the negatives to positives in the test set.

the methods over the `CourseProf` and `CourseTA` predicates since all other predicates had trivial rules such as `courseTA(C,A)` $\rightarrow$ `Student(A)`. We performed four-fold cross-validation where each fold corresponds to one university.

Table 3.9 and 3.10 present the results of the different algorithms in this domain. I present results for two different test cases here. First is the test set with all the negative examples in the test set and the second is the test set with twice the number of negatives as positives. Similar to the earlier case, in the test set with all negatives, current MLN methods such as LHL and Motifs exhibit good performance for the CLL evaluation measure for both the `courseTA` and `courseProf` predicates. On the other hand, the AUC-PR values are significantly lower than that of our boosting-based methods. This difference is magnified when we limit the number of negatives to twice the number of positives. In the latter case, even the CLL for the current MLN structure learning algorithms are significantly worse than our boosting methods. There is no statistically significant difference between the performance of the boosting methods.

3.5.5 Interpretability of the resulting trees

One issue with boosting approaches is the sacrifice of *comprehensibility* for better predictive *performance*. Since the trees learned in the later itera-

Algo	AUC-PR		CLL	
	courseTA	courseProf	courseTA	courseProf
MLN-BT	0.005 ± 0.003	0.029 ± 0.005	$\mathbf{-0.359 \pm 0.041}$	$\mathbf{-0.334 \pm 0.068}$
MLN-BC	0.004 ± 0.002	0.027 ± 0.007	$\mathbf{-0.479 \pm 0.041}$	$\mathbf{-0.304 \pm 0.010}$
LHL	0.004 ± 0.002	0.007 ± 0.002	-0.023 ± 0.011	-0.029 ± 0.005
Motif-L	0.003 ± 0.002	0.017 ± 0.009	-0.024 ± 0.011	-0.029 ± 0.005

Table 3.10: Results on WebKB data set using all the negative examples in the test set.

tions are dependent on the initial trees, these trees can not be interpreted individually. The entire conditional distribution is a set of regression trees and it does not make sense to interpret them individually. But this does not necessarily mean that the conditional distribution itself is not interpretable.

We can make the trees interpretable in two different ways. The first method is to collect all the predicates in the different trees and create a joint probability distribution over all the possible assignments. For every row in the joint probability distribution (i.e., an assignment to the predicates), we can compute the regression value returned by every tree and add them to compute the probability returned by the model. Once the probabilities are computed, it is possible to induce a single tree from the large joint distribution using standard tree-learning approaches. Though theoretically possible, this method scales poorly because the number of combinations is exponential in the number of the predicates.

Instead we use a more computationally feasible method of approximating the set of trees. The first-step is to predict the probabilities of the examples in the training set using all the trees. Now the new training set consists of the training examples along with their probabilities. We induce a single tree (with a high maximum tree depth) from these probabilistic examples. This resulting tree can then serve as a surrogate for our set of regression trees. This idea was earlier explored in the context of neural networks by Craven and Shavlik (1996). An example of the learned tree for predicting if a student is *advisedBy* a professor is presented in Figure 3.8.

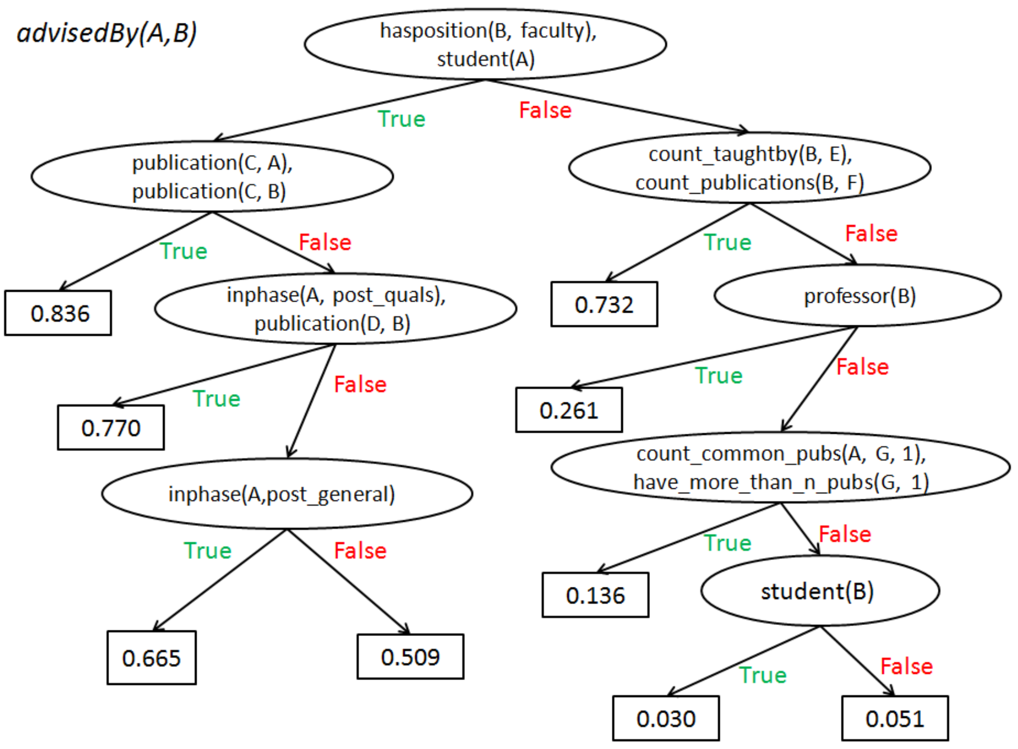

advisedBy(A,B)
hasposition(B, faculty), student(A)
True
False
publication(C, A), publication(C, B)
count_taughtby(B, E), count_publications(B, F)
True
False
True
False
0.836
inphase(A, post_quals), publication(D, B)
0.732
professor(B)
True
False
True
False
0.770
0.261
inphase(A,post_general)
count_common_pubs(A, G, 1), have_more_than_n_pubs(G, 1)
True
False
True
False
0.136
student(B)
0.665
0.509
True
False
0.030
0.051

Figure 3.8: A single tree induced from the set of regression trees learned for predicting the advisedBy relation between a professor and a student. Note that some of the nodes are conjunctions of predicates with "," denoting the conjunction symbol. The numbers at the leaves are the probability of the advisedBy relation being true.

3.5.6 Probability calibration

As mentioned before, relational datasets can be highly skewed with most relations being negative. As a result, learning approaches that maximize log-likelihood will push the probabilities closer to zero. Since most of the examples are negatively labeled, the error introduced by predicting lower probability values for positive examples has a marginal impact.

To handle this issue, calibration techniques (Platt, 1999; Zadrozny and Elkan, 2002) have been used to update the model probabilities. These techniques define a calibration function that takes the predicted probability as input and returns a calibrated probability. Two popular approaches for calibration used in literature are: Platt's calibration and Isotonic Regression.

Platt's technique (1999) applies a sigmoid function over the output from the classifier to calibrate the probabilities. The output of the classifier is scaled and shifted to align the calibrated probabilities with the true distribution. If f is the value returned by the classifier, Platt's calibrated probabilities is given by

$$P(f) = \frac{1}{e^{A \times f + B}}$$

where A and B are learned from the data to minimize the log-loss.

Isotonic Regression (IR; Zadrozny and Elkan 2002) finds a isotonic function $\hat{m}$ which minimizes the squared error between the true label and calibrated probabilities. Since $\hat{m}$ is an isotonic function (i.e., $f_i \geqslant f_j \Rightarrow \hat{m}(f_i) \geqslant \hat{m}(f_j)$), calibration via IR does not impact the area under the curves (since the order is unchanged). To find the isotonic function, the pair-adjacent violators (PAV) algorithm (Zadrozny and Elkan, 2002) is commonly used, which learns a piecewise constant function. The PAV algorithm begins with a calibration function which returns the example label for the model probability of that example. Adjacent examples that violate the isotonic constraint are then merged and the calibration function returns the mean probability instead. This process is repeated till there are no violators.

Since using the training set for calibration can bias the results (Platt, 1999), both of these approaches learn the calibration function over a held-out validation set. It has been shown that both Platt's and IR approach have similar results for calibrating boosted decision trees (Niculescu-Mizil and Caruana, 2005). Since our approach already computes a sigmoid function to compute the probabilities, we used IR to calibrate the probabilities.

Since the IR approach returns the same output probability for a range of input probabilities (called $range$ hereafter), the calibration function learned in IR is a $step$ function. But we prefer the probabilities to lie along the diagonal and hence we modified the IR approach by introducing a *slope* in the output function. The slope is introduced such that our calibration function returns higher probabilities on the higher end of a $range$ and lower probabilities on the lower end. Instead of a piece-wise constant function (step function), our new approach learns a piece-wise linear function. To do so, we start with the IR approach to get an isotonic function and then iteratively increase the slope (starting with a $slope = 0$ from the step function). We iterate over every $range$ and increase the slope such that the minimum output probability of a $range$ is higher than the maximum output probability of the $range$ before it on the input probability space and maximum output probability is lower than the minimum output probability of the $range$ after it. Algorithm 3.5 shows the pseudo-code of our approach.

To evaluate the calibration approaches, I create calibration curves on all the test examples. I perform five-fold cross-validation and use the model probabilities of all the examples from the five test folds. I partition the examples into 10 bins based on their model probabilities, i.e., examples with probability $\in [0, 0.1)$ fall in the first bin, $[0.1, 0.2)$ in the second bin and so on. For each bin, the proportion of positive examples is calculated to empirically estimate the true probability distribution for this bin. The same procedure is repeated for the calibrated probabilities. Estimated empirical probabilities for both uncalibrated and calibrated probabilities are plotted for every non-empty bin. In an ideal scenario, the empirical and predicted probabilities will match each other and all the points will lie on the diagonal.

I present the calibration curves for UW-CSE in Figure 3.9. As men-

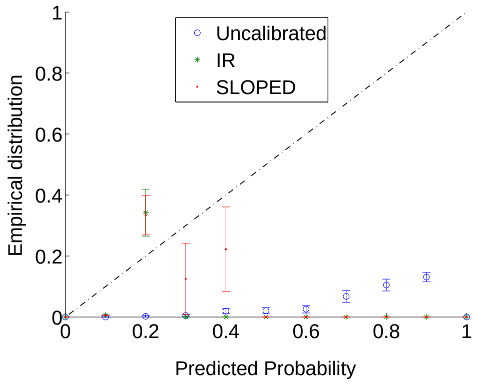
Algorithm 3.5 SlopedCalibrator(M, T)
Learn a calibration function for model M using the validation set, T.

```
1: $F := \mathtt{learnIRFunction}(M, T)$  ▷ Learn the Isotonic Regression function
2: $\mathtt{slopeUpdated} := \mathtt{true}$
3: while slopeUpdated do  ▷ Repeat till slope updated
4:   $\mathtt{slopeUpdated} := \mathtt{false}$
5:   for $1 \leqslant i \leqslant B$ do  ▷ Iterate over all calibration bins
6:     min := $F_{i-1}$.max_op  ▷ Previous bin's maximum output probability
7:     max := $F_{i-1}$.min_op  ▷ Next bin's minimum output probability
8:     slp := $F_i$.slope  ▷ Current slope of $i^{th}$ bin
9:     lowerLimit := (min+$F_i$.min_op)$/2$
10:    upperLimit := (max+$F_i$.max_op)$/2$
11:    ▷ Increase slope of the bin without exceeding the upper and lower limit
12:    updateSlope($F_i$, lowerLimit, upperLimit)
13:    if $|\mathtt{slp} - F_i.\mathtt{slope}| \geqslant \epsilon$ then  ▷ Check if non-trivial change in slope
14:      slopeUpdated := true
15:    end if
16:  end for
17: end while
18: return $F$
```

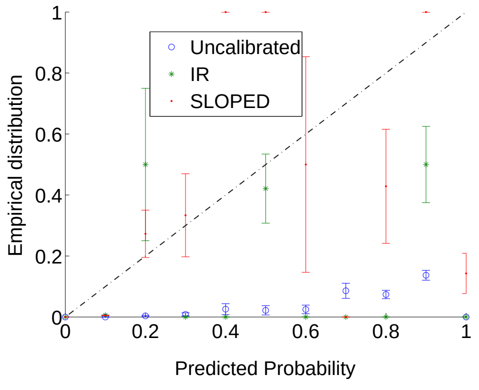
tioned earlier, we sub-sample the negative examples during every boosting iteration to be twice the number of positives. The y-axis is over the proportion of positive examples in each bin and the x-axis is over the bin probabilities. The error bars are calculated based on the standard error of binomial distribution[9] (Mitchell, 1997). We evaluate our approaches for different percentages of the training set used as validation set for calibration: 10% and 30%. With a smaller tuning set used for calibration, the difference in the calibrated and uncalibrated results is minimal. But with a larger validation set, both the *IR* and *SLOPED* calibrators have the empirical probabilities closer to the diagonal.

I show the calibration curves for WebKB in Figure 3.10. Unlike UW-

[9]Standard error = $\sqrt{\frac{p(1-p)}{n}}$, where p =proportion of positives and n =total number of examples.

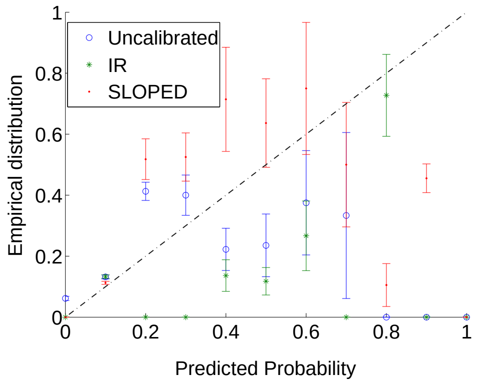


(a) 10% validation set

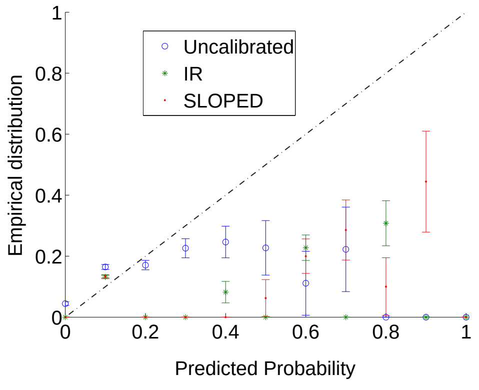


(b) 30% validation set

Figure 3.9: Calibration results on UW-CSE dataset with sub-sampled negatives to be twice the number of positives.



(a) 10% validation set

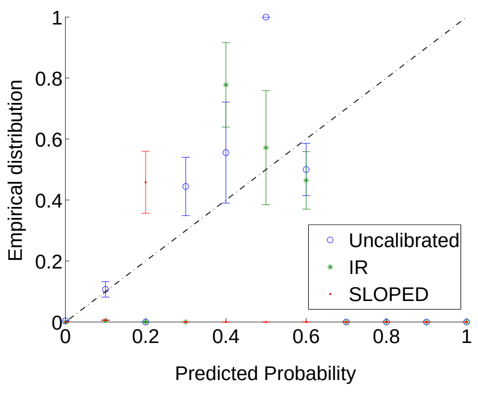


(b) 30% validation set

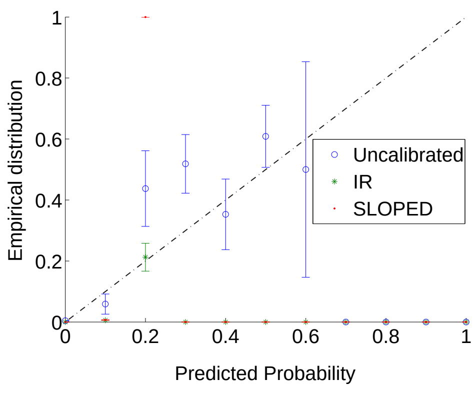
Figure 3.10: Calibration results on WebKB dataset with sub-sampled negatives to be twice the number of positives.

CSE, WebKB's uncalibrated probabilities are already close to the diagonal. With a larger validation set, the training set size reduces and the overall performance becomes worse. As a result both the calibrated and uncalibrated probabilities are further away from the diagonal.

I also present calibration results, with no sub-sampling of negative

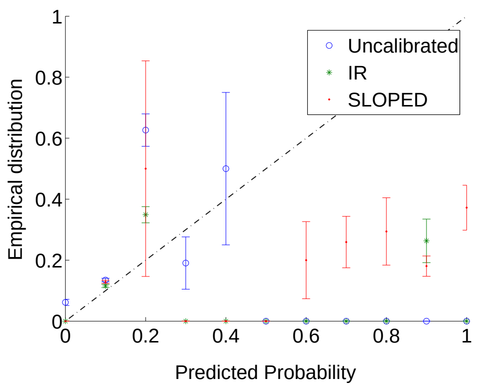


(a) 10% validation set

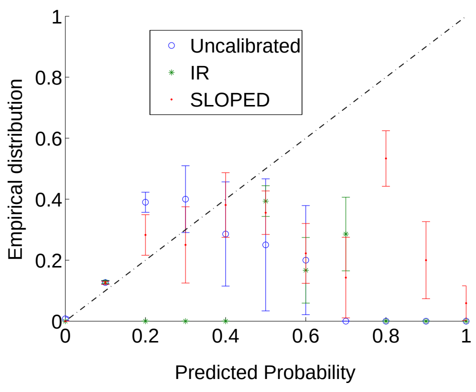


(b) 30% validation set

Figure 3.11: Calibration results on UW-CSE dataset with no sub-sampling of training examples.



(a) 10% validation set



(b) 30% validation set

Figure 3.12: Calibration results on WebKB dataset with no sub-sampling of training examples.

examples during training, in Figure 3.11 and 3.12. As seen from Figure 3.11, using all the training examples results in better calibrated probabilities without using any calibration function. On the WebKB dataset, however, there is no significant difference in the calibrated probabilities.

To further evaluate the impact of sub-sampling and the size of the

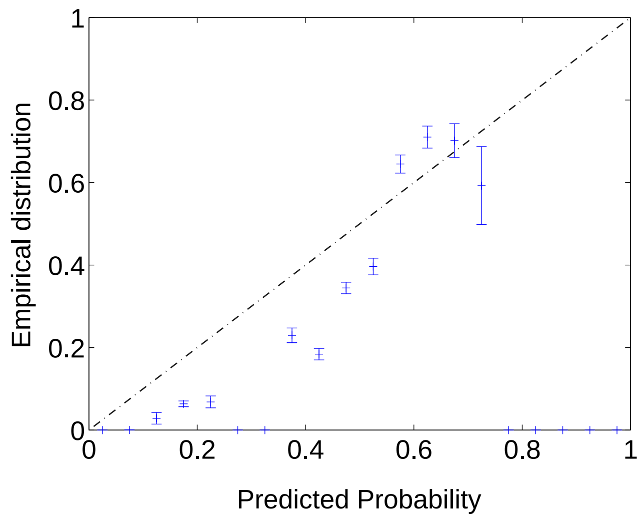
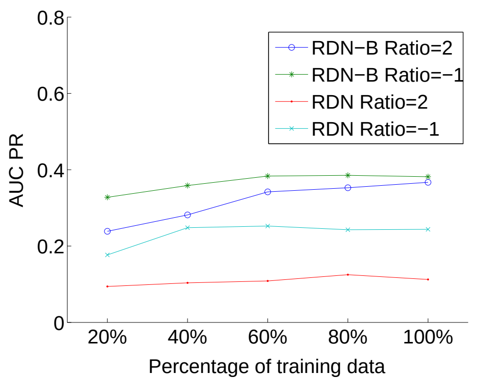


Figure 3.13: Calibration results on a proprietary dataset with 350K training and 20K test examples.

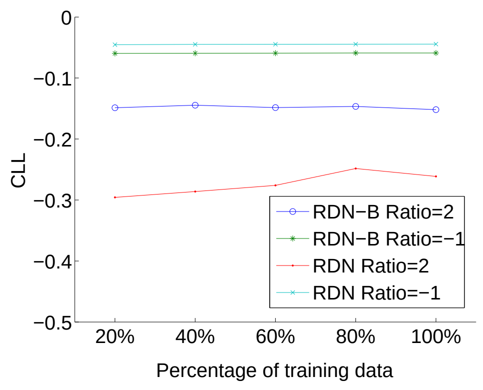
dataset, we ran RDN-B on a much larger dataset without any sampling of examples. While on sabbatical at Yahoo Research, Jude Shavlik ran RDN-Boost in parallel on a proprietary testbed with 350K training examples and 20K test examples. The calibration curve for the test examples is shown in Figure 3.13. As can be seen, the error bars are much smaller with more test examples and the curve is closer to the diagonal than on the other datasets. This shows that with more examples and no sub-sampling, RDN-B can achieve better calibrated results.

3.5.7 Learning curves

I present learning curves on UW-CSE in Figure 3.14. To create these curves, I calculate the AUC-PR and CLL values on the test set by learning the model using 20%, 40%, 60%, 80% and 100% of the training examples (and facts). We compare our boosting approach for RDNs (RDN-B) to the single tree learning approach (RDN). As mentioned earlier, we sub-sample the



(a) AUC PR



(b) CLL

Figure 3.14: Learning curves on UW-CSE dataset.

negative examples in every iteration of boosting to have twice the negatives as positive examples (marked with $\mathtt{Ratio} = 2$ in the figure). We also present a learning curve for RDN-B and RDN without any sub-sampling of examples, marked with $\mathtt{Ratio} = -1$ in the figures. The test examples are not sub-sampled for calculating the AUC-PR and CLL here. As can be seen in Figure 3.14, RDN-B outperforms RDN for any amount of training data in the UW-CSE dataset for predicting $\mathtt{advisedby}$ relationship. Also both of these approaches converge to the optimal performance with 60% of the training examples.

3.6 Discussion and Future Work

Rather than learning a single large model, I presented a structure-learning approach that uses functional gradient boosting to learn a sequence of small relational trees. We learn the structure as well as the parameters of the model simultaneously. I show how our relational functional gradient boosting approach can be applied for learning structure of RDNs as well as MLNs. I also compare our approach for learning each SRL model against their respective state-of-the-art learning approaches and show that boosting can learn more accurate models, usually in a fraction of the time.

Going forward, extending this work to directed relational models is an interesting future direction. Inference procedures can potentially take advantage of the context-specific independence (Boutilier et al., 1996) captured by the learned trees. I discuss these two approaches in detail in Section 8.1. I present a structural EM approach in the next section, which extends this work to handle missing data.

4 LEARNING IN THE PRESENCE OF MISSING DATA

SRL approaches allow one to handle noisy relational data without explicitly enumerating all the states. But most of these approaches apply the closed-world assumption (Getoor and Taskar, 2007), i.e., whatever is unobserved in the world is considered to be false. Research with missing data in SRL has mainly focused on learning the model's parameters. In such cases, algorithms based on classical EM (Dempster et al., 1977) have been developed for several SRL models, such as ones with combining rules (Natarajan et al., 2008). In this chapter, I present our work on an EM-based approach for learning structure of relational models. The work presented here has been accepted to the *Inductive Logic Programming* (ILP) conference and invited to the *Machine Learning* journal (Khot et al., 2014a).

4.1 Introduction

Li and Zhou (2007) learn the structure of a Probabilistic Relational Models from hidden data. They use maximum-likelihood trees to iteratively fill the missing values and update the structure, then use dependency analysis to learn the final structure. Since they are learning a directed model, they have to perform the expensive check of ensuring acyclicity in the ground model. Kersting and Raiko (2005) learn the structure of logical HMMs in the presence of missing data. Their approach, similar to Friedman's (1998) structural EM approach for Bayesian networks, computes the sufficient statistics over the hidden states and does a greedy hill-climbing search over the clauses.

Inspired by the success of structural EM on propositional graphical models (Friedman, 1998) and the success of boosting in learning SRL models, I present an EM algorithm using functional-gradient boosting. I derive and present the update equations of the E and M-steps of our algorithm.

One of the key features of our algorithm is that we consider the set of distributions in the models to be a product of conditional distributions. As shown in the previous chapter, this formulation allows us to learn different models such as MLNs (Khot et al., 2011), RDNs (Natarajan et al., 2012) and relational policies[1] (Natarajan et al., 2011). After deriving the EM algorithm, I show that adopting the standard approach of approximating the full likelihood by the MAP states produces the hard-EM approach. I empirically evaluate the proposed algorithm in different datasets and demonstrate the superiority of the proposed approach against several baseline algorithms.

4.2 Structural EM for Relational Functional Gradients

We derive an EM approach to learn the model's structure using functional gradients. We represent the observed data using **X** and the hidden data using **Y**. We use 1 and 0 to represent true and false respectively. Given a training set with missing data, we seek to maximize the log likelihood of the observed groundings. We average the likelihood function over all possible world states of the missing data (joint assignment over all hidden groundings) to compute the marginal probabilities of the observed groundings as shown below.

$$\begin{aligned}
\ell(\psi) &\equiv \log P(\mathbf{X} = \mathbf{x}|\psi) && \triangleright \text{Likelihood } (\ell) \text{ of observed data } \mathbf{x} \\
&= \log \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{x}; \mathbf{y}|\psi) && \triangleright \text{Marginalize over hidden data instantiations} \\
&= \log \sum_{\mathbf{y} \in \mathcal{Y}} \left\{ P(\mathbf{y}|\mathbf{x}; \psi') \frac{P(\mathbf{x}; \mathbf{y}|\psi)}{P(\mathbf{y}|\mathbf{x}; \psi')} \right\} && \triangleright \text{Multiply and divide by} P(\mathbf{y}|\mathbf{x}; \psi')
\end{aligned} \tag{4.1}$$

[1]A policy returns a distribution over the actions for a given world state.

Assume ψ' is our current estimate of the best model based on the log-likelihood function. We will derive gradient steps to find ψ that has a higher log-likelihood than ψ'. We then set the ψ obtained via these gradient steps as the new ψ' and iteratively find a better ψ. To make the iterative procedure clearer, we use ψ_t to represent the ψ' obtained after t iterations of the gradient steps. Similar to the RFGB approach presented in Chapter 3, we can start with a simple prior or some expert advice as the initial model, ψ_0. Unfortunately maximizing the log likelihood function directly is not feasible and so we maximize the lower bound on $\ell(\psi)$. To find the lower bound, we first rewrite Equation 4.1 using ψ_t,

$$
\begin{aligned}
\ell(\psi) =& \log \sum_{\mathbf{y}\in\mathcal{Y}} \left\{ P(\mathbf{y}|\mathbf{x};\psi_t) \frac{P(\mathbf{x};\mathbf{y}|\psi)}{P(\mathbf{y}|\mathbf{x};\psi_t)} \right\} \\
\geqslant & \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log \frac{P(\mathbf{x};\mathbf{y}|\psi)}{P(\mathbf{y}|\mathbf{x};\psi_t)} \qquad \triangleright \text{Jensen's Inequality} \\
= & \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log \frac{P(\mathbf{x};\mathbf{y}|\psi)P(\mathbf{x}|\psi_t)}{P(\mathbf{x};\mathbf{y}|\psi_t)} \qquad \triangleright \text{P(A | B) = P(A, B) / P(B)} \\
= & \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi) + \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\psi_t) \\
& - \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi_t) \\
= & \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi) + \log P(\mathbf{x};\psi_t) \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \\
& - \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi_t) \qquad \triangleright \mathbf{x} \text{ is constant w.r.t. } \mathbf{y} \\
= & \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi) + \log P(\mathbf{x};\psi_t) \qquad \triangleright \sum_{A} P(A \mid B) = 1 \\
& - \sum_{\mathbf{y}\in\mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi_t)
\end{aligned}
$$

The second term matches the log-likelihood function that we started with, except it is defined for ψ_t instead of ψ. The first and third term have similar form except for variations in the model used (ψ or ψ_t). To simplify these terms, we define a function $\mathcal{Q}$ as

$$\mathcal{Q}(\psi) \equiv \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x}; \psi_t) \log P(\mathbf{x}; \mathbf{y}|\psi) \tag{4.2}$$

We can now rewrite the lower bound of $\ell(\psi)$, derived above, as

$$\ell(\psi) \geqslant \mathcal{Q}(\psi) + \ell(\psi_t) - \mathcal{Q}(\psi_t) \tag{4.3}$$

Instead of finding ψ that maximizes $\ell(\psi)$, it is easier to find the ψ that maximizes this lower bound. Since ψ_t is constant with respect to the parameter ψ, we only need to find the ψ that maximizes $\mathcal{Q}(\psi)$. However, in many situations, finding a ψ that improves over $\mathcal{Q}(\psi_t)$ would suffice, since it will ensure that ψ has a higher log-likelihood than ψ_t as shown.

$$\begin{array}{rcll} & \mathcal{Q}(\psi) & \geqslant \mathcal{Q}(\psi_t) & \\ \Rightarrow & \mathcal{Q}(\psi) - \mathcal{Q}(\psi_t) & \geqslant 0 & \\ \Rightarrow & \mathcal{Q}(\psi) - \mathcal{Q}(\psi_t) + \ell(\psi_t) & \geqslant \ell(\psi_t) & \\ \Rightarrow & \ell(\psi) & \geqslant \ell(\psi_t) & \triangleright \text{From Equation 4.3} \end{array}$$

Hence in our iterative procedure, the $\mathcal{Q}$ function value and consequently the log-likelihood increases or stays the same after every iteration. Since there is no closed-form solution for finding the ψ function that maximizes $\mathcal{Q}(\psi)$, we use steepest descent with functional gradients. Running steepest descent until convergence would find the maxima of the $\mathcal{Q}(\psi)$ function (which might be a local maxima for some functions). Note that a single step of gradient descent with functional gradients involves learning one tree for every predicate. Running functional-gradient descent until convergence would result in learning a large number of trees for just one update to ψ_t. Hence, instead of maximizing the $\mathcal{Q}(\psi)$ function, we take few gradient steps (two in our experiments) to find the ψ function. We then use this ψ function as the base model ψ_t for the next iteration.

To derive the functional gradients, consider the definition of $\mathcal{Q}$ function

$$\mathcal{Q}(\psi) \equiv \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \log P(\mathbf{x};\mathbf{y}|\psi)$$

The second term in this equation is the joint likelihood of the observed data and an assignment for the missing data. In the previous chapter, I showed that RDNs approximate the joint distribution as a product of the conditional distributions. MLNs make a similar assumption by using pseudo-likelihood to make learning for MLNs tractable. Hence, both these models approximate a joint distribution such as $P(\mathbf{z};\psi)$ using a product of conditionals $\prod_{z\in\mathbf{z}} P(z \mid \mathbf{z} \setminus z;\psi)$. I use Z as the union of all the variables i.e. $Z = X \cup Y$. As mentioned in the background chapter, I use $\mathbf{z}_{-z}$ to denote $\mathbf{z} \setminus z$ and $\mathcal{Y}_{-i}$ to represent the world states for the set of groundings $\mathbf{y}_{-y_i}$ (i.e. $\mathbf{y} \setminus y_i$). Hence we can now rewrite $\mathcal{Q}(\psi)$ as

$$\mathcal{Q}(\psi) \quad = \quad \sum_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x};\psi_t) \sum_{z \in \mathbf{x} \cup \mathbf{y}} \log P(z|\mathbf{z}_{-z};\psi)$$

Next, we need to compute the gradients for each example (i.e. hidden and observed groundings of the target and hidden predicates), which will be used to learn the regression tree. The value returned by the ψ function also depends on other ground literals, since their values will influence the path taken in the regression tree. In the previous chapter, we included them as arguments to the function definition, i.e. $\psi(x;\mathbf{MB}(x))$. But $\mathbf{MB}(x)$ is observed and has the same values across all examples (the blanket varies across examples but the ground literal's Boolean values are the same) and so the function can be simplified to $\psi(x)$. However, with missing data, the assignment to the hidden variables $\mathbf{y}$ is not constant because each assignment to $\mathbf{y}$ may return a different value for a given example (due to different paths taken by examples). Hence, we include the assignment to the hidden variables in our function ($\psi(x; \mathbf{y})$) and compute the gradients for an example and hidden-state assignment.

4.2.1 Gradients for hidden groundings

We now derive the gradients of $\mathcal{Q}$ w.r.t the hidden groundings by taking partial derivatives of $\mathcal{Q}$ w.r.t $\psi(y_i; \mathbf{y}_{-i})$, where y_i is a hidden grounding. The value of $\psi(y_i; \mathbf{y}_{-i})$ is only used to calculate $P(y_i|\mathbf{x}, \mathbf{y}_{-i}; \psi)$ for two world states: where y_i is true and where y_i is false. So the gradient w.r.t. $\psi(y_i; \mathbf{y}_{-i})$ can be calculated as

$$\begin{aligned} \frac{\partial \mathcal{Q}(\psi; \psi_t)}{\psi(y_i; \mathbf{y}_{-i})} &= P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)\frac{\partial \log P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)}{\partial \psi(y_i; \mathbf{y}_{-i})} \\ &+ P(y_i = 0, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)\frac{\partial \log P(y_i = 0|\mathbf{x}, \mathbf{y}_{-i}; \psi)}{\partial \psi(y_i; \mathbf{y}_{-i})} \end{aligned}$$

As shown before, the gradients would correspond to the difference between the true value of y_i and the current predicted probability of y_i (i.e. $I(y_i = y) - P(y_i = y)$). As we have terms involving $P(y_i)$ for each value of y_i, we get two gradient terms.

$$\begin{aligned} & P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)(1 - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \\ + \quad & P(y_i = 0, \mathbf{y}_{-i}|\mathbf{x}; \psi_t)(0 - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \\ = \quad & P(y_i = 1, \mathbf{y}_{-i}|\mathbf{x}; \psi_t) - P(\mathbf{y}_{-i}|\mathbf{x}; \psi_t)P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)) \end{aligned} \qquad (4.4)$$

With the PLL assumption, the gradients can be written as $\prod_{j \neq i} P(y_j|\mathbf{x}, \mathbf{y}_{-j}; \psi_t)$ $[P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi_t) - P(y_i = 1|\mathbf{x}, \mathbf{y}_{-i}; \psi)]$. Intuitively, the gradients correspond to the difference between the probability predictions weighted by the probability of the hidden-state assignment.

4.2.2 Gradients for observed groundings

To compute the gradients for the observed groundings, we take partial derivatives of $\mathcal{Q}$ with respect to $\psi(x_i; \mathbf{y})$, where x_i is observed in the data. Similar to the gradients for hidden groundings, we use $\mathbf{y}$ as an argument in the ψ function and only consider the world states that match with the given argument. The gradient w.r.t. $\psi(x_i; \mathbf{y})$ is calculated as

$$\begin{aligned} \frac{\partial \mathcal{Q}(\psi; \psi_t)}{\psi(x_i; \mathbf{y})} &= P(\mathbf{y}|\mathbf{x}; \psi_t) \frac{\partial \log P(x_i|\mathbf{x}_{-i}, \mathbf{y}; \psi)}{\partial \psi(x_i; \mathbf{y})} \\ &= P(\mathbf{y}|\mathbf{x}; \psi_t)[I(x_i) - P(x_i = 1|\mathbf{z}_{-x_i}; \psi)] \end{aligned} \tag{4.5}$$

Similar to the hidden groundings, the gradients correspond to the difference between the predictions weighted by the probability of the hidden-state assignment.

4.3 The RFGB-EM Algorithm

I present the basic pseudo-code for our RFGB-EM (Relational Functional Gradient Boosting - EM) approach in Algorithm 4.1. Similar to other EM approaches, we sample the states for the hidden groundings based on our current model in the E-step and use the sampled states to update our model in the M-step. The function, ψ_t represents the model in the t^{th} iteration. The initial model ψ_0, can be as simple as a uniform probability for all examples or could be a model specified by a domain expert. We perform T iterations of the EM algorithm, where we fix $T = 10$ since results did not change much after 10 iterations in our experiments.

I present the algorithm for updating the model in Algorithm 4.2. The $\mathtt{updateModel}(W, D, \psi)$ function corresponds to the M-step. As mentioned before, we do not run gradient descent till convergence in our M-step. We take S gradient steps to find a better scoring model rather than

Algorithm 4.1 RFGB-EM:
Expectation-Maximization algorithm for RFGB to handle missing data.

```
Require: Hidden literals, H
Require: Observed literals, D
 1: Set initial model, $\psi_0$
 2: $t := 0$
 3: for $t < T$ do
 4:   W := sampleWorld(H, D, $\psi_t$)   ▷ E-step
 5:   $\psi_{t+1}$ := updateModel(W, D, $\psi_t$)   ▷ M-Step
 6: end for
 7: return $\psi_T$
```

the best possible model. In our experiments, S was set to two. This allowed us to amortize the cost of sampling the world states and run enough EM iterations in reasonable time without making the model too large.

Algorithm 4.2 updateModel(W, D, ψ):
Update the model ψ based on sampled states, W.

```
 1: $S := 2$   ▷ Number of trees learned in M-step
 2: for $i \leqslant S$ do
 3:   for $p \in P$ do   ▷ Iterate over target and hidden predicates, P
 4:     $E_p := \mathrm{sampleExamples}(D, p)$   ▷ Downsampled groundings of $p$
 5:     $D_p := \mathrm{buildDataset}(E_p, W, D, \psi)$
 6:     $T_p := \mathrm{learnTree}(D_p, D)$
 7:     $\psi = \psi + T_p$
 8:   end for
 9: end for
10: return ψ
```

Since each gradient step learns one tree, we learn $S \times T$ trees for each predicate after T EM iterations, which would be 20 trees in our case. We cycle over all the query and hidden predicates and learn one tree for each predicate per gradient step. We compute the gradients for the groundings of predicate p given by E_p, using the world states W, observed data D

Algorithm 4.3 buildDataset(E_p, W, D, ψ):
Build regression dataset for boosting.

```
1: $D_p := \emptyset$
2: **for** $e \in E_p$ **do**   ▷ *Iterate through examples*
3:   $\Delta_e := 0$
4:   **for** $w \in W$ **do**   ▷ *Iterate through sampled worlds*
5:     $\Delta_e := \mathtt{gradient}(e, w)$
6:     $D_p := D_p \cup \ < e, \Delta_e >$
7:   **end for**
8: **end for**
9: return $D_p$
```

and current model ψ. We then learn a relational regression tree using this dataset and add it to our current model.

As shown in the previous chapter, RDNs and MLNs use different scoring functions for learning the relational regression trees. So the $\mathtt{learnTree}$ function depends on the SRL model being used here. Relational datasets typically have many more negative examples than positive examples. We down-sample the negative examples during every tree learning step by randomly selecting negatives such that there are twice as many negative as positive examples. It has been shown that ensemble methods can perform better if the majority class is down-sampled (Chan and Stolfo, 1998).

The input examples to our regression tree learner are of the form $< (z; \mathbf{y}), \Delta >$. For every ground literal $z \in \mathbf{x} \cup \mathbf{y}$, we calculate the gradients for an assignment to the hidden variables. Algorithm 4.3 describes the $\mathtt{buildDataset}$ function used to compute the gradients for the examples. For every ground literal e and every world state w (i.e., $\mathbf{y}$), we compute the gradient of the example ($\mathtt{gradient}(e, w)$). For examples that are observed, we use Equation 4.5 to compute $\mathtt{gradient}(e, w)$ and for examples that are hidden, we use Equation 4.4.

I also present the high-level overview of our RFGB-EM (Relational Functional Gradient Boosting - EM) approach in Figure 4.1. Similar to other EM approaches, we sample the states for the hidden groundings based on our current model in the E-step and use the sampled states to update our model in the M-step. ψ_t represents the model in the t^{th} iteration. The initial model, ψ_0 can be as simple as a uniform probability for all examples or could be a model specified by an expert. We sample certain number of assignments of the hidden groundings (denoted as $|W|$) using the current model ψ_t. Based on these samples, we create regression examples which are then used to learn T relational regression trees. The learned regression trees are added to the current model and the process is repeated.

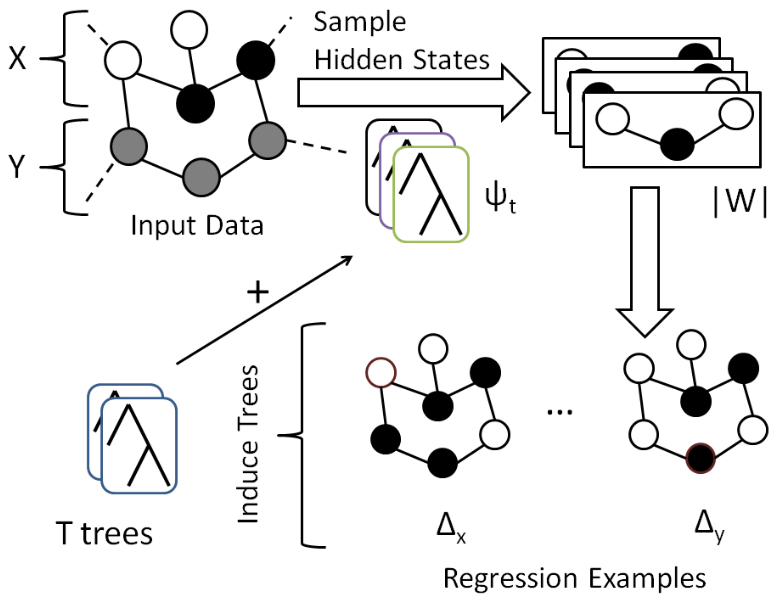


Figure 4.1: RFGB-EM flowchart. Shaded nodes indicate variables with unknown assignments, while the white (or black) nodes are assigned true (or false) values. The input data has observed (indicated by **X**) and hidden (**Y**) groundings. We sample $|W|$ assignments of the hidden groundings using the current model ψ_t. We create regression examples based on these samples, which are used to learn T relational regression trees. The learned trees are added to the current model and the process is repeated.

Approximations for the E-step As mentioned before, we approximate the log-likelihood with the pseudo-log-likelihood, as shown in the derivation. This approximation is necessary to make the approach computationally feasible and has been used in many structure and weight-learning approaches for SRL (Biba et al., 2008b; Mihalkova and Mooney, 2007; Kok and Domingos, 2009, 2010).

Computing probabilities for all possible world states would be exponential in the number of hidden groundings. This would also result in computing the gradients for all examples for each one of these world states. Hence we use Gibbs sampling to generate $|W|$ samples from the distribution $P(\mathbf{y}|\mathbf{x};\psi_t)$ to approximate all the world states, $\mathcal{Y}$. Since our gradients are weighted by the probability of the hidden-state assignment, an unlikely assignment will result in small gradients and thereby have little influence on the learned tree. Hence, we can sample the most likely hidden-state assignments to approximate the gradients. This is analogous to the Monte Carlo Expectation Maximization (MCEM) approach used for high dimensional data (Wei and Tanner, 1990). I refer to this approximation of the RFGB-EM approach as SEM-W (S stands for *Structural*), where W is the number of worlds sampled. Hence, SEM-1 corresponds to sampling the most likely assignment and corresponds to the hard-EM approach.

4.4 Adaptations of RFGB-EM

I now present how to adapt the RFGB-EM approach for RDNs, MLNs, and imitation learning. I also present experimental results for these adaptations. I use *SEM* to represent the structural EM approach which uses Gibbs sampling for generating the samples. I present results for SEM with a suffix to indicate the number of hidden state samples used, i.e., $|W|$ mentioned in the previous section (e.g., *S-10* uses ten samples while *S-1* uses the single MAP estimate). *S-10* corresponds to the soft-EM approach whereas *S-1* corresponds to the hard-EM approach. I present the results of using RFGB without using EM while setting all hidden groundings to false, i.e., using the closed-world assumption (*CWA*). This is essentially our prior work on RDNs (Natarajan et al., 2012), MLNs (Khot et al., 2011) and imitation learning (Natarajan et al., 2011). Each of these methods are run for 10 gradient iterations. We observed that this number was enough for convergence in all our domains. When our approach is statistically significantly better than the baseline approaches, I use **bold** letters. When one of our approach is statistically significantly better than the baseline and our other approaches, I use ***bold-italicized*** letters.

4.4.1 RDN adaptation

While learning RDN structure using functional-gradient boosting, all the trees are learned for a given target predicate before going on to the next predicate. Since the gradients for each predicate are independent of the model for other predicates, one can learn all the trees independently. In our EM approach, we update the hidden world states after every two iterations and hence for every predicate we learn two trees at a time. We then resample the hidden states and use the sampled states for the next two iterations. We use relational regression trees with the weighted-variance scoring function for fitting the gradients for each example. Hence the $\mathit{learnTree}$ function in Algorithm 4.2 can use any off-the-shelf relational regression tree learner. To evaluate the performance of our EM approach in RDNs, I present results on three datasets: Disjunctive (synthetic) dataset, UW-CSE and IMDB.

Disjunctive dataset

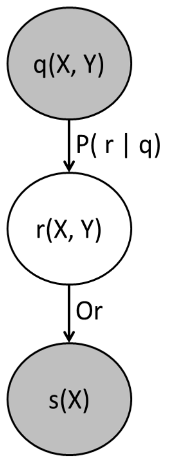
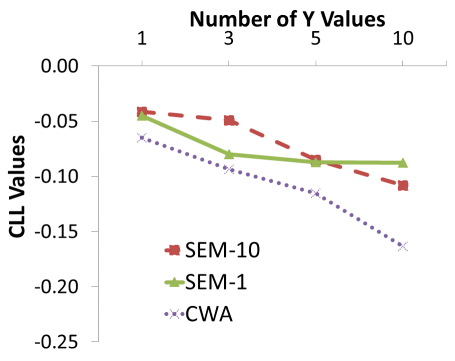


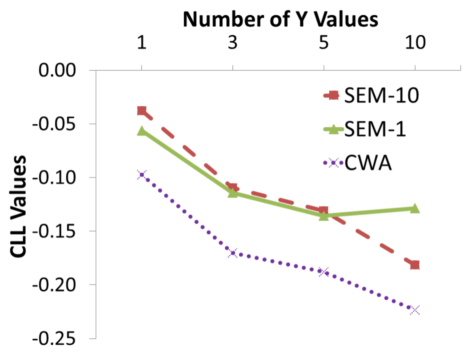
Figure 4.2: Synthetic experiment data generation. Predicate r is the hidden predicate and s is the target predicate.

We created a synthetic dataset using disjunctions (hence the name) to evaluate our EM approach for RDNs. We generated groundings of $q(X, Y)$ using a prior distribution of $P(q) = 0.9$ where $X = \{1 \ldots 100\}$ and varied Y to have four different values $|Y| \in \{1, 3, 5, 10\}$. Similarly, we also vary the amount of hidden data (Hidden % indicates the percentage of the groundings being hidden). We generate $r(X, Y)$ given $q(X, Y)$ using the conditional probability distribution $\{ P(r|q) = 0.1, P(r|\neg q) = 0.85 \}$. We then combine $r(X, Y)$ for different values of Y using an OR condition to generate $s(X)$. Hence $s(X)$ given $r(X, Y)$ is a deterministic rule where $s(X)$ is true if for some Y, $r(X, Y)$ is true. We show this process in Figure 4.2. We used r as the hidden predicate and s as the query predicate. We generated ten synthetic datasets with randomly sampled r groundings as hidden for $|X| = 100$. We independently train one model on each of the ten datasets. For each model, we evaluated the model on the other nine datasets that were not used for training this model. We average the results from the ninety runs.

The results on this domain are presented in Figure 4.3 (higher is better). We hide 20% and 40% of the groundings of the hidden predicates to simulate different levels of missing data. We only present the CLL values since the AUC-PR values are nearly equal for all the approaches. The EM approaches outperform CWA in all scenarios thereby affirmatively answering *Q1* for this domain. *SEM-10* outperforms both *SEM-1* and *CWA* methods on this dataset for $|Y| = 1$ and $|Y| = 3$, whereas *SEM-1* outperforms the others for $|Y| = 10$. Although the difference is very small in some cases, it is statistically significant (except for $|Y| = 5$ where *SEM-10* has similar performance to *SEM-1*).

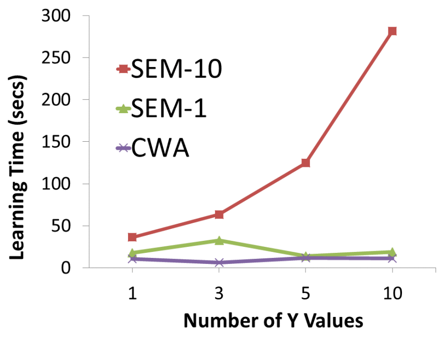


(a) 20% missing data



(b) 40% missing data

Figure 4.3: Results on the Disjunctive dataset.



(a) Average learning time on the Disjunctive dataset.

As we increase the number of values that can be taken by Y, we increase the number of possible hidden states. With just 10 samples, *SEM-10* is able to capture a relatively large space of the possible assignments to the hidden states for Y = 1 and Y = 3. On the other hand for Y = 10, both *SEM-10* and *SEM-1* capture a very small number of hidden-state assignments compared to the total number of possible assignments. As a result, the simpler *SEM-1* is able to perform better when |Y| = 10. Increasing the number of sampled states for soft-EM would improve the performance, but at the cost of learning time.

We also present the change in learning time of the various approaches with increasing missing data (by varying the |Y| values) in Figure 4.4a. We averaged the learning times across the ten folds for 20% missing data. Both *SEM-1* and *CWA* have similar learning times showing that the sampling step for this dataset is not computationally intensive. Since we used a heterogeneous cluster of machines, a slower machine may have introduced a small bump, viz. at $|Y| = 3$. On the other hand, learning time for *SEM-10* increases exponentially with increase in the number of missing values. This is primarily due to the fact that every node in the tree has to be scored for every sampled world state in W (by adding and removing the required facts every time). As the number of hidden groundings increase, the size of each sampled state increases requiring more operations during the learning phase. Increasing $|W|$ would further increase the learning time, but could improve the accuracy.

UW-CSE

We use the UW-CSE dataset described in Section 2.5 to evaluate the EM approach for learning RDNs. We randomly hid groundings of the `tempAdvisedby`, `inPhase`, and `hasPosition` predicates during training. Due to these hidden groundings and the different type of SRL model being learned, our numbers are not exactly comparable to the ones reported in Section 3.5.2. We performed five-fold cross-validation and present the CLL values in Table 4.1. We do not present the AUC PR values since the difference is not statistically significant. We also varied the amount of hidden data in our experiments ("Hidden %" in the table indicates the percentage of the groundings being hidden).

Table 4.1: CLL values for UW-CSE

Hidden %	20%	40%
SEM-10	**-0.168**	**-0.170**
SEM-1	**-0.150**	**-0.151**
CWA	-0.187	-0.192

Table 4.2: CLL values for IMDB

Hidden %	10%	20%
SEM-10	**-0.501**	**-0.551**
SEM-1	**-0.423**	**-0.467**
CWA	-0.586	-0.80

In general, the EM methods perform statistically significantly (with p-value < 0.05) better than the closed-world assumption. It appears that in this domain, using a single sample for the hidden state has the same performance as that of using 10 samples. This is in line with most EM algorithms, where using a single state (MAP) approximation generally suffices.

IMDB

We also use the IMDB dataset described in Section 2.5 to evaluate the EM approach. We predict the `gender` predicate given all the other predicates. We randomly hid the groundings of `actor` and `workedUnder` predicates during learning and inference. Again due to these hidden predicates, our numbers are not comparable to the ones reported earlier. We performed five-fold cross-validation.

I present the CLL values for hiding 10% and 20% of the groundings of the two hidden predicates in Table 4.2. Similar to the disjunctive dataset, there is no statistically significant difference between any two of the three methods in the AUC-PR values and hence are not reported here. In general, the EM methods perform statistically significantly (with p-value < 0.05) better than the closed-world assumption. Between the two EM methods, using one sample is sufficient to capture the underlying distribution and, hence, the simpler SEM-1 has a higher CLL value than SEM-10.

4.4.2 MLN adaptation

For applying our approach to learn MLNs, we learn relational regression trees for the gradients, but with the modified scoring function as described in Section 3.4. Unlike for RDNs, to compute the marginal probability of any example, trees for all the predicates are used. Hence during learning, a single tree is learned for each predicate and the gradients are computed based on the trees learned in earlier iterations. In our EM approach, we

Table 4.3: Results on the Cancer dataset.

Hidden %	20%		40%	
Algorithm	CLL	AUC-PR	CLL	AUC-PR
SEM-10	**-1.445**	**0.482**	**-1.315**	**0.510**
SEM-1	-1.648	**0.483**	-1.586	0.500
CWA	-1.629	0.478	-1.693	0.488

resample the hidden states after two such iterations over the target and hidden predicates. I also presented an approach (in Section 3.4) to learn MLN clauses to fit the gradients. Here, I only present the results for learning trees for EM, but it is trivial to extend this work to learn clauses.

The cancer MLN is a popular synthetic data set (Kersting et al., 2009; Domingos and Lowd, 2009). We created a friend network with 500 people using a symmetric predicate, `friends(X,Y)`, where each pair of people are friends with a probability of 0.01. Each person has three attributes: `stress(X)`, `cancer(X)`, and `smokes(X)`. The `stress` attribute for each person is set using a Bernoulli distribution. A person is more likely to smoke if he is stressed (set using a Bernoulli distribution) or has friends who smoke (set using an exponential distribution). Similarly, a person is likely to have cancer if he smokes (set using a Bernoulli distribution) or he has a lot of friends who smoke (set using an exponential distribution). The more smoker friends a person has, the more likely he is to get cancer. Such rules can be captured by MLNs since the probabilities are proportional to the number of groundings of a clause (e.g., $\mathtt{smokes}(y) \wedge \mathtt{friend}(x, y) \rightarrow \mathtt{smokes}(x)$). The target predicate is `cancer`, while `smokes` has some missing groundings. We trained the model on 10 generated datasets with randomly sampled hidden data and evaluated each model on other nine datasets and present the average results.

As seen in Table 4.3, *SEM-10* mostly outperforms the other approaches, both in terms of CLL and AUC-PR. For 20% missing data, there is no statistically significant difference between the two EM approaches, but both methods outperform CWA. Unlike the previous domains, SEM-10 is at least as good as or better than SEM-1 in this domain.

Since Alchemy does not have a mechanism to handle missing data for structure learning, we ran weight learning (generative with 10000 iterations and 10^{-5} threshold) on hand-written rules and learned the weights using the missing data weight learning approach of Alchemy. The AUC PR values were around 0.6. This shows that simply learning the parameters is reasonably comparable to our models that learn both the structure and parameters with hidden data.

For this synthetic dataset too, we generated ten synthetic datasets with randomly sampled $smokes$ groundings as hidden for $|X| = 100$. We independently train one model on each of the ten datasets. For each model, we evaluated the model on the other nine datasets that were not used for training this model. We average the results from the ninety runs.

4.4.3 Imitation learning adaptation

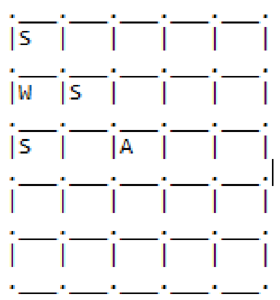
By observing the actions of an expert for every state in a domain (called *trajectories*), imitation learning (Ratliff et al., 2006) builds a model (called *policy*) that imitates the expert's action for a given state. In other words, imitation learning problem can be stated as:

Given: A set of states, **S**
A set of actions, **A**
Training trajectories, $\mathbf{T} = \{t_1, \ldots, t_n\}$, where a trajectory $t_i = \{s_1, a_1, s_2, \ldots, a_{k-1}, s_k\}$ is a sequence of states and actions.
Learn: Policy $\pi : \mathbf{S} \rightarrow \mathbf{A}$ that returns the action (or a

distribution over them) for any state that imitates the training trajectories.

For relational imitation learning using RFGB, the actions are the target predicates and the state predicates are used in the trees for the action predicates (Natarajan et al., 2011). For partially-observed imitation learning task, the set of *target* predicates, P contain all the action predicates and the hidden state predicates. We then learn relational trees to predict each action while updating the hidden values. Natarajan et al. (2011) learned all the trees for every action independently, whereas in our EM approach we learn two trees for every predicate before resampling the hidden states.

To evaluate the performance of our EM approach on the imitation learning task, we perform imitation learning in a partially-observed relational domain. We created a simple version of the Wumpus task (Russell and Norvig, 2003), where the location of wumpus is partially observed. We use a 5x5 grid with a wumpus placed at a random location in every training trajectory. The wumpus is always surrounded by stench on all four sides. We do not have any pits or breezes in our task. Figure 4.5 shows one instantiation of the initial grid locations. The agent can perform eight possible actions: four move actions in each direction and four shoot actions in each direction. The agent's task is to move to a cell such that he can fire an arrow to kill the wumpus. The dataset contains predicates for each cell such as `cellAt`, `cellRight`, and `cellAbove` and obstacle locations such as `wumpus` and `stench`. The wumpus is not observed in all the trajectories, although the stench is always observed. Two hundred trajectories were created by real human users



```
.___.___.___.___.___.
|S  |   |   |   |   |
.___.___.___.___.___.
|W  |S  |   |   |   |
.___.___.___.___.___.
|S  |   |A  |   |   |
.___.___.___.___.___.|
|   |   |   |   |   |
.___.___.___.___.___.
|   |   |   |   |   |
.___.___.___.___.___.
```

Figure 4.5: The text UI for the simple Wumpus World. *W* indicates the wumpus location, *S* indicates the stench location, and *A* is the agent.

Table 4.4: Results on the Wumpus dataset

Hidden %	20%		40%	
Algorithm	CLL	AUC-PR	CLL	AUC-PR
SEM-10	**-0.245**	**0.857**	**-0.261**	**0.853**
SEM-1	-0.278	0.845	-0.283	0.839
CWA	-0.282	0.826	-0.270	0.826

whose policy generally is to move towards the wumpus' row or column and shoot accordingly.

The EM approaches (using the trajectories where the wumpus is observed) learn that wumpus is surrounded by stench and fill the missing values in other trajectories. The *CWA* approach (Natarajan et al., 2011) on the other hand assumes that the wumpus is not present and relies on the stench to guess the action to be performed. The results are presented in Table 8. From the results, it can be easily observed that the EM methods are superior to that of the prior work on imitation learning on this testbed. Moreover, *SEM-10* which uses multiple samples outperforms the single-sample *SEM-1* approach. This domain clearly shows that the previous method of boosting in imitation learning is not sufficient in problems with partial observability and it is imperative to employ methods that do not assume closed-world.

4.4.4 Wrap up

In conclusion, our experiments have shown that opening the closed-world assumption definitely results in an improvement in post-learning performance. Between the two EM approaches, we have shown empirically that for certain domains (e.g. UW, IMDB) a single sample (hard-EM) might be sufficient, whereas in other domains (e.g. Cancer, Wumpus) multiple samples (soft-EM) are needed to capture the true distribution.

But these improvements come at the cost of increased learning time. SEM-10 can be comparatively much slower than SEM-1. SEM-10 can take from 15 hrs (Wumpus) to 22 minutes (UW-CSE) where SEM-1 takes 3 minutes on both of these datasets. The CWA approaches take 1 minute to learn the model on all of these datasets. Since the gradients are computed for every example and hidden state in SEM-10, the number of examples grows to $n \times 10$, where n is the number of examples in CWA.

Also the improvement in accuracy of RFGB-EM model over the RFGB model is relatively small. This is primarily due to RFGB's ability to handle the noise introduced by the CWA assumption. While RFGB-EM uses the model for the missing predicates to update the expected values, RFGB implicitly learns a similar sub-model within its model for the target predicates. For example, if a missing predicate p has a trivial model, $p(x) \Leftrightarrow q(x)$; RFGB with the closed-world assumption will check for $q(x)$ to remove the noise in $p(x)$.

4.5 Discussion and Future Work

I addressed the challenging problem of learning SRL models in the presence of hidden data. Towards this goal, we developed an EM-based algorithm for functional-gradient boosting. We derived the gradients for the M-step by maximizing the lower bound of the gradient and showed how to approximate the E-step. The proposed approach was employed for three different types of relational learning problems: RDNs, MLNs, and imitation learning in relational domains. The algorithms were evaluated on different domains for each learning problem. Our results indicate that the proposed algorithms outperform the respective algorithms that make closed-world assumptions.

To minimize the learning time of our approach, there are multiple future directions of research. Our current approach computes the probability of an example for each world state of the hidden groundings. But based on the relational tree structure, we can avoid recomputing the probabilities for examples, if a different world state will have no impact on the probability of the example. Developing inference techniques for tree-based models will also help to reduce the learning time of our EM approach. We can calculate the marginal probabilities of each hidden grounding and potentially use them as probabilistic facts to compute the gradients. There has been work (Jank, 2005) done in the MCEM approach on using samples from previous iteration, changing the number of samples in each iteration and deciding the number of iterations that could be effectively exploited by our method.

5 RELATIONAL ONE-CLASS CLASSIFICATION

Traditional classification approaches rely on labeled examples from all classes to learn a model. The approaches presented in previous chapters suffer from the same limitation. Even our approach to handle missing data uses the observed data to populate the missing examples. With only one class of examples, the EM approach will predict the missing examples to belong to the observed labels.

But in several real world problems, labeled examples are available for only one class. For example, in information extraction from text, a common annotation approach is to mark the true instances in the text (UzZaman et al., 2012; Kim et al., 2009). Since human annotation in these tasks can be quite costly, the negative examples are not annotated or in rare cases very few negative examples are annotated. For instance, while presidents of countries will be marked in an article for training, there is no explicit mention of who the negative examples of presidents are.

One possible approach to handle this problem is to assume that the unmarked instances are negative examples for the task at hand. Since the number of unmarked examples can be very large, this can increase the learning time for standard classification approaches. A bigger problem is that this can possibly lead to class imbalance where the number of instances of the negative class can dominate the positive instances. Finally, this simplistic assumption could be incorrect as all instances of positive examples may not be annotated (e.g., all mentions of presidents need not be annotated).

5.1 Introduction

Consequently, a lot of research has focused on developing one-class classification (OCC) methods (Khan and Madden, 2009) that directly learn using examples from only class or using very few examples of the other class. For example, Tax and Duin (1999) fit a hypersphere with the smallest radius around the labeled examples. All the examples inside this hypersphere are assumed to belong to the labeled class. Anomaly or Novelty detection tasks can also be viewed as an OCC task where many examples are available for the normal class and few or no examples of the anomalous class are provided. One-class classification can also be viewed as a specific case of semi-supervised learning (Zhu and Goldberg, 2009) with the labeled data being provided for only one class. Hence methods such as clustering and nearest-neighbors have also been used previously for OCC (de Ridder et al., 1998). These methods, in general, rely on a feature space representation (e.g., to fit a hypersphere) or distance measure (e.g., for clustering) to perform one-class classification.

While successful, these methods rely on a propositional representation of the data (i.e., as a flat feature vector). As shown earlier, data in the real world is inherently relational, i.e., they consist of inter-related objects and dependencies. Such structured data cannot be easily represented using a fixed-length feature vector and doing so can result in loss of possible information (Jensen and Neville, 2002). While first-order logic representation provides a loss-less way to model structured data, it introduces the additional complexity of having unbounded features associated with each example. E.g., consider the task of predicting the blood type of a person based on his lineage. We can potentially introduce a feature for the bloodtype of every ancestor of a person. As a result, standard distance metrics such as Euclidean norms cannot be naively used for relational problems. Thus, as we show empirically, standard OCC techniques for propositional data sets cannot be directly applied successfully to relational domains.

I present the first non-parametric approach for OCC using a tree-based distance measure between relational examples. We define this measure using the path similarity in relational trees. We combine the distances from multiple incrementally learned trees and labeled examples to calculate the probability of an example belonging to the labeled class. I present three interpretations of our approach: 1) based on combining rules for probability distributions (Natarajan et al., 2008), 2) based on using our distance with Kernel Density Estimation (Bishop, 2006) and 3) based on using a similarity kernel with Support Vector Data Description (SVDD). Using this model definition, we derive a scoring method that allows for learning the trees so as to maximize the likelihood of the labeled examples. Another interesting interpretation of our approach is to view the method as a *feature selection* strategy for standard OCC methods. From this perspective, our approach can be seen as a way to propositionalize relational data for OCC methods. Standard propositionalization can possibly result in irrelevant features but our method identifies the top features needed for propositional classifiers.

We make several key contributions:

1. We develop a new distance metric for relational examples that can be learned incrementally.

2. We propose a relational one-class classification approach using this distance metric.

3. We relate and place this work in the context of existing work on combination functions, density estimation and OCC.

4. We show how we can perform feature selection for propositional OCC methods by evaluating our trees.

5. Finally, we demonstrate over multiple domains that our approach can learn more accurate models than standard relational learning and OCC methods.

The work presented here has been published in the AAAI Conference on Artificial Intelligence (Khot et al., 2014b).

5.2 Propositional OCC

One-class classification (OCC) problem was first described by Moya and Hush (1996) for an automatic target recognition task using neural networks. Following their work, Schölkopf et al. (2000) learn a Support Vector Machine (SVM) to separate the labeled examples from the origin. Tax and Duin (1999), on the other hand, learn a hypersphere with the smallest radius that encompasses the labeled examples. They have also shown that using the Gaussian kernel with a normalized dataset gives solutions comparable to the SVM approach by Scholkopf et al. Comité et al. (1999), on the other hand, use modified C4.5 to perform one-class classification. All these approaches rely on a propositional representation.

Anomaly or outlier detection methods can also be viewed as a one-class classification approach. Although there have been many outlier detection methods proposed in literature (Chandola et al., 2009), outlier detection approaches generally assume that outliers are far and isolated (Liu et al., 2012). This is not true for the one-class classification problem where the labeled examples can be clustered and close to other unlabeled examples.

Semi-supervised learning approaches can also be used for OCC as shown by de Ridder et al. (1998). Our approach too can be viewed as a semi-supervised relational learning approach as it leverages unlabeled examples for learning the relational distance function.

One of the earliest works on relational one-class classification (Muggleton, 1997) learns the smallest first-order hypothesis that can cover the labeled examples. Since this work only performed Boolean predictions, it was extended to perform relational density estimation (Cussens, 1998). This approach relied on calculating proportions of examples captured by every hypotheses which can be prohibitively expensive for large domains.

Multiple propositionalization techniques and distance measures have been proposed for relational data which can potentially be used for OCC. One simple unsupervised approach to generate propositional features is by creating Boolean features corresponding to random first-order logic clauses (Anderson and Pfahringer, 2008). Relational Instance-based Learning (RIBL; Horváth et al. 2001) uses similarity between local neighborhoods to calculate the similarity between examples. Such unsupervised approaches may not capture long range information while we leverage the labeled examples to guide our approach. kFoil (Landwehr et al., 2010) uses a *dynamic* approach to learn clauses to propositionalize relational examples for SVMs. Each clause corresponds to one Boolean feature and is scored based on the improvement in the SVM learned from the propositionalized data. Rather than learning a new model for every candidate structure, we compute the error reduction on the fly locally.

5.3 Relational OCC

The high-level overview of our approach to Relational One-Class Classification (RelOCC) is shown in Figure 5.1. Our approach consists of the following steps:

1. Learn a first-order tree for calculating relational distances. (Section 5.3.1)

2. Use the distance function to perform OCC. (Section 5.3.2)

3. Learn the next distance tree to improve the classification. (Section 5.3.3)

4. Go to Step 2, if maximum allowed trees learned or ϵ percent of examples have less than δ change in distance.

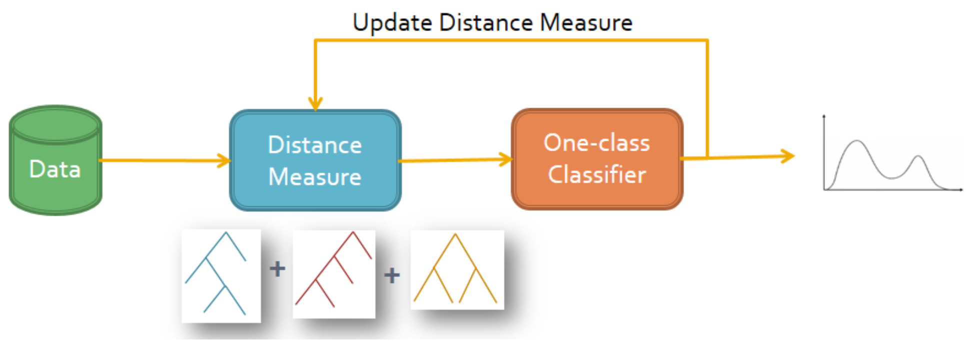


Figure 5.1: Relational one-class classification system design.

Term	Description
x, x_i, y	Relational examples
$\{x_1, x_2, \dots x_l\}$	Labeled examples
$\mathrm{lca}(x_1, x_2)$	Lowest Common Ancestor of the leaves reached by x_1 and x_2
$\mathrm{depth}(n)$	Depth of node n. $\mathrm{depth}(\mathrm{Root}) = 0$.
$d_i(x, y)$	Distance between x and y based on i^{th} tree
$D(x, y)$	Distance between x and y based on all trees
$P(y \in \mathcal{L})$	Probability of y being in labeled class
α_j	Weight of labeled example x_j
β_i	Weight of i^{th} tree
$I(\mathrm{condition})$	Indicator function that returns 1 if condition is true, 0 otherwise.
$P^t(y \notin \mathcal{L})$	$P(y \notin \mathcal{L})$ using t trees

Table 5.1: Description of terms used in this chapter.

Table 5.1 presents the terminology used throughout the chapter. I will now describe the individual steps of our approach.

5.3.1 Tree-based distance

Inspired by semantic similarity measures (D'Amato et al., 2008), we propose a distance measure based on the paths taken by examples in a relational tree. For instance, consider two examples that reach the same node, n in a decision tree before taking separate paths down the tree i.e., n is the lowest common ancestor (LCA) of these examples. If the node n is at depth of three, the two examples share *at least* three common attribute values. If n is at depth of one, they share at least one attribute value. Intuitively, the two examples in the first case are more likely to be similar than in second case. Thus, our distance measure is inversely proportional to the depth of LCA of two example paths.

$$d(x_1, x_2) = \begin{cases} 0 & lca(x_1, x_2) \text{ is a leaf} \\ e^{-\lambda \cdot depth(lca(x_1,x_2)} & \text{otherwise} \end{cases}$$

Examples that differ at the root node i.e. have no common attribute will have a distance of 1. A major feature of this distance definition is that it can be used on any off-the-shelf tree learner (such as TILDE (Blockeel and Raedt, 1998)). The key difference from a first-order decision tree is that, in our formulation, there are no values associated with the leaves since all we are interested are the paths that determine the distance between examples. The parameter λ (set to 0.5 in experiments) ensures that the distance value decreases gradually as the depth increases.

Properties It can be easily shown that this distance function satisfies the following distance metric properties:

- $d(x_1, x_2) \geqslant 0$ (non-negative)
- $d(x_1, x_2) = d(x_2, x_1)$ (commutative)
- x_1 and x_2 are identical $\Rightarrow d(x_1, x_2) = 0$[1] (equality)

[1]The reverse condition does not hold.

- $d(x_1, x_3) \leqslant d(x_1, x_2) + d(x_2, x_3)$. (triangle property)
  Proof: If $l_{ij} = \mathtt{depth}(\mathtt{lca}(x_i, x_j))$, $l_{13} \geqslant \min(l_{12}, l_{23})$. WLOG, $l_{12} = \min(l_{12}, l_{23})$.
  $l_{13} \geqslant l_{12} \Rightarrow e^{-l_{13}} \leqslant e^{-l_{12}} \Rightarrow e^{-l_{13}} \leqslant e^{-l_{12}} + e^{-l_{23}} \Rightarrow d(x_1, x_3) \leqslant d(x_1, x_2) + d(x_2, x_3)$■

Using just one tree may not suffice when dealing with potentially infinite dimensions in relational data. A simple next step would be to learn multiple relational trees. However this step introduces an additional complexity –the need for a way to combine these trees without increasing the complexity of the already complex problem. Our solution is inspired by density estimation techniques that I present next.

5.3.2 Density estimation model

I now define a model (shown in Figure 5.2) to combine distances from multiple trees and then distances from multiple examples to perform relational one-class classification. Recall that to calculate the probability estimate of an example y to belong to the labeled class, we use the distances between y and the labeled examples. We denote the distance between the current example y and a labeled example x from the i^{th} tree as $d_i(x, y)$. $D(x, y)$ combines the tree distances to return the overall distance between x and y. We finally compute the density estimate $P(y \in L)$ using the distances from all the labeled examples $\{x_1, x_2, \dots x_l\}$. The combination function that we use in both levels is weighted mean:

$$D(x_j, y) = \sum_i \beta_i d_i(x_j, y) \text{ s.t.} \sum_i \beta_i = 1, \beta_i \geqslant 0$$
$$P(y \notin \mathcal{L}) = \sum_j \alpha_j D(x_j, y) \text{ s.t.} \sum_j \alpha_j = 1, \alpha_j \geqslant 0$$

One of the key advantage of this function is that it allows for efficient learning of trees as I show in Section 5.3.3. Also, this approach is closely related

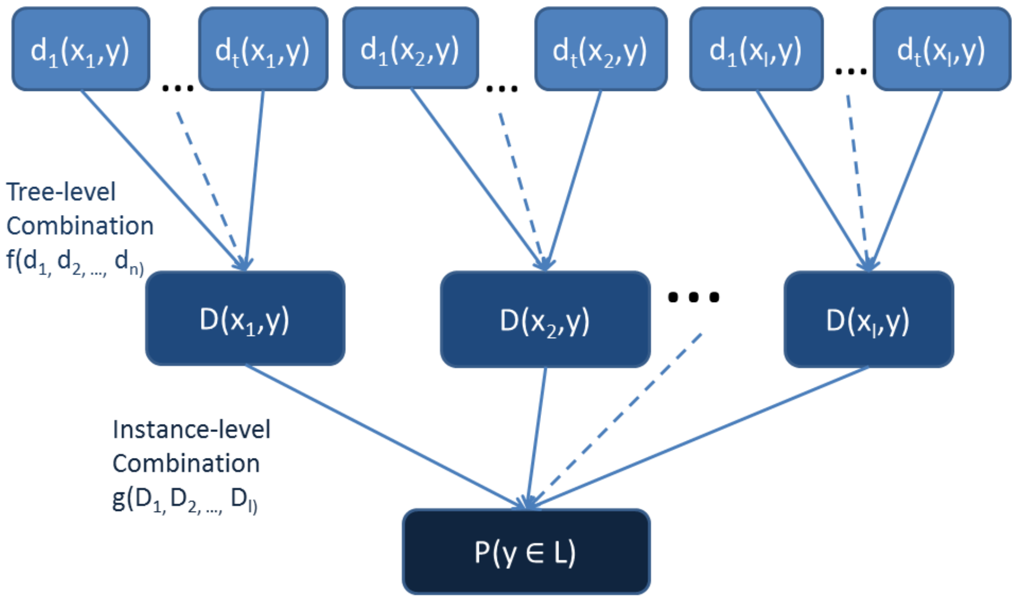


Figure 5.2: Density Estimation using Distance measure.

to existing research in one-class classification thus making it theoretically well grounded. I first present the learning of the trees followed by the relation to existing methods.

5.3.3 Model learning

Tree Learning As shown in Figure 5.1, we update the distance measure by iteratively adding to the set of trees. Assume that we have learned t trees already and are now learning the $(t+1)^{th}$ one.We can use any off-the-shelf tree learner and we employ the method described by TILDE tree learning (Blockeel and Raedt, 1998). Since our distance metric only relies on the LCA position, we can make local scoring decisions at every node. If a pair of examples are already split by the tree, any further splits at lower levels will have no impact on the distance between these examples i.e., adding a node to the tree only affects the distance between the pair of examples that are split at that node. Specifically, splitting example x_i and x_j at node n, increases their distance $d_{t+1}(x_i, x_j)$ from zero to $\Delta_{t+1}(x_i, x_j) = e^{-\mathtt{depth}(n)}$.

We use a modified splitting criterion which is the squared error over the examples i.e., $\sum_y [I(y \notin \mathcal{L}) - P(y \notin \mathcal{L})]^2$. $I(y \notin \mathcal{L})$ is the indicator function which returns 1 if y is an unlabeled example. Hereafter we use $I(y)$ to compactly represent $I(y \notin \mathcal{L})$. As can be seen here, our scoring function does rely on using the unlabeled examples in the dataset. Even if unlabeled examples are not provided, these can be generated by using constants of matching types (e.g. $\{\mathtt{friends}(x, y) | \forall x, y \in \mathtt{People}, \mathtt{friends}(x, y) \notin \mathcal{L}\}$). Relying on only the labeled examples for learning a distance function will result in trivial distances of $d(x, y) = 0$.

Consider $\mathbf{x}$ to be the set of examples that reach node n. $\mathbf{x_l}$ and $\mathbf{x_r}$ are the set of examples that take the left and right branch respectively. Since introducing the node has no impact on the distances (and in turn probabilities) for examples not reaching node n, the squared error is

$$\begin{aligned}
&\sum_{y \in \mathbf{x}} [I(y \notin \mathcal{L}) - P(y \notin \mathcal{L})]^2 \\
&= \sum_{y} [I(y) - \Sigma_j \alpha_j \{\Sigma_i^t \beta_i d_i(x_j, y) + \beta_{t+1} \Delta_{t+1}(x_j, y)\}]^2 \\
&= \sum_{y} \left[I(y) - P^t(y \notin \mathcal{L}) - \Sigma_j \alpha_j \beta_{t+1} \Delta_{t+1}(x_j, y)\right]^2
\end{aligned}$$

At every node in the tree, we minimize this squared error. If y is close to the labeled examples ($P^t(y \notin \mathcal{L}) \approx 0$) for an unlabeled example ($I(y) = 1$), increase in the distances to the labeled examples will reduce the squared error. If y is already spread apart from the labeled examples (i.e. $P^t(y \notin \mathcal{L}) \approx 1$), the distances to the labeled examples will not be increased any further. As a result, minimizing the squared error *introduces new features* to spread out the examples and *prevents redundant features* being added.

Since introducing node n only increases the distance between the pair of examples split at the node, $\Delta_{t+1}(x_j, y) \neq 0$ for $x_j \in \mathbf{x_l}, y \in \mathbf{x_r}$ or $x_j \in$

$\mathbf{x_r}, y \in \mathbf{x_l}$. Hence the splitting criterion can be modified to

$$\min \sum_{y \in \mathbf{x_r}} \left[I(y) - P^t(y \notin \mathcal{L}) - \Sigma_{j:x_j \in \mathbf{x_l}} \alpha_j \beta_{t+1} \Delta_{t+1}(x_j, y) \right]^2$$
$$+ \sum_{y \in \mathbf{x_l}} \left[I(y) - P^t(y \notin \mathcal{L}) - \Sigma_{j:x_j \in \mathbf{x_r}} \alpha_j \beta_{t+1} \Delta_{t+1}(x_j, y) \right]^2$$

We use a greedy search procedure for learning the tree where we pick the feature whose split reduces this squared error. The key insight is that the use of the mean squared error along with our relational path based distance function allows us to efficiently make these local scoring decisions. Note that our approach is non-parametric as the only parameter is the number of trees that increases as more data is obtained. Thus we can potentially update the distances in online fashion.

Weight Learning After learning each tree, we update the weights α and β for weighted mean function. We use a co-ordinate gradient descent approach where we iteratively update α and β to minimize the squared error. Since there is no closed form solution, we update the weights only after learning an entire tree instead of updating after learning every node in the tree. The gradient of error w.r.t. α is

$$\frac{\partial}{\partial \alpha_j} \sum_y \left[I(y) - \Sigma_j \alpha_j \Sigma_i \beta_i d_i(x_j, y) \right]^2$$
$$= -2 \sum_y \left[I(y) - P(y \notin \mathcal{L}) \right] \Sigma_i \beta_i d_i(x_j, y)$$

and the gradient w.r.t. β is

$$\frac{\partial}{\partial \beta_i} \sum_y \left[I(y) - \Sigma_j \alpha_j \Sigma_i \beta_i d_i(x_j, y) \right]^2$$
$$= -2 \sum_y \left[I(y) - P(y \notin \mathcal{L}) \right] \Sigma_j \alpha_j d_i(x_j, y)$$

To ensure that weights are always greater than 0 and sum to 1, we project them as described by Natarajan et al. (Natarajan et al., 2008).

5.3.4 Model interpretations

I now place our work in context to existing work on learning parameters in SRL and propositional OCC methods.

Combining Rules: Since $d_i(x, y) \leqslant 0$, it can be interpreted as the probability of example x being not equal to y based on the i^{th} tree, i.e., $P_i(x \neq y) = d_i(x, y)$. Similarly $D(x, y)$ can be viewed as overall probability of example x being not equal to y, i.e., $P(x \neq y) = D(x, y)$. This can be viewed as using two-level combining rules (Natarajan et al., 2008) in directed SRL models with the weighted mean combination function. At first level, we combine probability estimates from the set of trees. At second level, we combine probabilities based on individual examples.

Kernel Density Estimation: An alternate interpretation of our model is that of using Kernel Density Estimation (Bishop, 2006) with a triangular kernel[2] and bandwidth 1.

$$
\begin{aligned}
1 - \sum_j^n \alpha_j \sum_i \beta_i d_i(x_j, y) &= \sum_j^n \frac{1}{n}(1 - \sum_i \alpha_j \beta_i d_i(x_j, y)) \\
= \sum_j^n \frac{1}{n}|1 - D'(x_j, y)| &= \sum_j^n \frac{1}{nh} K(\frac{D'(x_j, y)}{h}) \text{ for h=1}
\end{aligned}
$$

where $D'(x_j, y) = \sum_i \alpha_j \beta_i d_i(x_j, y)$. Since the term $|x_j - y|$ cannot defined easily for relational examples, we use D' function to represent the absolute distance between two examples needed by the kernel density estimator. D' function value is always less than 1 and as a result is within the bounds of the triangular kernel.

[2] $K(x, y) = (1 - |x - y|) \cdot I(|x - y| < 1)$

SVDD: SVDD (Tax and Duin, 1999) approach learns a hypersphere around the labeled one-class examples. An example z is classified as belonging to the labeled class if $(z \cdot z) - 2\sum_i \alpha_i (z \cdot x_i) + \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j) \leqslant R^2$. Since the classification only depends on the dot product, SVDD also uses kernel functions. Using the function $1 - D$ in our case as the kernel functions[3] changes the function used for classification to $1 - D(z, z) - 2\sum_i \alpha_i (1 - D(z, x_i)) + \sum_{i,j} \alpha_i \alpha_j (1 - D(x_i, x_j)) \leqslant R^2$. Since $D(z, z) = 0$, we can rewrite the decision boundary as $1 - \sum_i \alpha_i D(z, x_i) \geqslant 1 - \sum_i \alpha_i D(x_k, x_i)$[4] where x_k is any labeled example. We use the left hand side of the equation as the probability estimate for z belonging to the labeled class. Our definition allows to return probabilistic estimates, but we can now also use $1 - \sum_i \alpha_i D(x_k, x_i)$ as a decision boundary to predict the example class. Since we optimize a different function, it is not necessary for all examples x_k to result in the same boundary. But we can approximate it by using the average distance to all labeled examples.

Feature Selection for OCC: It is also possible to view our proposed approach as a feature selection strategy for standard one-class classification methods. More specifically, we can convert every path from root to leaf into a Boolean feature since these paths can be represented as a horn clause. Since the resulting feature is Boolean, doing this for every path will yield a propositionalized data set that can then employ standard OCC methods such as SVM-OCC. We call this approach SVMOCC-Rel and show the importance of this feature selection strategy in our experiments. Propositionalization of relational data can possibly result in infinite features as a walk through the relational graph can lead to several resulting features. For instance, when reasoning about a particular genetic disposition, it is possible to use generations of ancestors' information. Our method on the other hand, can be seen as selecting these relational features (i.e., identifying the most important set of ancestors) for a propositional classifier

[3]this is possible as kernel functions are similarity functions

[4]after expanding R^2

making it possible for the propositional methods to work with relational data more seamlessly.

Implementation details I now present some heuristics and parameters used by our approach. Our approach begins with the distance of 0 between every pair of examples. During every tree learning step, we use only a subsample of examples for efficiency which has shown to improve performance in highly skewed datasets (Chan and Stolfo, 1998). We further subsample the examples based on their current predictions. Based on active learning (Settles, 2012) techniques, we pick unlabeled examples based on their proximity to the decision boundary as well as the misclassified examples. For the weight learning step, we use a step length of $\eta = 0.001$ for all our experiments.

5.4 Experiments

We evaluate the following algorithms:

- RelOCC: This approach is our relational OCC method using the predictions from the combination function.
- RND: We modify the tree learner to randomly pick features at every node. To ensure a strong baseline, we do not pick any feature that results in more than 95% examples going to one branch. We still use the combination functions to obtain the final probabilistic prediction.
- O-Rel: This refers to the strategy described above for using SVM-OCC with features generated from RelOCC.
- O-Rnd: This refers to SVM-OCC with features from RND trees (i.e, random features).

- SVM: Instead of using only labeled examples, we use all the examples for training a standard SVM on the propositionalized data. We use *S-Rel* and *S-Rnd* to denote the SVM learned from RelOCC and RND trees respectively.
- RPT: As we are learning conditional discriminative models, we learn relational probability trees (Neville et al., 2003a) for the target class as a relational baseline.

Due to the skew in relational datasets, we use a subset of examples with a reduced skew (twice the negatives to positives) for testing. The same subset is used for all approaches within each domain. Also since one-class SVM methods return only binary predictions we compare the accuracies for these approaches. But, the relational approaches return a probabilistic prediction and as a result we can compare the AUC-PR values for them.

5.4.1 UW-CSE

In the UW-CSE dataset, the goal is to predict the `advisedBy` relationship between a student and a professor. Since the primary motivation of our approach is one-class classification where all the positive examples are not marked, we use only a subset of positive examples and assume they were the only ones marked by annotators. We pick 20%, 40% and 60% of the labeled examples for training and use all the positives for testing. The results for five-fold cross-validation are shown in Table 5.2 and Table 5.3.

% marked	RelOCC	RND	RPT
20%	**0.93**	0.87	0.76
40%	**0.93**	0.81	0.81
60%	**0.92**	0.83	0.88

Table 5.2: AUC-PR on UW dataset.

% marked	RelOCC	O-Rel	O-Rnd	S-Rel	S-Rnd
20%	**89.75**	77.39	75.24	60.56	60.58
40%	**89.97**	75.55	74.94	60.56	60.56
60%	**89.60**	76.02	74.43	60.56	60.56

Table 5.3: Accuracy on UW dataset.

As can be seen from the AUC-PR values, our approach outperforms both OCC with random feature selection and discriminative learning using RPTs thus answering Q1 affirmatively. Our approach also has a higher accuracy than both using SVM-OCC and standard SVMs on the propositionalized dataset. The training dataset is highly skewed and as a result basic SVM approaches (S-Rel and S-Rnd) predict the most common label. Thus Q2 can also be answered affirmatively in that using our method for feature selection helps SVM over the other feature selection methods. For all other datasets, S-Rel and S-Rnd show similar behavior where they predict all examples belonging to the negative class and hence we do not show them in other data sets.

5.4.2 IMDB

We focused on the task of predicting the `workedUnder` relation in this dataset. We performed five-fold cross-validation and the results are presented in Table 5.4 and 5.5. The results are similar to UW-CSE where RelOCC outperforms the SVM-based approaches. But for this dataset, RPT are also able to learn with few positive examples and have similar performance to RelOCC. Both these approaches still outperform RND.

5.4.3 NFL

One of the primary motivations of our work was to learn models for information extraction data sets with only positive labels. To this end, we evaluate our approach on the task of extracting winners of National

% marked	RelOCC	RND	RPT
20%	**0.98**	0.80	**0.99**
40%	**0.98**	0.88	**0.97**
60%	**0.97**	0.91	**0.97**

Table 5.4: AUC-PR on IMDB dataset.

% marked	RelOCC	RND	O-Rel	O-Rnd
20%	**93.88**	56.76	81.15	84.10
40%	**94.68**	60.49	81.27	78.90
60%	**94.36**	62.14	78.69	77.53

Table 5.5: Accuracy on IMDB dataset.

Football League (NFL) games from sports articles. The annotations only provided few positive examples. We use the Stanford NLP toolkit [5] to convert the text into NLP structures such as parse trees and dependency graphs.Since this dataset only has few annotated postives, we do not drop any labeled examples in our experiments. The results for ten-fold cross-validation on this task are shown in Table 5.6 and 5.7.

RelOCC	RND	RPT
0.63	0.35	0.59

Table 5.6: AUC-PR on NFL dataset.

RelOCC	RND	O-Rel	O-Rnd
69.95	34.92	**70.22**	47.70

Table 5.7: Accuracy results on NFL dataset.

As can be seen in these tables, RelOCC outperforms the other relational approaches. But as compared to the SVM-based approaches, using the RelOCC features for SVM-OCC (O-Rel) does as well as the relational OCC approach in terms of the accuracy. This shows clearly that our proposed approach serves as a good feature selection method as well. Since this

[5]http://nlp.stanford.edu/software/corenlp.shtml

dataset has a large number of possible features, random feature selection is not able to pick the relevant attributes and both RND and O-Rnd are much worse.

5.4.4 Heart

To evaluate the quality of feature selection, we use the Heart dataset [6], which is a multivariate data set with 13 attributes. The task is to predict the presence of heart disease in patients. We performed four-fold cross-validation and show the results in Tables 5.8 and 5.9.

% marked	RelOCC	RND	RPT
40%	**0.44**	0.37	0.36
60%	**0.43**	0.38	**0.43**

Table 5.8: AUC-PR on Heart dataset.

% marked	RelOCC	RND	O-Rel	O-Rnd
40%	50.34	37.67	**62.59**	49.50
60%	46.38	39.31	**60.55**	51.53

Table 5.9: Accuracy on Heart dataset.

While RelOCC is better than both RND and RPT, using SVM-OCC with RelOCC as a feature selection procedure does much better than using the RelOCC model. Since this dataset is originally a propositional dataset, the complex interactions captured by the distance function in RelOCC are not required and can possibly overfit the data. The results here show that SVM benefits from the feature selection, as it is better than both O-Rnd and the baseline SVM methods.

[6]http://archive.ics.uci.edu/ml/datasets/Heart+Disease

5.5 Discussion and Future Work

I presented a non-parametric approach for one-class classification from relational data. We defined a new distance metric based of first-order decision forest and a density estimation model using the distance metric. We can efficiently update the distance metric to improve the classifier's performance. Our approach as a side effect introduces novel and relevant features, which can also be used to augment propositional OCC methods.

The next step is to apply our approach to other information extraction tasks such as TempEval (UzZaman et al., 2012) and BioNLP (Kim et al., 2009). Integrating other kernels and combination functions into our method might help in further improving the performance. Given prior knowledge about the proportion of positively labeled examples, our approach can be modified to leverage that information.

6 BRIDGING THE GAP BETWEEN DIRECTED AND UNDIRECTED MODELS

Traditionally directed graphical models such as Bayesian networks (and more recently their relational counterparts, e.g., Bayesian Logic Programs) have been used to encode causal domain knowledge, since they can intuitively capture the cause and effect in a domain. On the other hand, MLNs are undirected relational models that have recently gained popularity due to their ease of representation. MLNs allow domain experts to specify the structure using intuitive first-order logic rules without having to deal with the acyclicity constraints of the directed models.

One may use a directed model in a domain because they have been used traditionally to represent knowledge in that domain. Also directed models can be used to represent causal relations and temporal relations intuitively. On the other hand, undirected models, such as MLNs, can represent correlations and spatial relations better since they do not have any acyclicity constraints.

Converting the domain knowledge representing using directed models into MLN clauses would allow one to combine a directed model with other MLN clauses into a single MLN model without being constrained by the acyclicity condition. Converting the conditional distributions in directed models to MLNs can be easily performed by introducing one rule for every parameter in the model[1]. But directed models also use combining rules (explained below) to combine distributions from independent causes.

I present an approach to translate *decomposable* combining rules into MLN clauses. The important aspect of this representation is that we do not use the “ground" Bayesian network and instead use a “lifted" representation. Using a lifted representation avoids grounding clauses

[1] http://alchemy.cs.washington.edu/faq/index.html

containing logical variables to an exponentially large number of variable-free clauses.

In the next subsection, I will present the idea of "Independence of Causal Influence" (ICI) and decomposable combining rules. In Section 6.2, I will present our work on converting combining rules to MLN clauses. Finally, I will present experimental results that demonstrate the usefulness of combining rules in learning a more accurate model with small number of clauses. The work presented here has been previously published in the *European Conference in Machine Learning* (Natarajan et al., 2010).

6.1 Causal Independence in Directed Models

Directed models learn conditional probability distributions (CPDs) for each variable with one parameter for each combination of its parents. The notion of "Independence of Causal Influence" (ICI; Heckerman and Breese 1994; Zhang and Poole 1996), also known as "causal independence", assumes that there may be multiple *independent* causes for a target variable. For example, assume we have three causes A, B and C for a target variable D. By assuming that these causes independently influence the target variable, directed models can learn the CPDs due to each cause separately ($P(D \mid A)$, $P(D \mid B)$ and $P(D \mid C)$) and combine them using a (possibly stochastic) function, reducing the number of parameters from eight to six.

Given independent CPDs for each cause, we can combine these distributions using functions such as OR, noisy-OR, mean and weighted mean (Heckerman and Breese, 1994). For example, the mean combining rule applied on the three distributions mentioned above would give us $P(D \mid A, B, C) = \frac{1}{3}(P(D \mid A) + P(D \mid B) + P(D \mid C))$. One can also use multiple levels of combining rules. I present details regarding *decomposable* combining rules with multiple levels below and later show how one can convert them into MLN clauses.

6.1.1 Decomposable combining functions

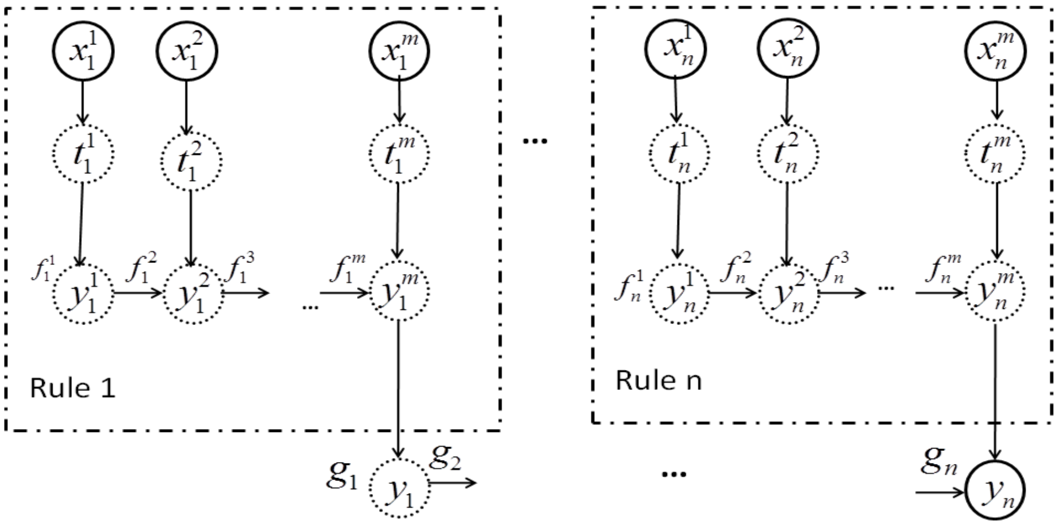


Figure 6.1: Decomposable combining rules.

Consider Figure 6.1 where y is the target and x's are the influents (causes/parents). The t_i^j's are temporary y-values due to each instantiation of x_i^j based on Rule i. The y's are deterministic nodes obtained by using function f's for the first level (that combine instances of the same rule) and g's for the second level (that combines different rules). The observed nodes are shown using solid circles while the dotted circles correspond to the hidden nodes. These hidden nodes are created when the functions are applied successively.

In the figure, I present two levels of combining rules (f_i and g_j). The first level of the combining rule f_i combines the causes $\{x_i^1, \cdots, x_i^m\}$ while the second level g_j combines the outputs from the rules $\{y_1, \cdots, y_n\}$. Let us consider the first-level combining rule. For simplicity, consider the set of functions f_1^i from rule 1. The result y_1 can be represented as:

$$y_1 = f_{1,\sigma}^m(t_{1,\sigma}^m, f_{1,\sigma}^{m-1}(t_{1,\sigma}^{m-1}, ...f_{1,\sigma}^1(t_{1,\sigma}^1, p))) \tag{6.1}$$

for the given ordering σ of the different x_1^i; p is some prior on the value of y for rule j. Equation 6.1 can be written as

$$y_1 = f_{1,\sigma}(t_{1,\sigma}^m, t_{1,\sigma}^{m-1}, ...t_{1,\sigma}^1, p) \tag{6.2}$$

where f_1 is a combining rule operating over all t_1^i.

Definition: A combining rule is called *decomposable* if it satisfies Equation 6.2 for **all** orderings (i.e., $\forall\sigma$). This is to say that for every possible ordering of the inputs, the combining rule can be decomposed into a set of functions that yield the same distribution.

In our setting, we require that the combining rules at both levels are *decomposable*. In other words, any permutation of the x_i^j's within a rule should produce the same value for y_i and any permutation of the y_i's should produce the same final y value. Note that most common combination functions used in the literature (Koller and Pfeffer, 1997a; Natarajan et al., 2008; Jaeger, 2007; Heckerman and Breese, 1994; Zhang and Poole, 1996) such as: *Noisy-Or*, *Noisy-And*, average-based combination functions such as *mean*, *weighted-mean* and *context specific independence*(CSI) are decomposable. In this chapter, I show how multi-level combining rules (rules that combine instances of the same clause and the ones that combine the distributions due to different clauses) can be represented and learned using MLNs. Rather than using complex figures such as Figure 6.1 to describe combining rules in directed models, we use an abstract syntax which I present next.

6.1.2 Combining rules in directed models

We use an abstract syntax called as First-Order Conditional Influence (FOCI) statements (Natarajan and Altendorf, 2005) to present the semantics of combining rules in directed models. Natarajan and Altendorf (2005) showed how most directed models such as BLPs (Kersting and De Raedt,

2007) and PRMs (Getoor et al., 2001) can be represented using this syntax. By converting FOCI statements to MLNs, we show that the knowledge captured by directed models can be represented using MLNs.

Each FOCI statement has the form:

If $\langle \mathit{condition} \rangle$
then $\langle \mathit{qualitative\ influence} \rangle$

where the $\mathit{condition}$ is a conjunction of literals. A $\langle \mathit{qualitative\ influence} \rangle$ is of the form $X_1, \ldots, X_k$ *Qinf* Y, where the X_i and Y are of the form $V.a$, and V is a variable that occurs in $\mathit{condition}$ and a is an object attribute. Associated with each FOCI statement is a *conditional probability distribution* 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. Combining rules can be represented as $X_1, \ldots, X_k$ *Qinf* (CR) Y where CR is the name of the specific combining rule used. An example of a two-level combining rule for predicting the satisfaction of a student is shown below.

```
mean {
If {student(S), course(C), takes(T,S,C)} then
    T.grade Qinf (noisy-OR) S.satisfaction.
If {student(S),paper(P,S)} then
    P.quality Qinf (noisy-OR) S.satisfaction.
}
```

The first rule specifies that the grade that a student obtains in a course influences his/her satisfaction. The CPD $P(S.\text{satisfaction} \mid T.\text{grade})$ associated with the first statement (partially) captures the quantitative relationships between the attributes. The second states that if the student has authored a paper, then its quality influences the satisfaction of the student. The distributions due to multiple instantiations of the respective rules (the different course grades or the different paper qualities) are combined using the noisy-OR combining rule and the distributions due to different rules are combined using mean combining rule. Similarly Figure 6.1 can be represented using FOCI statements as:

```
g {
    If {true} then
      x_1 Qinf (f_1) y.
        ...
    If {true} then
      x_n Qinf (f_n) y.
  }
```

Note that there are two levels of combination functions - one for combining multiple instances of the same rule and the other for combining different rules. This idea of two-level combining rules is sufficient to capture the notion of ICI in SRL models and hence I address the two-level combining rules in this chapter.

The use of combining rules make learning in directed SRL models easier: multiple instances of the same rule share the same CPT and hence each instance (x_1^i for rule 1 in Figure 6.1) can be treated as an individual example while learning the CPTs. Similarly, the different CPTs can be learned *independently* of each other, thus exploiting the notion of causal independence. Yet another advantage of the combining rules is that they allow for richer combination of probability distributions. MLNs in their default representation use an exponentiated weighted count as an (indirect) combination function of the different clauses. To express combining rules in MLNs, a straightforward method would be to construct the grounded BN for each rule and then construct the equivalent Markov Network. Unfortunately, this leads to an exponential number of clauses in the MLN, making the twin problems of learning and inference computationally expensive. Instead we resort to a "lifted" method that avoids unrolling (grounding) all the clauses to create the MLN.

6.2 Decomposable Combining Rules in MLNs

The key idea is to view combination functions as choosing a value among several values proposed by the parents. For instance, taking the average of distributions corresponds to choosing the target value using an uniform distribution among the values proposed by the parents. Weighted mean, on the other hand, can be understood as choosing a value based on the distribution given by the weights. Given this intuition, we create *hidden* predicates that propose values based on the conditional distribution and *multiplexer* predicates that select one value from the proposed values.

Consider the following two FOCI statements: $a(X, Y)$ *Qinf* $b(Y)$ and $c(Z, Y)$ *Qinf* $b(Y)$. Associated with each clause is a conditional probability distribution $P(b(Y)|\text{parent(b)})$ where the parent(b) for the first statement is $a(X, Y)$ and the second is $c(Z, Y)$. Note that there could be several possible instantiations for X and Z in the above rules. For simplicity, let us assume that the distributions due to the different instances of the same rule are combined using some CR_1 and the resulting distributions due to the different rules are combined using some CR_2.

Consider the BN presented in Figure 6.2. For ease of explanation, assume that there are n instantiations of each rule and k such rules (I present only two of them for brevity). In addition to the a, b and c predicates, we introduce two more types of predicates indicated using dashed nodes: *hidden* (temporary) value predicates (t and tr) and *multiplexer* predicates (h and hr). Since there are two levels of combining functions, there are two different sets of multiplexers and hidden nodes represented by two different boxes in the figure. The first box corresponds to choosing a value from a single rule (given by $r(y, i)$, where i is the rule index) and in the next level the final value of the target is chosen from one among the different r values. We now explain the multiplexers inside the same rule (the top box) and the same idea is extended for different rules (bottom box).

The hidden predicates t's can be understood as choosing a value of

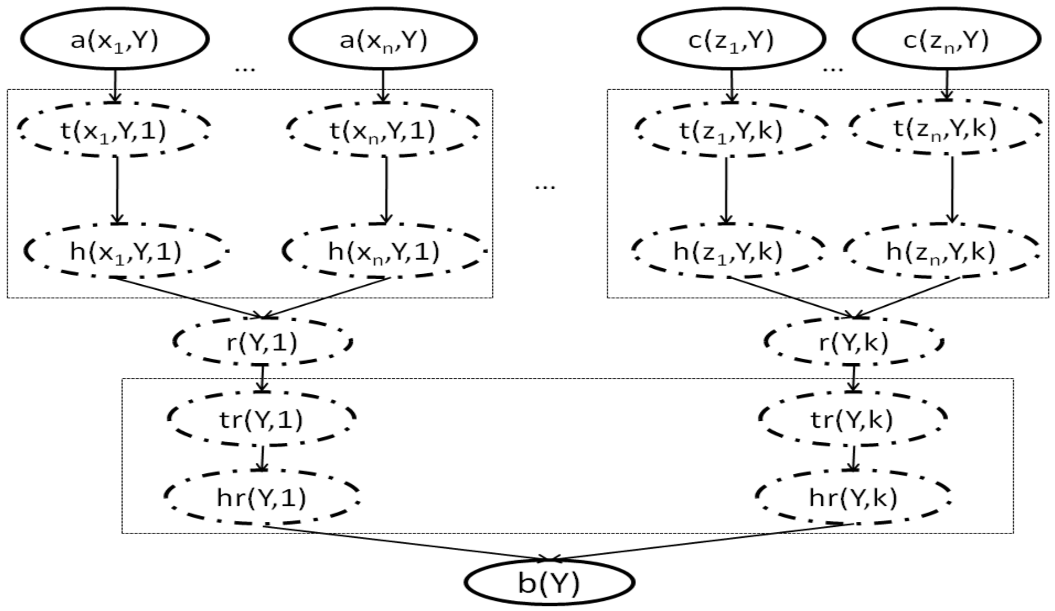


Figure 6.2: Understanding combining rules using multiplexers. The dashed nodes are the hidden nodes and the multiplexer nodes, while the solid nodes are observed in the data.

the target given the instantiation of the parent based on the CPD. The multiplexers (h-nodes) serve to choose one of the n t-values for the target. The idea is that if a particular h is activated, the value of the corresponding t node is chosen to be the value of the target for the current rule (i.e., $r(y, i)$ is set to be that particular t-value). Given the different values of $r(y, I)$ for all I, the final value of the target b is chosen using the next level of the multiplexer. In this work, we derive a general representation that covers all decomposable combining functions. Our translation consists for four different kinds of clauses:

1. **CPT Clauses:** This follows the standard translation of Bayesian networks to MLNs (Domingos and Lowd, 2009). Each independent parameter in the CPT of the Bayes net becomes a clause in the MLN.

An example of such a clause is

$$\begin{aligned} w_1^1 &: \mathrm{a}(X,Y) \Rightarrow \mathrm{t}(X,Y,1) \\ w_1^0 &: \neg\mathrm{a}(X,Y) \Rightarrow \mathrm{t}(X,Y,1) \end{aligned} \tag{6.3}$$

where $w_1^j = \log\frac{p_1^j}{(1-p_1^j)}$, $p_1^j = P(b(Y) = 1|a(X,Y) = j)$[2]. Hence, for each independent parameter of the original CPT in the directed model, there is a clause in the MLN with the weight as a function of the parameter. We use the number 1 in the third argument of t and w_1^j to indicate the rule index which changes for every rule. In general, the set of arguments in the temporary predicate t is the union of all the arguments in the body of the clause and an argument for the rule index.

2. **Multiplexer Clauses:** These are the clauses that choose a particular value of the target given a set of parent values. For the first-level multiplexer (h in the Figure 6.2), this corresponds to the set of values due to different instantiations of the same rule. For the second-level multiplexer, this set consists of the values due to different rules. For the first level, the MLN clauses are of the form

$$\infty : h(X,Y,I) \Rightarrow (t(X,Y,I) \Leftrightarrow r(Y,I)) \tag{6.4}$$

The above clause is a hard clause (i.e., infinite weight) that specifies that for a particular value of X, if $h(X,Y,i)$ is true for a rule i, then the value of the target for that rule ($r(Y,i)$) must be chosen to be the corresponding $t(X,Y,i)$. Note that the multiplexer always has the same number of variables as that of t. Similarly, for the next level,

[2]The CPT clauses are defined for rule 1 that uses predicate a. All the other rules will have similar clauses.

the multiplexer clause would be

$$\infty : \mathtt{hr}(Y, I) \Rightarrow (\mathtt{tr}(Y, I) \Leftrightarrow \mathtt{b}(Y)) \tag{6.5}$$

3. **Stochastic Function Clauses:** These are the clauses that specify the stochastic function to be employed on the values. These are essentially the "prior" on the $\mathtt{h}$ predicates. For mean, the idea is to choose a target value from the set of $\mathtt{h}$-values uniformly. In the case of Noisy-Or, the target is chosen from using an Or function over the hidden variables ($\mathtt{t}$ and $\mathtt{tr}$). I present the stochastic function clauses for two different cases later in the section.

4. **Integrity Constraints:** These are the constraints that are used to specify that among the different multiplexer nodes, only **one** of them can be true for any particular example. These are of the form:

$$\begin{aligned} \infty : \quad & \mathtt{h}(X_1, Y, I) \wedge \mathtt{h}(X_2, Y, I) \Rightarrow (X_1 = X_2) \\ \infty : \quad & \exists X \; \mathtt{h}(X, Y, I) \end{aligned} \tag{6.6}$$

   The above set of clauses specifies that if $\mathtt{h}$ is true for two values of X, they should be identical and there exists a grounding of X to make $\mathtt{h}$-true. These constraints are identical for the second level as well[3].

In our formalism, there is no restriction on the equality of the combining rules at both the levels as long as they are decomposable. For instance, it is possible to use a mean combining rule to combine the instances of a single rule while a Noisy-Or could be used to combine the different rules themselves. It can be easily observed from our translation to MLNs that the only change for the different cases would be the encoding of the

[3]Alchemy supports constraints of this form using the syntactic sugar "!". However, we ran into issues when learning weights with "!" and hence explicitly provide the constraints.

multiplexers. Note that it is possible to imagine developing specialized clauses for each combination of the combining functions.

6.2.1 Transformation of combining rules

As mentioned earlier, the Bayesian network representation is used **only** to explain the translation by the use of multiplexers. The translation itself is independent of the number of groundings (note that all the predicates in the clauses are variablized and not grounded). We now present two most common types of combination functions from the literature: (1) *average*-based and (2) *noisy* combination functions. Let us consider just a single clause $a(X, Y) \Rightarrow b(Y)$ for ease of explanation. Associated with this clause is a conditional probability distribution $P(b|a)$ (we use a and b as shorthand notations for the predicates). As we mentioned, the differences between different combination functions lie mainly in the stochastic function clauses. For each case, we first present the translation and prove the correctness of the resulting distribution. We then present the worked example corresponding to the student satisfaction rules presented earlier.

Average-Based Combining Rules: Assume that the different instantiations of the above rule are combined using the weighted-mean combining rule. Then the posterior over the target b given the different sets of parents is given by

$$P(b|a_1, ..., a_n) = \frac{1}{\sum w_i} \sum_i w_i \times P(b|a_i) \tag{6.7}$$

where a_i denotes $a(x_i, y)$. For the case of mean, all $w_i = 1$ (note that the w's are not the weights of the MLN clauses, they are the weights of the combining rule). The CPT clauses will be of the form presented in the earlier section, where the weights are log functions of the CPT parameters $(\log \frac{p_i}{(1-p_i)})$. The multiplexer clause is again a hard clause that specifies the value of the target based on the value of the multiplexer $(h(X, Y, I))$.

The integrity constraints are also the same as the ones presented above. The stochastic function is the weighted-mean. This specifies the prior on the multiplexer nodes, i.e., defines the prior probability with which each multiplexer node is true. Hence, they are of the form: $u_i : h(x_i, y, i)$, where $u_i = \log(w_i)$ is the log-odds of the given x_i. Actually, any weight of the form $u_i = \log(const \times w_i) = \log(const) + \log(w_i)$ would work. For mean, the log-odds imply $u_i = \log(1/n)$, where n is the number of instantiations. From the previous equation, it follows that any u_i are acceptable as long as they are constant for all i. The intuition is that each $t(X, Y)$ chooses the value of the target based on the CPT, and the final value of the target is chosen from the different t's using the multiplexer nodes. The multiplexer is activated such that it takes only one value given by the stochastic function (mean or weighted-mean).
Proposition: The given representation of MLNs exactly captures the distribution given by Equation 6.7.
Proof: For simplicity, consider only two instantiations of the rule presented above. We are interested in $P(b|a_1, a_2)$, which is given by Equation 6.7 for $i = 2$. There will correspondingly be four different cases: both t_1 and t_2 (hidden variables) are true and one of h_1 or h_2 (multiplexers) is true (two cases) and two cases where only the multiplexer h_i and the corresponding t_i are true. i.e.,

$$\begin{aligned}
&P(b|a_1, a_2) = \\
&\quad P(b, t_1 = 1, t_2 = 1, h_1 = 1, h_2 = 0|a_1, a_2)+ \\
&\quad P(b, t_1 = 1, t_2 = 1, h_1 = 0, h_2 = 1|a_1, a_2) + \\
&\quad P(b, t_1 = 1, t_2 = 0, h_1 = 1, h_2 = 0|a_1, a_2)+ \\
&\quad P(b, t_1 = 0, t_2 = 1, h_1 = 0, h_2 = 1|a_1, a_2)
\end{aligned}$$

$$
\begin{aligned}
=&\frac{1}{Z}\left(e^{\theta_1+\theta_2+\log(w_1)}+e^{\theta_1+\theta_2+\log(w_2)}+e^{\theta_1+\log(w_1)}+e^{\theta_2+\log(w_2)}\right),\\
&\text{where }\ \theta_i=\log\frac{p_i}{1-p_i}\\
=&\frac{1}{Z}\left(\frac{p_1}{1-p_1}\frac{p_2}{1-p_2}w_1+\frac{p_1}{1-p_1}\frac{p_2}{1-p_2}w_2+\frac{p_1}{1-p_1}w_1+\frac{p_2}{1-p_2}w_2\right)\\
=&\frac{1}{Z}\left(\frac{p_1p_2w_1}{(1-p_1)(1-p_2)}+\frac{p_1p_2w_2}{(1-p_1)(1-p_2)}+\frac{p_1(1-p_2)w_1}{(1-p_1)(1-p_2)}+\frac{p_2(1-p_1)w_2}{(1-p_1)(1-p_2)}\right)\\
=&\frac{1}{Z}\left(\frac{p_1w_1+p_2w_2}{(1-p_1)(1-p_2)}\right)=\frac{p_1w_1+p_2w_2}{w_1+w_2},\ \text{since } Z=\frac{w_1+w_2}{(1-p_1)(1-p_2)}
\end{aligned}
$$

To show $Z=\frac{w_1+w_2}{(1-p_1)(1-p_2)}$, I sum over the weights for all the possible assignments (with both $b=0$ and $b=1$). I use $w(\mathcal{I})$ to represent the weight of the clauses satisfied by the assignment $\mathcal{I}$. Due to the hard clauses in the MLN, there are only eight assignments consistent with the clauses.

$$
\begin{aligned}
Z=&w(b=1,t_1=1,t_2=1,h_1=1,h_2=0)+w(b=1,t_1=1,t_2=1,h_1=0,h_2=1)+\\
&w(b=1,t_1=1,t_2=0,h_1=1,h_2=0)+w(b=1,t_1=0,t_2=1,h_1=0,h_2=1)+\\
&w(b=0,t_1=0,t_2=0,h_1=1,h_2=0)+w(b=0,t_1=0,t_2=0,h_1=0,h_2=1)+\\
&w(b=0,t_1=1,t_2=0,h_1=0,h_2=1)+w(b=0,t_1=0,t_2=1,h_1=1,h_2=0)\\
=&\ e^{\theta_1+\theta_2+\log(w_1)}+e^{\theta_1+\theta_2+\log(w_2)}+e^{\theta_1+\log(w_1)}+e^{\theta_2+\log(w_2)}+\\
&e^{\log(w_1)}+e^{\log(w_2)}+e^{\theta_1+\log(w_2)}+e^{\theta_2+\log(w_1)}\\
=&\ e^{\log(w_1)}\left(e^{\theta_1}+e^{\theta_2}+e^{\theta_1+\theta_2}+1\right)+e^{\log(w_2)}\left(e^{\theta_1}+e^{\theta_2}+e^{\theta_1+\theta_2}+1\right)\\
=&\ (w_1+w_2)\left(\frac{p_1}{1-p_1}+\frac{p_2}{1-p_2}+\frac{p_1}{1-p_1}\frac{p_2}{1-p_2}+1\right)\\
=&\ (w_1+w_2)\left(\frac{p_1(1-p_2)+p_2(1-p_1)+p_1p_2+(1-p_1)(1-p_2)}{(1-p_1)(1-p_2)}\right)\\
=&\ \frac{w_1+w_2}{(1-p_1)(1-p_2)}
\end{aligned}
$$

Thus we can show that the final distribution due to these MLNs is equal to the distribution presented in Equation 6.7. The same proof can be extended for multi-level combining rules as well.

Worked Example: Consider the FOCI statements about student satisfaction presented earlier. We now present the case where CR1 is *mean* while

$\mathsf{CR2}$ is *weighted-mean*. We show the translation to MLNs below. First, consider the CPT clauses. We use Alchemy's $+\mathsf{G}$ notation that creates one clause for every possible grounding of G. Assuming the grade G of the student can be any one of the five values $\{'A','B','C','D','F'\}$, each clause with a $+\mathsf{G}$ would be converted to five clauses with G being replaced by one of the grade values in every clause. We refer to the Alchemy manual for a detailed discussion on the $+$ notation[4]. The CPT clauses for each rule are as follows:

$$\begin{aligned} w_G &: \mathit{student}(S), \mathit{course}(C), \mathit{takes}(T,S,C), \mathit{grade}(T,+G) \Rightarrow \\ &\quad t1(S,T,C,+G) \\ w_Q &: \mathit{student}(S), \mathit{paper}(P,S), \mathit{quality}(P,+Q) \Rightarrow t2(S,P,+Q) \end{aligned}$$

where each w_G and w_Q are the log-odds for each grade (G) and quality (Q) respectively. Next, I present the multiplexer clauses.

$$\begin{aligned} \infty &: h1(S,T,C,G) \Rightarrow t1(S,T,C,G) \Leftrightarrow r(S,1) \\ \infty &: h2(S,P,Q) \Rightarrow t2(S,P,Q) \Leftrightarrow r(S,2) \\ \infty &: hr(S,R) \Rightarrow r(S,R) \Leftrightarrow \mathit{satisfaction}(S) \end{aligned} \tag{6.8}$$

The first two clauses serve to choose the intermediate values of r corresponding to rules 1 and 2. The third rule then chooses the final value of satisfaction from the two intermediate values. The stochastic function clauses are given by:

$$\log(w_1) : hr(S,1)$$
$$\log(w_2) : hr(S,2)$$

The above clauses specify the prior over the intermediate nodes as a function of their weights w_i.[5] At the first level, the value of the intermedi-

[4]http://alchemy.cs.washington.edu/user-manual/4_2MLN_Syntax.html

[5]These weights are the weights of the combining function and must not be confused

ate node is chosen according to an uniform distribution (mean combining rule) and hence the weights of the MLN clauses are 0 and are not presented here. Finally, I present the integrity constraints that restrict the multiplexer to choose only one value from among a set of possible values

$$\begin{aligned}
\infty &: h1(S,T1,C1,G1) \wedge h1(S,T2,C2,G2) \\
&\qquad\qquad \Rightarrow (T1 = T2 \wedge C1 = C2 \wedge G1 = G2) \\
\infty &: \exists\, T,C,G\; h1(S,T,C,G) \\
\infty &: h2(S1,P1,Q1) \wedge h2(S2,P2,Q2) \Rightarrow (P1 = P2 \wedge Q1 = Q2) \\
\infty &: \exists\, P,Q\; h2(S,P,Q) \\
\infty &: hr(S,R1) \wedge hr(S,R2) \Rightarrow R1 = R2 \\
\infty &: \exists\, R\; hr(S,R)
\end{aligned} \tag{6.9}$$

Noisy Functions: For this case, let us assume a single rule and that the different instantiations of that rule are combined using a noisy function. For *Noisy-Or*, the marginal is computed as,

$$P(b = T|a_1,\ldots,a_n) = 1 - \prod_{i=1}^{n} f_i^{a_i} \tag{6.10}$$

where f_i's represent the probability that a present (Boolean-valued) cause, a_i, fails to make the result b true. When converting these to MLNs, the transformation is mostly similar to the earlier case. Though the CPT clauses are constructed similarly, I present them for clarity. They are of the form:

$$\begin{aligned}
\infty :&\; \neg a(X,Y) \Rightarrow \neg t(X,Y,1). \\
w_i :&\; a(x_i,Y) \Rightarrow t(x_i,Y,1).
\end{aligned}$$

with the weight of the MLN clauses.

where, $w_i = \log((1-f_i)/f_i)$. As can be seen, if $a(X,Y)$ is false for a particular value of X, $t(X,Y,1)$ will always be false while if a is true, t can be false due to some noise. The multiplexer and integrity clauses are similar to the average case. A careful reader will note that the multiplexer and integrity clauses are redundant for this case as they derive the r-values directly from t-values as shown below. The stochastic function (deterministic here) is given by,

$$\infty : r(Y,I) \Leftrightarrow \exists X.t(X,Y,I) \tag{6.11}$$

This asserts that $r(Y,i)$ is true if and only if some $t(X,Y,i)$ is true, which is effectively deterministic OR applied to noisy versions of the inputs. It can be shown that this set of clauses exactly capture the distribution given by Equation 6.10.

Noisy existentials can be constructed similarly, except that we have tied weights. When constructing noisy-and, the noise adds a probability of success instead of a probability of failure:

$$\begin{aligned} w_i : &\quad \neg a(X,Y,1) \Rightarrow \neg t(X,Y,1) \\ \infty : &\quad a(x_i,Y,1) \Rightarrow t(x_i,Y,1). \end{aligned}$$

The multiplexer and the stochastic functions are also modified accordingly to reflect the *And* function.

Worked Example: We now present the rules for the satisfaction example where CR2 is *Or* while CR1 is *Noisy-Or* with q_i as inhibition probability. The CPT clauses are

$$\begin{aligned} \log(\frac{1-q_1}{q_1}) &: student(S), course(C), takes(T,S,C), grade(T,G) \Rightarrow \\ &\qquad\qquad t1(S,T,C,G) \\ \log(\frac{1-q_2}{q_2}) &: student(S), paper(P,S), quality(P,Q) \Rightarrow t2(S,P,Q) \end{aligned}$$

Note that the CPT parameters are a function of the noise (inhibition) for the two rules. The multiplexer clauses can be constructed similar to the weighted mean case given in Equation 6.8. The stochastic function clauses are created according to the following clauses. Note that the stochastic function clauses state that there is an Or function at each level (the noise at the first-level is captured in the CPT clauses).

$$\begin{aligned}
&\infty : \mathtt{r}(S,1) \Leftrightarrow \text{Exists T,C,G} \;\; \mathtt{t1}(S,T,C,G)\\
&\infty : \mathtt{r}(S,2) \Leftrightarrow \text{Exists P,Q} \;\; \mathtt{t2}(S,P,Q)\\
&\infty : \mathit{satisfaction}(S) \Leftrightarrow \text{Exists R} \;\; \mathtt{tr}(S,R)
\end{aligned}$$

The integrity constraints are similar to the earlier case (Equation 6.9). The example here combines *Noisy-Or* with the Or combining rule. We can similarly imagine combining different decomposable combining rules at the different levels. The same templates can be used to construct the different sets of combining functions.

6.2.2 MLN macros

While theoretically MLNs can represent most of the distributions that we considered, it seems impractical to expect a domain expert to come up with these rules. The domain expert needs to be an MLN expert as well and has to understand the translation. In this section, I present a macro that can be used to construct MLNs given the domain expert's statements. The key idea is to remove the burden of specifying the MLNs from the user and allow our "translator" to create the MLNs corresponding to the true distribution. I now present the structure of the macro:

CR {
CR_1: $X^1_1 \wedge ... \wedge X^1_{n_1} \Rightarrow Y$
CR_2: $X^2_1 \wedge ... \wedge X^2_{n_2} \Rightarrow Y$
...

```
}
```

The above macro can be interpreted as: CR, CR_1 and CR_2 are the combination functions – *And, Or, Noisy-Or, Noisy-And*, etc. While CR_i combines the multiple instantiations of clause i, CR combines the multiple clauses. X_i^j and Y are predicates. The first clauses specifies n_1 causes for the target predicate Y. X_1^1 is the first cause of Y in rule 1 and so on. Instead of writing 2^n different clauses, the user specifies a single clause that is then unrolled into the different clauses by the translator. The user can specify several clauses that can be combined.

The translator then converts these macros to the MLN clausal representation. A natural question now is: where do the weights come from? A simple solution would be to construct the clauses and allow the underlying MLN package (in our case *Alchemy* (Kok et al., 2010)) to learn the weights. We could hold the weights of the hard clauses (integrity constraints and some multiplexer clausers) and instruct Alchemy to learn the weights of only the "soft clauses" leading to a more efficient learning. In cases where the training data is not available, we allow the conditional probabilities to be specified for the different configurations of the predicates in the body of the clause. Hence the clauses are now of the form, $\langle p_1, p_2, ..., p_{2^n} \rangle X_1 \wedge ... \wedge X_n \Rightarrow Y$, where $p_i = P(Y = T | Config(X_1, ..., X_n) = i)$ is the conditional probability of the target being T given that the truth value of the predicates in the body from the i^{th} configuration. Similarly, the parameters of the combining rules (e.g., weights of weighted mean) can also be specified as $CR\langle w_1, ..., w_m \rangle$ for m clauses. Based on the combining rule used, the translator then computes the weights of the different clauses based on the probabilities and assigns the weights to the corresponding clauses.

6.3 Experiments

In the following experiments, we use the Alchemy system[6] to learn the weights and/or perform inference. We use the same settings for both MLNs with combining rules (denoted by MLN^{+}) and the default MLNs (MLN^{*}). The clauses of the MLN^{*} are the parent configurations of the CPT of each rule. Hence for each independent parameter of the CPT, there exists a clause in MLN^{*}. MLN^{*} was chosen so that it had the same number of parameters as that of a directed model to make a fair comparison. The clauses of MLN^{+} consist of the CPT clauses and the multiplexer, stochastic function and integrity clauses. For both MLN^{+} and MLN^{*}, we used the same settings for the learning and inference algorithms (i.e., used the same number of iterations, discriminative learning, same number of MCMC steps, MC-SAT for inference, etc.).

I present our learning experiment on the *UW-CSE* domain. The goal of the experiment is: given minimal domain knowledge (typically two rules to predict the target), will the structure imposed by combining rules be useful in learning a good model? For the UW-CSE dataset, the task was to predict the *advisedBy* relationship between a student and a professor. The rules that we use were:

```
N-Or {
     N-Or:student(S) ∧ professor(P) ∧ course(C) ∧
          taughtBy(P,C,Q) ∧ ta(S,C,Q) ⇒ advisedBy(S,P)
     N-Or:student(S) ∧ professor(P) ∧
          publication(P,W) ∧ publication(S,W) ⇒ advisedBy(S,P)
    }
```

The first rule uses the noisy-Or combination function over all the courses taught by the professor P and ta-ed by the student S. The second

[6]http://alchemy.cs.washington.edu/

rule uses the noisy-Or combination function over the common publications between the student and the professor. Finally, noisy-OR combination function is applied over the two rules to predict the probability of the student S being advised by the professor P. MLN^* used all the combinations of the predicates in the head of the clauses (i.e., $2^5 + 2^4 = 48$ clauses) and learned weights for each of them. For MLN^+, we used Noisy-Or as the combining rule at both levels. We learned the weights using Alchemy and used MC-SAT for performing inference. We trained the algorithms on the AI group data that consisted of 35 positive instances of the *advisedBy* relation.

I present the average likelihood (i.e., $\frac{1}{n}\sum_i P(y_i = \hat{y}_i)$, where n is the number of examples, $\hat{y}$ is the predicted label and y is the true label) of the test set in the last column of Table 6.1. Note that since we are in the relational setting, the test set will mostly consist of negatives. Hence, an algorithm that always predicts *false* will have a reasonably high likelihood. To avoid this situation, we forced the test set to contain 50% negative examples by sampling the negative examples randomly. This way a likelihood of 0.5 would mean that everything is either predicted *true* or as *false*.

Algorithm	AUC-ROC	AUC-PR	Likelihood
MLN^+	0.560	0.672	0.611
MLN^*	0.472	0.523	0.500

Table 6.1: Testset results on the UW-CSE dataset.

We also compare the area under curve for the ROC and PR curves. MLN^* was not able to learn reasonable weights with a small number of rules and hence predicts everything as false. In a test-set with 50% positive examples, this yields a likelihood of 0.5. On the other hand, with MLN^+, we were able to learn a more reasonable model that has a higher likelihood. The values of AUC for ROC and PR for MLN^+ are also significantly higher than MLN^*. More importantly, MLN^+ did not predict every query predicate as 0 or 1 and instead had a reasonable distribution over the target. So if we have limited background knowledge (two rules in this experiment), we showed that using combining rules improves the accuracy of the model. To see if additional knowledge will improve the performance of default MLNs, we added more rules to MLN* (seven more rules from the Alchemy website that were earlier used in other MLN experiments (Richardson and Domingos, 2004; Singla and Domingos, 2005) to predict *advisedBy*). Weight learning on MLN* with more rules improved the average likelihood to 0.63.

6.4 Discussion and Future Work

Combining rules capture the notion of causal independence for SRL models. I have presented an algorithm for representing a class of combining rules (decomposable combining rules) in an undirected model (specifically, an MLN). Our experiments demonstrated that for a small number of clauses, combining functions are useful in learning more accurate models. The structure imposed by these functions helps in guiding the learning algorithms towards reasonable weights. Jaeger (2008) showed that Relational Bayesian networks can capture some MLN types and pointed out to the reverse as an open problem. We take an important step in that direction by showing how MLNs can capture combination functions of the directed models and, in turn, most of the features of directed models.

However, inference on the MLNs with the translated combining rules is 4-5 times slower than the one that does not use the combining rules. The problem is that the inference engine does not utilize the hidden structure that is naturally exploited by the directed models. One possible future direction is to develop specialized inference algorithms that can detect structure in MLNs (Niu et al., 2012a) and exploit it for efficiency. A more general and important direction is to develop hybrid models that allow one to specify different parts of the model differently and combine them using a decomposable structure. This should allow the application of specialized learning algorithms inside each module and then combine the results in an efficient manner.

7 ADDITIONAL EXPLORATIONS

In this chapter, I present real-world applications that I was involved in along with the exploratory experiments conducted. In Section 7.1, I present the application of RFGB to the task of temporal relation extraction in text (published in StarAI workshop (Khot et al., 2012)). In Section 7.2, I present an approach for Alzheimer's prediction using RFGB over segments from MRI images (published in International Journal of Machine Learning and Cybernetics (Natarajan et al., 2013)). Next, I present a large-scale NLP task where we use a propositional model for relation extraction over terabytes of data (published in Text Retrieval Conference (Khot et al., 2013b)).

7.1 Temporal Relation Extraction

Information extraction (IE) has been an important problem in the Natural Language Processing (NLP) community (Manning and Schütze, 1999). One specific challenging IE problem is extraction of temporal ordering between events and temporal expressions. The introduction of corpora such as the TimeBank (Pustejovsky et al., 2003b) and TimeML (Pustejovsky et al., 2003a) makes it possible to use machine learning methods to learn ordering relations between events and time expressions (timex). For example, for the sentence "He met the ambassador on June 3rd.", we would like to extract the relations `OVERLAP("met", "June 3rd")` and `BEFORE("met", DOCTIME)`, where DOCTIME corresponds to the document's creation time.

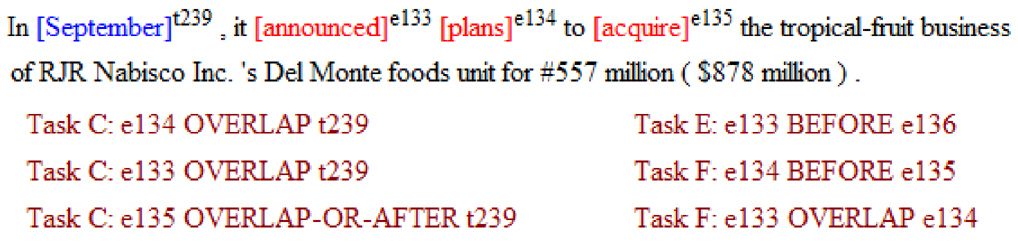
The TempEval dataset (Verhagen et al., 2007) simplified the TimeML annotations by using six coarse-grained, temporal-ordering relations between events and timexes; between events and document creation time; and between events. TempEval-2 (Verhagen et al., 2010) extended this dataset to six tasks, including the three tasks from the original dataset.

Most of the approaches applied to the TempEval tasks use propositional features and independently learn relations for each task. However, learning to predict each task independently can lead to inconsistencies in the final prediction. For example, predicting event A happened before time T ($A < T$) and event B happened after time T ($T < B$) is inconsistent with predicting event A happened after event B ($A > B$).

There have been approaches to handle these global inconsistencies for propositional models, such as creating a globally consistent set of joint predictions by selecting from the individual predictions during inference (Chambers and Jurafsky, 2008). In this section, I concentrate on employing the relational approach to address this issue. Relational approaches have the advantage of focusing on the joint set of predictions during learning, rather than deferring the consideration of interaction among predictions to the inference step

Using SRL models such as MLNs allow joint inference across various examples and tasks. As shown above, in the TempEval task we need to ensure consistent ordering between events and timexes. Also, the independence and identically distributed (i.i.d.) assumption made by most propositional methods is not valid as the events are not independent, further making the case for using SRL models. Moreover, many constraints are not necessarily hard constraints in this task requiring the use of probabilistic models. For example, if event A occurred before event B and event B overlapped with C then it is *likely* that event A occurred before event C. Hence, Yoshikawa et al. (2009) and more recently UzZaman and Allen (2010) used Markov Logic Networks (MLNs) to specify the model as well as the global constraints as weighted first-order logic rules.

We learn rules in the absence of expert advice by using RFGB. We also develop two extensions to leverage expert advice whenever available. Preliminary results of our approach show promise for structure learning approaches in IE and other NLP tasks. This work has been published in the StarAI workshop (Khot et al., 2012).



In [September]^{t239} , it [announced]^{e133} [plans]^{e134} to [acquire]^{e135} the tropical-fruit business of RJR Nabisco Inc. 's Del Monte foods unit for #557 million ($878 million) .

Task C: e134 OVERLAP t239

Task C: e133 OVERLAP t239

Task C: e135 OVERLAP-OR-AFTER t239

Task E: e133 BEFORE e136

Task F: e134 BEFORE e135

Task F: e133 OVERLAP e134

Figure 7.1: Sample TempEval-2 annotations.

7.1.1 TempEval tasks

The TempEval task (Verhagen et al., 2007) in SemEval 2007 used the Time-Bank corpus to create three separate relation-extraction tasks:

1. identify relations between event and timex,

2. identify relations between event and document time and

3. identify relations between events

The TempEval-2 task extended this dataset to include the problem of identifying timexes and events along with their properties. It also modified task 3 of the previous TempEval into temporal ordering tasks between: 1)

1. events in consecutive sentences

2. events where one event syntactically dominates the other

I show initial results in Section 7.1.3 for identifying relations between events and temporal expressions (called task C in TempEval-2). Figure 7.1 shows a sample TempEval-2 annotation, where tokens `e133`, `e134` and `e135` are the event words whereas `t239` marks a timex. In this example, since the announcement happened in September, the annotations marked an `OVERLAP` relation between `e133` and `t239`.

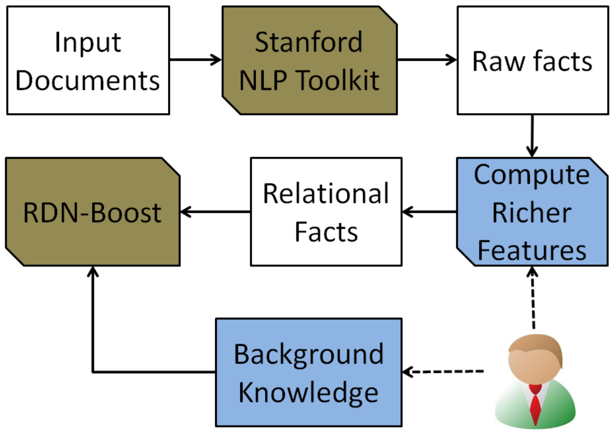


Figure 7.2: Flowchart describing our approach for relation-extraction.

7.1.2 Structure learning for TempEval-2

I first use the Stanford NLP toolkit[1] to convert the documents into first-order logic facts. I then use these raw features to create richer features based on our analysis of the domain. If provided, I can also use expert advice such as the rules written by previous work in this domain as the initial model. Given the initial model and the set of facts, I use RDN-Boost to learn a joint model for the target relations. Figure 7.2 presents our approach.

Raw Facts: For each sentence, the Stanford NLP toolkit returns the tokenization, parse tree, dependency graph and named entity[2] information. We create a word object for each token in the sentence and a phrase object for each phrase in the parse tree. Table 7.1 presents a subset of the generated facts. Dependency paths[3] are considered to be important features for relation extraction and hence we create a special predicate to store the dependency path between every pair of words. Since there can be

[1]http://nlp.stanford.edu/software/corenlp.shtml

[2]Recognizing entities in text such as Obama (PERSON type), Washington (LOCATION type) and Saturday (TIME type).

[3]Dependency paths are paths in the dependency graph between a pair of entities

Example	Definition
wordText(W3, occurred)	Word W3 corresponds to the token "`occurred`" in the article
wordLoc(S1, W1, 1)	Word W1 is the first word of the sentence S1
wordType(W5, NN)	Word W5 is a noun (NN)
phraseType(P3, NP)	Phrase P3 is a noun phrase (NP)
phrHasWord(P3, W5)	Phrase P3 contains the word W5
headWord(P5, W11)	Word W11 is the head word of P5
depType(W3, W7, CCOMP)	Dependency graph contains an edge of type CCOMP between W3 and W7

Table 7.1: Sample facts generated using the Stanford toolkit.

many such paths, we create the dependency path facts only if the path length is smaller than seven. For TempEval, we also convert the event and timex properties to relational facts such as `eventHasProperty(Event, Property, Value)`.

Domain Advice: We allow the provision of two forms of domain knowledge:

(1) Specialized Features. We noticed that for most of the valid event-timex pairs (i.e. having some relation), the event word is present in the dependency path (DP) from the timex to the root of the dependency graph (DG). Hence, if the DP goes up the tree and then goes down i.e. if there is a $\swarrow\searrow$ in the DP, then it is a strong signal that the event and timex are *not* related. We added a predicate `veeInDepPath(W1,W2)` which is true if neither W1 nor W2 is the ancestor of the other word. For example, in Figure 7.3 we would create the fact: `veeInDepPath("be", "2002")`.

Typically, a timex `t` is related to the first verb that appears in the DP from `t` to the root of the DG. However, additional verbs in the path to the root can also be related to `t` if they are preceded by special dependency tags (e.g. CCOMP). In order to learn such tags, we include a predicate

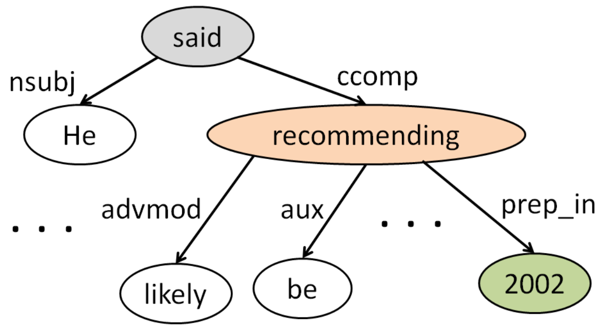


Figure 7.3: Dependency graph for a sentence where OVERLAP relation exists between "said" and"2002."

`verbAlongDependencyPath(word, word, verb, depType)` to represent this feature. We now let RDN-Boost discover which dependency types could be present for valid relations. Figure 7.3 shows a snippet of a DG. Although there is a verb in the DP from "2002" to "said," since "recommending" is connected by a CCOMP dependency type, "2002" applies to "said" too.

(2) Expert Rules. For the TempEval task, Yoshikawa et al. (2009) designed rules to encode the constraints for consistent ordering between events, timexes and document times. Similar rules were also used by the TRIPS/TRIOS system (UzZaman and Allen, 2010) for the TempEval-2. We can use these rules as the initial model for RDN-Boost. Each Horn clause is used as a part of the initial model for the predicate that appears in the head of the clause. For example, a sample rule used by previous approaches was relE2T(e1, t, "BEFORE") $\wedge$ relE2T(e2, t, "AFTER") $\rightarrow$ relE2E(e1, e2, "BEFORE"). This rule can be used as the initial model for predicting relE2E. While relE2T represents relations between events and timexes, relE2E represents relations between events.

7.1.3 Initial Results

I present the initial results of our approach on task C of TempEval-2. We did not use any cross-task rules in the initial model, since we learn a model for a single task. When not using any domain-specific features, RDN-Boost is able to achieve an accuracy of *0.56* on the test set. Including the domain-specific features improved the testset accuracy of the system to *0.60*. Most of the systems that competed in TempEval-2 had an accuracy ranging between 0.62-0.65. We believe with better features and simultaneously using the data from all the TempEval tasks to learn a joint model would further improve the results.

7.2 Alzheimer's Prediction

Alzheimer's disease (AD) is a progressive neurodegenerative condition that results in the loss of cognitive abilities and memory, with associated high morbidity and cost to society (Sun et al., 2009). Accurate diagnosis of Alzheimer's disease and its precursor, mild cognitive impairment (MCI) are important steps towards finding a cure and have been a focus of many studies (Sun et al., 2009; Ye et al., 2008; Supekar et al., 2008).

Magnetic resonance imaging (MRI) is a neuroimaging technique that can be used for visualization of brain anatomy with a high degree of spatial resolution and contrast between brain tissue types. Structural MRI methods have been used to identify regional volumetric changes in brain areas known to be associated with AD and MCI, demonstrating the utility of such methods for studying this disease (Sun et al., 2009; Ye et al., 2008). The work presented here has been published in International Journal of Machine Learning and Cybernetics (Natarajan et al., 2013).

Structural MRI approaches have identified changes in particular regions of the brain, such as reduction in size of the hippocampus, to be associated with AD and MCI (Ye et al., 2008). More recently, MRI data have become the focus of machine learning experiments aimed at classifying subjects as AD versus cognitively normal (CN) or MCI versus CN. Recent approaches employ network analysis (Sun et al., 2009; Supekar et al., 2008) or use machine learning directly on the voxels (Ye et al., 2008).

These approaches only consider two-way classification problem of differentiating AD from CN, in which case a clear decision boundary between these categories can be easily obtained. In reality, Alzheimer's patients span different stages from MCI to AD, making classification much more difficult.

We develop a novel data mining approach for the significantly more challenging problem of classifying the subjects into one of three categories $\langle \mathrm{AD}, \mathrm{MCI}, \mathrm{CN} \rangle$ given volumetric structural MRI data. Specifically, we propose a novel knowledge-based approach that allows the combination of state-of-the-art MRI data processing and modern machine learning techniques. Our pipeline consists of three stages – first is a *s*egmentation stage that takes volumetric brain MRI data as input and divides it into anatomically relevant regions, second is a *r*elational learning stage that uses the different segments obtained over the image to build multiple binary classifiers using RFGB and the final stage is the *c*ombination stage that combines the different classifiers to provide a single prediction.

7.2.1 Pipeline design

Our problem of Alzheimer's prediction is formulated as the following classification task:

Given a set D of tuples $\{\langle \mathbf{x}_1, y_1 \rangle, \cdots \langle \mathbf{x}_n, y_n \rangle\}$, where each $\mathbf{x}_i$ is a 3D voxel image corresponding to a subject and y_i is a class label (AD, MCI or CN).

Learn a function/classifier h that predicts y_i given $\mathbf{x}_i$.

One approach to this problem would be using standard propositional classifiers, where we assume the examples are independent and identically distributed. Unfortunately, for a problem as complex as the 3-way classification of AD, the standard propositional approaches may not capture the

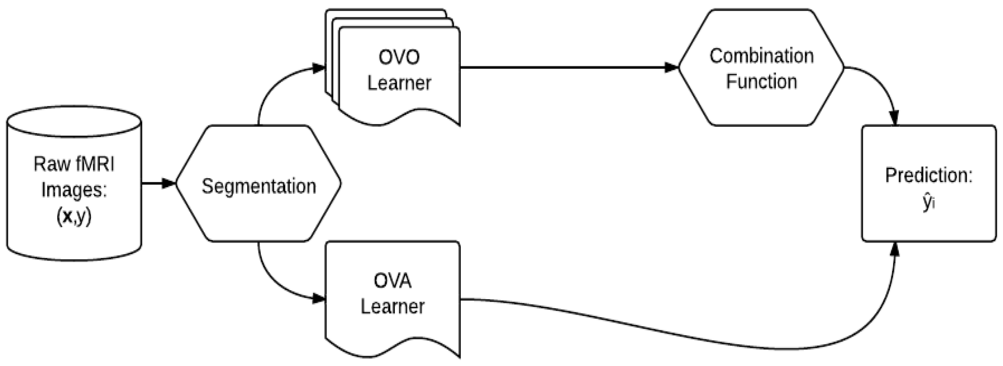


Figure 7.4: Graphical representation of the pipeline.

visual aspects of the image data. For example, directly using the voxels in the image as features does not capture aggregate properties about the important regions in the brain.

Instead, we model the function h as a three stage pipeline, i.e., $h(\mathbf{x})$ is approximated by $h_3(h_2(h_1(\mathbf{x})))$. Each stage h_i of the pipeline is designed to expose interesting and informative aspects of the data. We model the search for building the pipeline as a sequential search over individual stages. In particular we address the following three problems **(1)-(3)**.

(1) Image Segmentation

> **Given** the dataset D, **generate** a dataset $D' = \{\langle h_1(\mathbf{x}_1), y_1\rangle, \cdots, \langle h_1(\mathbf{x}_n), y_n\rangle\}$ where each $h_1(\mathbf{x}_i)$ is a representation of the image $\mathbf{x}_i$ segmented into regions.

To segment volumetric brain MRI data, we use two different segmentation techniques, namely

i) A knowledge-based segmentation method that partitions the MRI into 116 regions using Automated Anatomic Label (AAL) atlas [4].

[4]http://prefrontal.org/blog/2008/05/brain-art-aal-patchwork

ii) A knowledge-free segmentation technique based on Expectation Maximization (EM).

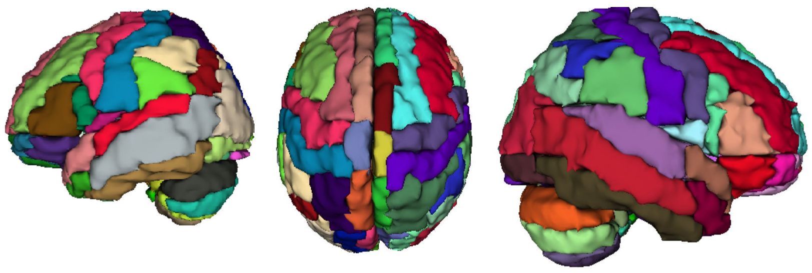


Figure 7.5: AAL atlas segmentation showing the different regions of interest in the brain.

The output of the image segmentation procedure, $h_1(\mathbf{x}_i)$ is a set of vectors $\langle\langle s_{i,1}, f(s_{i,1})\rangle \dots \langle s_{i,m}, f(s_{i,m})\rangle\rangle$ where each $s_{i,j}$ is a segmented region and $f(s_{i,j})$ is a vector of features and neighborhood information for $s_{i,j}$. Intuitively, $h_1(\mathbf{x}_i)$ can be viewed as a graph where each $s_{i,j}$ is a node, and there is an edge between two nodes if the corresponding regions are neighbors in the original image. An important thing to note here is that two examples in D' need not have the same number of regions. Also, two regions need not have the same number of features (because each region can have a different set of neighbors). This makes it difficult — if not impossible — to represent D' using a "flat" feature vector without extensive feature engineering. A relational representation, however, is ideally suited.

(2) Relational models

Given the dataset D', **train** a relational probabilistic classifier on D' that given an example $\langle h_1(\mathbf{x})\rangle$ generates example $\langle h_2(h_1(\mathbf{x}))\rangle$ where $h_2(h_1(\mathbf{x}))$ is a distribution over the classes AD, MCI and CN.

Before learning the classifier, we convert the segments and their features into first-order logic. Some of the predicates we used are presented in Table 7.2. The predicate names denote the attributes while the parameters are variables that define the attribute values. Note that the attributes of the regions are defined in a logical form that allows for different number of regions for different persons. Similarly, the predicate adj allows for neighborhood definitions and this will allow us to encode an arbitrary network structure of the brain and does not constraint the number of neighbors for a region. We denote all the query predicates (*ad, cn, mci*) as y and all other ones as x.

Predicate	**Explanation**
centroidx(P, R, X)	Centroid of region R is X
avgSpread(P, R, S)	Average spread of R is S
size(P,R, S)	Size of R is S
avgWMI(P, R, W)	Avg intensity of white matter in R is W
avgGMI(P, R, G)	Avg intensity of gray matter in R is G
avgCSFI(P, R, C)	Avg intensity of CSF in R is C
variance(P, R, V)	Variance of intensity in R is V
entropy(P, R, E)	Entropy of R is E
adj(R1,R2)	R1 is adjacent to R2
ad(P)	P has AD
mci(P)	P has MCI
cn(P)	P is cognitively normal

Table 7.2: Examples of predicates used in the pipeline. Here, P stands for a patient and R for a region. The last three predicates are the query predicates that are predicted by our classifiers.

A set of classifiers are trained to produce a distribution between every pair of classes (One versus One abbreviated as OvO) and one class against all (One versus All abbreviated as OvA). The OvO classifier is trained on examples from one class (say AD) as positives and examples from another (say CN) as negatives. The OvA classifier is trained on examples from one class as positive and examples from all the other classes as negatives.

Since D' is a relational database, we cannot use propositional classifiers and resort to relational methods. Additionally relational methods are well suited to leverage neighborhood information. We use RFGB to learn the binary classifiers for every class in the data (AD, MCI, CN). Now, we have everything together for the final stage (3) of our pipeline.

(3) Combining classifiers

> **Given** the classifiers learned from the previous stage, i.e., h_2 for the three different combinations, **design** a combination function h_3 that combines their results.

The result of previous step is a set of probabilistic classifiers for each pair of classes from AD, MCI and CN (in essence, three classifiers). Let us denote each classifier as $c^k, k = 1, 2, 3$. We have used the following combination functions:

- **Voting:** Each c^k outputs a prediction and the class has the maximum vote i.e., $argmax_c \sum_k [I(y^k = c)]$, where y^k is the predicted label of the k^{th} classifier and c is the class.
- **Weighted Voting:** In this case, $class = argmax_c \sum_k [w^k \cdot P(y^k = c)]$. We derived a gradient for the log likelihood of the training data and also used a grid search over the weight space.
- **Pairwise Coupling:** We considered the PC method (Hastie and Tibshirani, 1998) where the goal is to determine the posterior over each of the classes from the estimated joint distributions.
- **Classifier method:** We used the output of each OvO classifier to train a propositional classifier that combines the output of these different classifiers to make its final prediction. The input of the new classifier is essentially the predictions of the classifiers of the

previous stage. More precisely, the input is a set $P = \langle p_1^1, p_2^1, ..., p_3^3 \rangle$ for each patient i, where p_j^k is the posterior probability of the class j as predicted by the classifier k. Hence, we aim to learn a function h_3 such that $h_3(P) = y_i$ where y_i is one of AD, CN or MCI. The advantage is that we can combine the OvO results in a non-linear fashion.

The OvA strategy uses three classifiers, where each classifier discriminates class j from $j' \in \mathtt{class} \setminus j$. We use a simple aggregation method called *Maximum confidence strategy*, which is similar to the voting strategy presented earlier. The output class is taken from the classifier that has the largest posterior probability, $\mathrm{argmax}_c\, p_c$. For more details on the OvA aggregation, please refer to Galar et al. (2011).

Given the above combination functions, the resulting classifier predicts the disease progression (AD, MCI or CN) for the patient, given the volumetric MRI scan. The resulting classifier h is essentially a nested classifier $h_3(h_2(h_1(x)))$ as illustrated in Fig. 7.4.

7.2.2 Experimental results

We compare several versions of the algorithms on this task, including the list of propositional classifiers on the AAL segmented data and the relational classifiers using both segmentation methods (EM and AAL) as well as different combination functions. To understand the need for segmentation, we use modulated gray matter voxel data without any segmentation with LibSVM reported as SVMMG.

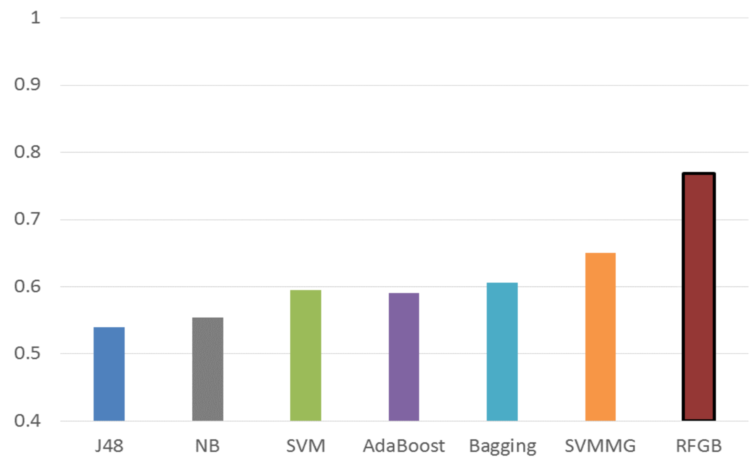
- **Propositional Classifiers** - Naive Bayes (NB), Decision Trees (J48), SVM, AdaBoost and Bagging on the AAL data and SVMs with gray matter data (that we denote as SVMMG) using Weka (Hall et al., 2009).

- **Relational OvO with AAL segmentation** - Using various combination functions: Weighted voting with grid search (AALGS), gradient descent (AALGD), bagging (AALB), AdaBoost (AALA) and Pairwise coupling (AALPC).
- **Relational OvO with EM segmentation** - Also using various combination functions: Weighted voting with grid search (EMGS), gradient descent (EMGD), bagging (EMB), AdaBoost (EMA) and Pairwise coupling (EMPC).
- **Relational OvA** - With AAL (OvAAAL) and EM (OvAEM).

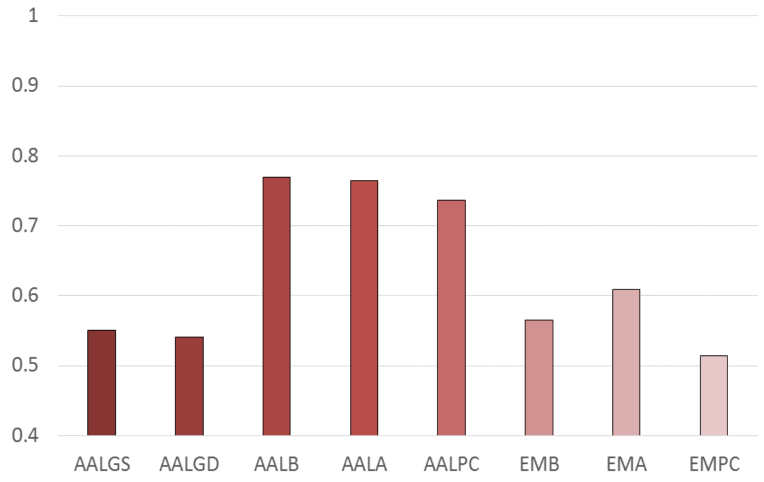
First, we compare the propositional classifiers with AAL segmentation presented in Figure 7.6a. We use Weka (Hall et al., 2009) with the multi-class classification setting for the propositional approaches. As can be seen from the figure, SVMMG, which uses the voxel data as is, does better than the propositional algorithms which use the AAL segmented data. The best relational algorithm (AAALB) does even better than the SVMMG approach.

We also evaluate different versions of the relational learning algorithms presented in Figure 7.6. We also included the OvA classifiers with AAL and EM in the results. It can be seen that the best performing algorithms use AAL segmentation technique. AALB has the best results among the different relational algorithms. The other classification functions did not have nearly as good a performance as bagging but are significantly better than the propositional algorithms. This clearly shows that treating the problem as a multi-class classification problem may not be the best solution (OvA methods also do not perform well). Instead, posing the problem as a slightly more complex OvO problem significantly improves performance.

In general, the relational methods have a superior performance compared to the propositional algorithms with AAL segmentation. The knowledge-based segmentation algorithm of AAL also has a higher performance than



(a) Propositional classifiers compared against the relational AALB (red with black border).



(b) Comparison of different relational classifiers.

Figure 7.6: Classification performances in terms of "Area under the ROC curve" of the different algorithms on Alzheimer's prediction.

the knowledge-free EM algorithm. An interesting future direction would be to guide the EM algorithm using the domain knowledge to improve segmentation. In problems such as identifying MCI patients who are likely to develop AD, it may help to combine the clinical knowledge to guide the segmentation algorithm and the classifiers.

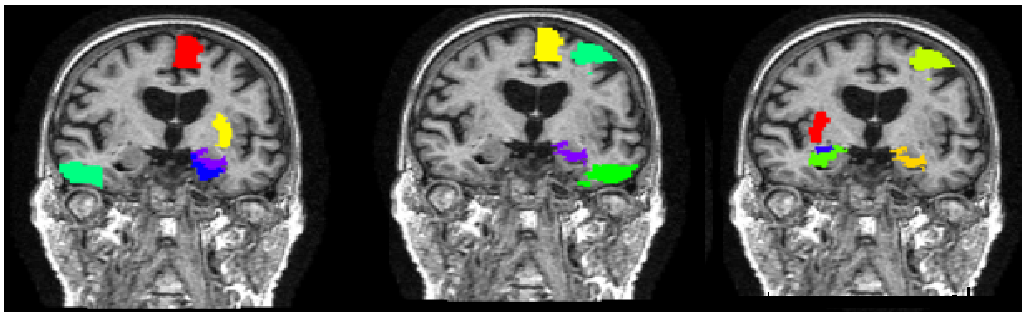


Figure 7.7: Predictive segments as identified by our pipeline (different colors indicate different regions).

I present the segments that are used in our learned models in Figure 7.7. These are the regions that discriminate among the classes as identified by our models and correspond to the medically relevant regions as verified by neuroradiologists. The first, second, and third images correspond to predicting AD (vs. CN), AD (vs. MCI) and MCI (vs. CN) respectively. Our algorithm shows consistency in detecting the regions (for example, hippocampus, occipital, parietal and temporal regions) that are known clinically to be affected by AD (Sun et al., 2009). This shows that the learning algorithms perfectly complement the segmentation algorithms in this task.

7.3 Knowledge Base Acceleration

The Knowledge Base Acceleration (KBA) task[5] seeks to help humans expand knowledge bases like Wikipedia by automatically recommending edits based on incoming content streams. To this end, KBA systems must filter a large stream of text to find changes to a knowledge base. To recognize changes in knowledge base profiles for particular entities of interest,

[5]http://trec-kba.org/

a KBA system has to extract relations (slot fillers) for these entities from the corpus stream and find novel relations from these extractions. This work has been published in Text Retrieval Conference (Khot et al., 2013b).

Given the scale of the streaming corpus (~9TB), running a relation extraction system on all the documents is infeasible. Hence, our system had to filter down the documents for relation extraction. Moreover, the filtering step needs to be much faster than the relation extraction step. Even after filtering the documents, we need a scalable learning and inference system to perform relation extraction. Since many of the relations in this task are not defined in other tasks (as a result, potentially lacking training data), we also needed a flexible system which allows us to easily specify new features or rules for these relations.

We use basic string search to quickly filter out irrelevant documents (that do not mention the query entities). From the filtered list of documents, we extract relations using Elementary (Niu et al., 2012b; Zhang and Ré, 2013), a statistical inference and learning system. A key advantage of the Elementary system is that it is a prototype system that scales to very large corpora. A secondary advantage of this system is that it uses Markov logic networks (Domingos and Lowd, 2009), allowing to model and capture rules that are likely, but not certain, to be correct. We use the Stanford NLP toolkit[6] to extract parse trees, dependency graphs and named entities to generate the features necessary for Elementary. We use the model learned for TAC-KBP 2010 (Ji et al., 2010) by mapping the relations from KBP domain to the KBA task. We design features to handle the new relations and entity types.

I learned various lessons in our first attempt at KBA. Although we filtered down the documents for relation extraction, we still had to download all the data and decrypt it locally, which was time-consuming. In hindsight, performing the filtering on Amazon Cloud[7] would have been

[6]http://nlp.stanford.edu/downloads/corenlp.shtml
[7]http://aws.amazon.com/ec2/

more efficient. We also assume given the size of the corpus, that most of the relations would be mentioned multiple times. Hence we did not rely on having all the possible rules for each relation, but on capturing the common cases. But since the set of entities were not popular, we were not able to extract many slot values for these entities.

The rest of the section is organized as follows. In Section 7.3.1, I present a few details about the Streaming Slot-Filling task (SSF). Following that, I present the details of our approach in Section 7.3.2. Finally, I briefly discuss the results of our approach and future steps to further improve them.

7.3.1 KBA task

The KBA track provides a large streaming corpus by breaking the documents up into hourly chunks, which can then be processed sequentially. The track also provides a list of target entities represented as links to Wikipedia[8] or Twitter[9] pages. Given the corpus and entity list as input, the track consists of two tasks:

- Cumulative Citation Recommendation (CCR) task. The CCR task involves filtering documents worth citing in a profile of a target entity. The system needs to recognize whether a document is *useful* (time-invariant, e.g., place of birth) or *vital* (timely, e.g., title).
- Streaming Slot Filling (SSF) task. The SSF task involves detecting changes to slot filler values (relations) for target entities. The system needs to extract slot filler values for target entities and then recognize changes to slot values. We concentrated only on this task in our system.

[8]http://www.wikipedia.org/
[9]http://twitter.com/

7.3.2 Our approach

The three key challenges that we faced with Streaming Slot Filling task are:

1. Handling the large scale of data
2. Extracting slot values for target entities
3. Detecting novel slot values

To solve each of these problems, we developed heuristic approaches that I outline in this section.

Large scale

Due to the scale of the proposed task (~9TB of compressed data), it is not feasible to perform relation extraction on the entire corpus in a reasonable amount of time ($<$ 2 weeks). Similarly, employing many of the standard, publicly available natural language processing tools would not be feasible as well. In order to make the corpus size manageable, we search for target entities (and variants of their names) in the articles. For e.g., if “William Smith” is a person of interest, we accept any document that mentions “William", “Bill" or "Smith". If an article contains any useful information about a target entity, we assume that some variant of the target entity name will be mentioned in the article. Computing this heuristic is very fast, as it does not require any processing to be done on the article and we only need to find the first match for a target entity in the article. We use a simplified version of the Aho-Corasick algorithm (1975) since we only need to find the first match. This reduces the size of the corpus from ~60TB to a manageable ~1TB. I implemented parallel version of the filtering algorithm on these articles to further speed up our system. Since we ran this step on our local cluster with limited storage space (order of hundreds of GB), this step still took a week to run.

Extracting slot values

To perform relation extraction, we first extract parse trees and dependency graphs for sentences in the filtered articles. Although TREC does provide some basic NLP annotations as part of the corpus, we employ Stanford toolkit because we needed dependency graphs as features for relation extraction. We then used the Elementary system developed at the University of Wisconsin-Madison to perform relation extraction. At a high level, Elementary first creates the potential mentions based on the named entities. It then creates a list of potential relations between pairs of mentions in the same sentence. For each potential relation, the path in the dependency graph and parse tree is calculated and used as features for a logistic regression model. The weights are learned for each feature and each relation type using gold-standard training data as well as distant supervision examples (Mintz et al., 2009). For further details about Elementary, please refer to Niu et al. (2012b).

Once these features are obtained, we need to learn the weights for the features in Elementary. But for the initial runs of the SSF task, there was no available training data. Hence, we use a model that has been previously learned using distant supervision (Mintz et al., 2009) on a subset of the relations for the Knowledge Base Population (TAC KBP 2010) task (Ji et al., 2010). We found mappings between the relation names from KBP task to the KBA task, shown in Table 7.3. For the relations in KBA that could not be mapped, we manually create features based on few sample sentences. To do so, we search for sample sentences containing these relations and add the corresponding dependency path or parse tree features to the model. We assume that capturing few features for these new relations will be sufficient. Given the scale of the data, we assume that at least one mention of a valid relation will be extracted using a simple relation extractor. Once we extract the relations, we employ Elementary's entity-linking model to link the slot filler entities with the target entities. We filter out the relations

KBP Relations	KBA Relation
per : date_of_death	DateOfDeath
per : title	Titles
per : spouse	SignificantOther
per : employee_of, per : member_of, org : member_of, org : top_employees	EmployeeOf
org : top_members	TopMembers
org : subsidiaries, per : schools_attended	Affiliate

Table 7.3: Mapping between KBP and KBA relations. All extractions of the relation type on the left were marked as KBA relations of the type on right.

that did not include any of the query entities.

Novelty detection

Given the extracted relations, we process them chronologically based on the document time to check for any changes in the slot values. Unlike the previous steps, this step can not be performed in parallel because novelty of a slot-filler depends on all the previous (chronologically) slot-fillers. For every relation, we check if we have already extracted any relations of the same type for the same entity. If not, we accept this relation as a novel slot value. If we already have a slot value extracted, we compare the slot values and accept the new relation as novel if the edit distance between the two values is large enough.

Figure 7.8 shows the overall system design as a flowchart. As mentioned before, we filter the documents using a basic entity filter. We perform entity linking and relation extraction using the Elementary software. We only accept relations over mentions linked to the target entities. We then perform our basic novelty detection over the stream of extracted rela-

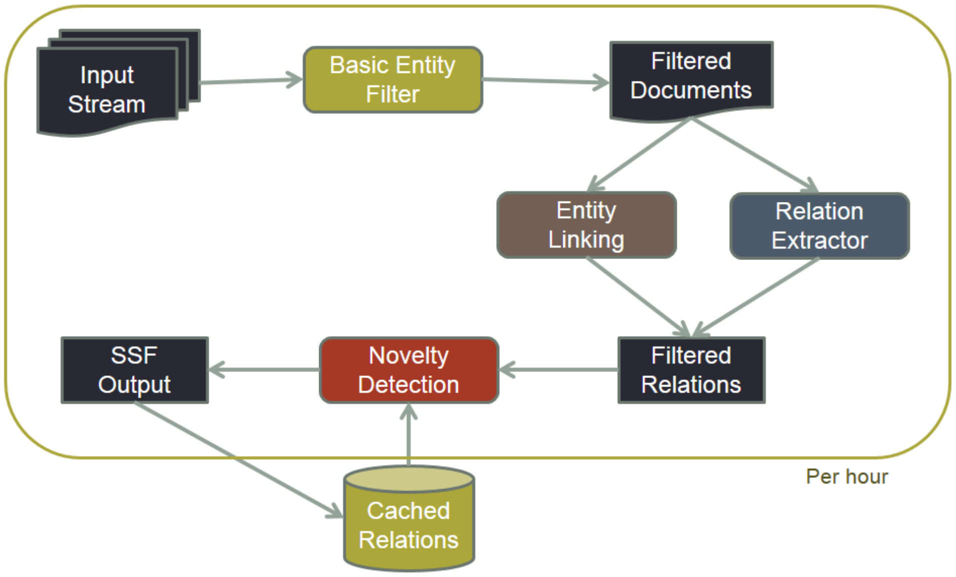


Figure 7.8: Overall system design.

tions where the previous extraction are cached. Only the novel relations are then used to create the output for the SSF task.

7.3.3 Results

There were two evaluation measures used by TREC for this task: (1) average F-1 score and (2) average Scaled Utility (SU; Hull and Robertson 1999). Each score is calculated over four stages in a pipeline:

- **SSF-DOCS**: In this stage, the system's capability to identify the slot type in a document is evaluated. The scores are calculated by comparing the slot types of the output run against the annotations (the slot values are ignored).
- **SSF-OVERLAP**: This stage evaluates the slot values of the output run by checking for overlap with the annotations, but only considers

the true positives from the previous stage.

- **SSF-FILL**: This stage too takes as input the true positives from the previous stage but checks if the output run recognizes the equivalence between the same slot values.
- **SSF-DATE_HOUR**: This stage checks if the system is able to recognize the first occurrence of the slot value and ignores the duplicates.

Since our approach concentrated on a subset of the relations (due to limited training data on some relations), the F1 score on any of these measures averaged over all the relations was low. In general for all the stages, our F1 score was close to zero but the scaled utility was close to the median value of all the reported scores. Our basic assumption that most of the relations would be mentioned multiple times in multiple ways was flawed. Since we only relied on capturing the common cases, we missed many extractions for the uncommon entities resulting in a low recall.

8 CONCLUSION

Machine Learning (ML) has been used to address challenging tasks such as relation extraction, web search, medical diagnosis, etc. (Ferrucci et al., 2010; Kosala and Blockeel, 2000; Kononenko, 2001). Although popular ML approaches are able to handle noise in the domains via probabilities, handling structured data with inter-related objects is non-trivial and often unaddressed. On the other hand, first-order logic (FOL) can naturally represent structured data, while traditionally FOL can only make Boolean predictions. One promising approach to handle noisy structured data in ML is Statistical Relational Learning (SRL), which combines first-order logic with probability theory. My thesis has mainly focused on developing learning methods for SRL.

The expressivity of SRL, which allows it to model noisy structured data, comes at the cost of complexity of the models. Although SRL models such as MLNs (Domingos and Lowd, 2009) and Problog (Raedt et al., 2007) make it relatively easy for an expert to specify the model (assuming some knowledge of first-order logic), such a model may not always be known or easy to define. Structure-learning approaches (Mihalkova and Mooney, 2007; Biba et al., 2008a; Kok and Domingos, 2009, 2010; Huynh and Mooney, 2008) have been developed for SRL models to directly learn the structure of the model. But the space of possible structures in relational models can be very large and most of these approaches need to re-learn the parameters for every candidate structure resulting in computationally slow approaches.

In this thesis, I presented structure learning approaches which learn the structure and parameters of the model simultaneously. I presented extensions of this approach for learning multiple relational models and learning under two kinds of missing observations. The various algorithms[1]

[1]Code and datasets available at http://pages.cs.wisc.edu/~tushar/Boostr/.

(used as paragraph **headings**) and the contributions of this thesis are presented next.

RFGB: In Chapter 3, I present the basic algorithm for Relational Functional Gradient Boosting (RFGB). Our approach uses Friedman's functional-gradient boosting (2001) to learn multiple weakly predictive models instead of a single complex model. The model is incrementally updated to correct the mistakes made in the previous iteration. In every iteration, the model is updated by fitting a regression function to the error and adding this function to the model. Since any off-the-shelf regression learner can be used to fit the regression function, advances made in the learning of relational regression functions can be easily plugged in. We use relational regression trees (Blockeel and Raedt, 1998) as the regression functions, which introduces non-linearity and context-specific independence in the model. The non-linearity results in our approach learning a linear sum of non-linear models. The context-specific independence reduces the number of parameters in the model and can be exploited for inference.

RDNBoost: In Section 3.3, I present our approach (Natarajan et al., 2012) to learn the structure of RDNs using RFGB as the underlying algorithm. RDNs approximate the joint distribution as a product of conditional distributions where each conditional distribution can be learned independently. We use RFGB to learn the conditional distributions for every predicate. This was the first boosting-based approach designed to learn the structure of RDNs. Based on prior work (Blockeel and Raedt, 1998; Srinivasan, 2004), we developed a relational tree learner which uses ILP "mode" specifications to limit the search space at every node. We also defined a special mode to learn non-trivial recursive rules.

MLNBoost: In Section 3.4, I present our approach (Khot et al., 2011) to learn the structure of MLNs based on RFGB. We note that MLN learning approaches maximize pseudo-likelihood which approximates the true joint distribution as a product of conditional distributions. Hence, MLNs can be viewed as a set of RDNs for the purposes of learning, which is an important contribution of my work. Since MLN semantics use the number of true groundings to calculate the probabilities, we modified the definition of the regression function in standard RFGB. We also developed a new scoring function for learning relational regression trees with MLN semantics. Our approach also uses a clause-based representation of the regression functions for faster inference, showing the flexibility of our approach.

Experiments: In Section 3.5, I present multiple experiments showing that our approach can learn more accurate models than state-of-the-art structure-learning approaches for RDNs and MLNs. I also show that in some cases the structure learned using RFGB can be more accurate than an expert-defined model and can be learned in less time than the weight-learning approaches. We evaluate on relational datasets from diverse tasks such as link prediction, entity resolution, information extraction, and adverse drug event predictions.

RFGB-EM: In Chapter 4, I present our approach for learning the *structure* of SRL models in the presence of missing data (Khot et al., 2013a). We derive EM update equations for relational models and use RFGB in the M-step to incrementally update the structure using RFGB. Since this approach extends RFGB to handle missing data, any functional-gradient boosting based approach developed for relational models can use this approach. We show the generality of our approach by learning the structure of RDNs, MLNs, and relational policies. Our approach is the first structural-EM approach derived for these three models.

RelOCC: One of the issues with applying our approach to NLP tasks was the lack of negatively labeled examples. Since our EM approach can not handle completely missing values of one class, I present a novel non-parametric approach developed for relational one-class classification in Chapter 5. We define the first relational distance measure that can be learned specifically for the task of one-class classification. We use relational trees to define this distance measure with a local splitting criterion for scoring a node. The splitting criterion is designed to maximize the one-class classification performance given the current distance measure. Hence, our approach can iteratively learn a sequence of relational trees where every new tree learned improves upon the one-class classification performance of the previous model. Moreover our approach can be viewed as propositional one-class classification techniques such as kernel density estimation (Parzen, 1962) and SVDD (Tax and Duin, 1999) with a relational distance measure learned specifically for this task.

MLN-CR: Apart from learning of SRL models, this thesis also presents an approach to transform knowledge from directed models to undirected models, namely MLNs. Although it has been shown that simple directed models can be represented using MLNs, combining rules from relational directed models had not been translated into MLNs. In Chapter 6, I present an algorithm to translate multi-level combining rules into a MLN thereby introducing independence of causal influence (ICI) in MLNs. I show two combination functions, noisy-or and weighted-mean, that can be represented using our approach. I define four types of clauses based on prior work on combination functions (Natarajan et al., 2008) that can be used to represent both of these combination functions. I also show the correctness of our transformation and how ICI can improve the accuracy of the model with very few rules.

Finally in Chapter 7, I show how RFGB can be used for Alzhiemer's disease prediction using MRI images as well as to augment expert rules for temporal relation extraction. I present our approach for a large-scale novel relation extraction task, namely TREC Knowledge Base Acceleration (KBA), where we processed terabytes of streaming data to detect changes in extracted relations.

8.1 Future Work

Following the work presented, there are multiple avenues of future research and open problems. I have presented some of the future directions specific to the approaches mentioned above in the corresponding chapter's discussion section. Of these directions, I present two major directions of future research next, namely learning of directed models and efficient inference based on our models.

8.1.1 Directed models

As shown in Chapter 3, I used relational boosting to learn models for bidirected cyclic models (RDN) and undirected models (MLN). A potential extension of our work is to learn directed acyclic relational models such as Probabilistic Relational Models (PRM; Getoor et al. 2001). Unlike cyclic models, directed acyclic models always guarantee that the product of the conditional distributions equals the joint distribution. Since directed models do not have a normalization term, commonly required by undirected models, inference and consequently learning for these models can be much more efficient than undirected models.

To extend our approach to directed relational models, we need to ensure acyclicity of the model. Acyclicity can be ensured in Bayesian networks by checking for cycles in the network generated using all instances, whereas relational models can check for cycles at the first-order logic level. For example, a model for `disease` that contains only the clause $\mathit{disease}(X) \leftarrow \mathit{ageAbove}(X, 50)$ would never result in a cycle in the ground network because the $\mathit{disease}$ predicate depends only on the $\mathit{ageAbove}$ predicate. Hence, we can avoid generating the ground network for all the instances by leveraging the first-order logic representation. On the other hand, clauses such as $\mathit{parent}(X, Y) \wedge \mathit{disease}(Y) \rightarrow \mathit{disease}(X)$ appear to have created a cyclic dependency, since the $\mathit{disease}$ predicate depends on the parent and $\mathit{disease}$ predicates. But whether a person has a disease or not would only depend on his ancestors having that disease or not. If we generated the ground network for this clause, it would not contain any cycles. Hence, checking for cycles on the predicates may ignore some valid structures, while checking for cycles on the ground network would be too expensive, as it would require grounding out every candidate structure.

To handle the acyclicity condition, one approach is to order the predicates based on some information-theoretic measure and use the ordering to ensure acyclicity. Once we have an ordering on the target predicates, say $\{p_1, \ldots p_k\}$, we use only $p_1, \ldots, p_{i-1}$ and the evidence predicates as parents for p_i. Since there is no edge from p_j to p_i where $i \leqslant j$, any path in this graph starting from p_j would never loop back to p_j. Now, we can use our relational boosting approach to learn the conditional distributions for each predicate with the restricted set of parents. So the problem of avoiding cycles during structure-learning can be reduced to finding an ordering over the predicates.

Cyclic dependencies on predicates do not always imply cyclic dependencies on the groundings. Requiring acyclicity in the predicate dependencies may result in learning a sub-optimal model. For example, consider the task of classifying tokens (X) within citations into three categories: $\{\mathit{title}(X), \mathit{author}(X), \mathit{venue}(X)\}$. One obvious rule is "if the word before X is not a punctuation and that word is of type venue then X is of type venue too". We can write that as a rule: $\mathit{before}(X, Y) \wedge \mathit{notPunct}(Y) \wedge$

$venue(Y) \rightarrow venue(X)$. This rule results in a cyclic dependency among the predicates but not in the ground network.

PRMs (Getoor et al., 2001) have handled this problem by requiring the user to specify the predicates in a domain that would not result in a cycle. Rather than relying on the user to specify the acyclicities, potentially we can use the data to check if a potential structure will result in a cycle. Naively checking for cycles in the ground candidate network would be computationally intensive. However, we can sample examples or paths from the ground network to approximately prune out acyclic structures. We can perform a random walk over the ground network without constructing the complete network. We can start with a ground literal and randomly pick a parent ground literal by grounding a random node from a random tree. We repeat this process till we reach a literal that we already visited proving that the model is cyclic or reach an evidence literal. When we reach an evidence literal, we start the random walk from a new ground literal. Also during the random walk, with probability p, we can restart the walk from a new ground literal.

8.1.2 Scalable models

Although our approaches can scale to millions of facts (Khot et al., 2012), they cannot handle web-scale domains yet. There are multiple approaches possible to scale our learning algorithm.

Databases: One approach to scale RFGB is to use databases to store and process the data, instead of processing the data in-memory. We can now replace Horn clauses with SQL (Ramakrishnan and Gehrke, 2003) queries, which can speed up both the learning as well as inference of our models. When learning MLNs, we also need to calculate the number of groundings of a Horn clause, which can be done efficiently using databases. To further scale up our model, we can use approximate counts of the groundings instead of exact counts. The count of the groundings, which are the number of results of an SQL query, can be calculated using cardinality estimation (Ramakrishnan and Gehrke, 2003) techniques from databases.

Parallelism: For learning RDNs, each conditional distribution is learned independently and hence RDN-Boost can be naturally parallelized by learning each distribution on a different machine or core. Even while learning a single tree, parallelism can be exploited by performing the search at every node on a different machine. Since our trees (both in RFGB and RelOCC) partition the examples into disjoint sets and make local scoring decisions at every node, we can search over all possible literals at every node independent of the other nodes (once the parent nodes are fixed).

Scalable Inference: For a model to be usable on a web-scale domain, the inference approach should also be scalable. As mentioned before, we can use databases for efficient evaluation of the trees and thereby scale inference. We can also exploit parallelism for scalable inference. For instance, since we use a set of trees as our model, we can perform inference on different machines using a single tree on each machine (Map step in Map-Reduce) and then merge the predictions from each tree (Reduce step in Map-Reduce). When performing joint inference, multiple iterations of this Map-Reduce step need to be performed, where predictions from previous iteration are used as evidence in the next iteration. Even while performing inference in-memory and on a single machine, we can exploit the model structure for efficient inference, which I present in Section 8.1.4. These techniques can also be used with the database and Map-Reduce.

8.1.3 Adaptable models

Due to the incremental updating of the model in our boosting approach, it can naturally adapt to new examples, new domain knowledge or change in label definition. The challenge in these approaches lies in recognizing when to change the learned model and how much to change it by, given new information from the user.

New examples: In many tasks, the predictions of our model can be directly seen by the user, who can then easily point out any mistakes made by the model. For example in medical domain, our system can be used to suggest drugs based on a patient's symptoms. If the doctor disagrees with our suggestions, he can mark these as incorrect. Similar system can be used in web search to mark irrelevant webpages, relation extraction to mark incorrect relations, etc. The model may also directly ask the user or an expert to mark the examples it is unsure about, to get better training examples (known as active learning).

Our boosting approach can be modified to maintain a list of "representative" examples from the training set. Ideally, at least one example from this set reaches every leaf in each tree to ensure coverage and an example is weighed based on the number of examples that take similar paths. As our model receives more labeled examples from the user, it can add them to the set of representative examples. Once the error on this set increases by ϵ, we can learn a new tree to update the model via functional-gradient boosting. By using a smaller set of representative examples, we can ensure efficient computation of these updates. We can also recompute the set of representative examples once they become too large.

New rules: Rather than providing individual examples, an expert can provide a rule to correct our model and thereby cover a large space of examples. For example, if our model suggests a drug, A that can lead to complications due to another prescribed drug, B; rather than marking A incorrect every time B is prescribed, the doctor can provide this as a rule to our system. We can now add this rule to our current model as another tree. However, since the rule may not always be correct or may be incomplete, we need to use the training examples to calculate the weight of the rule or modify the rule by introducing or dropping literals.

New definition: Over time the definition of the learned concept might change (known as concept drift). For example, riots in Egypt were anomalous in 2012, but by late 2013 were not considered anomalous. To recognize the change in concept, feedback from the user in some manner is needed, otherwise the system will be oblivious to this change. Given new examples or rules from the user, we can use the approaches presented above to update our model. However, we need to relax the requirement of the model fitting to the original training examples, and in certain scenarios, drop nodes from the trees or drop trees completely, if they were learned using the original training examples.

8.1.4 Efficient inference

Our previous learning approaches use relational regression trees to represent the conditional distributions. Similarly, relational probability trees have been used to compactly represent the conditional distributions in RDNs (Neville and Jensen, 2007). But the inference procedures for relational models do not take advantage of the context-specific independence captured by such trees. Inference procedures such as Gibbs sampling (Bishop, 2006) and belief propagation (Koller and Friedman, 2009) do not leverage the structure of the probability distributions, but instead use them like a "black box". One potential extension of our work is to leverage the tree structure of the distributions to perform faster inference.

Belief propagation (BP; Koller and Friedman 2009) is an approximate inference (exact when the graph has no loops) for graphical models. To perform belief propagation in relational models, the simplest approach is to create the ground factor graph. The ground factor graph uses all the groundings of all the predicates as the variable nodes, which would result in a large factor graph. Performing inference on such a large graph can be computationally expensive. Hence *lifted* inference techniques (Kersting et al., 2009; Singla and Domingos, 2008) have been developed that used a compact *lifted* network for inference. In these lifted approaches, similar variables as well as factors are clustered together in one node, thereby reducing the size of the graph and the number of messages.

In our models, we can begin with a lifted factor graph where each variable node is a first-order predicate and each factor is a relational probability tree. Given this lifted factor graph, a message from a variable node needs to communicate the belief states about all groundings of the predicate corresponding to that variable node. To leverage the tree structure, rather than using a table to represent these messages, one can use a relational tree representation, which compactly represents these messages. Since belief propagation performs sums and product operations over the messages, we need to define these operations for tree-structured messages too. Approaches defined for first-order decision diagrams (FODD; Joshi et al. 2011) can be potentially used for relational regression trees.

8.2 Final Wrap-up

In this thesis, I presented my contributions to the field of learning in Statistical Relational models, specifically in structure learning. My work demonstrates that functional-gradient boosting can be used to efficiently learn the structure of SRL models, which can be more accurate and have a shorter learning time than even parameter-learning approaches. I show my work can be used to learn the structure in presence of missing data as well as missing examples classes. Since learning structure requires minimal effort from the expert, my work can further the applicability of relational models, which I also demonstrate by applying it on diverse tasks.

Aho, A. V., and M. J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Communications of the ACM* 18(6):333–340.

Anderson, G., and B. Pfahringer. 2008. Exploiting propositionalization based on random relational rules for semi-supervised learning. In Proc. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.

Bell, R., Y. Koren, and C. Volinsky. 2007. The Bellkor solution to the Netflix prize. Technical Report, AT&T Labs.

Biba, M., S. Ferilli, and F. Esposito. 2008a. Discriminative structure learning of Markov Logic Networks. In Proc. *Inductive Logic Programming*.

———. 2008b. Structure learning of Markov logic networks through iterated local search. In Proc. *European Conference on Artificial Intelligence*.

Bilenko, M., and R. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. In Proc. *Knowledge Discovery and Data Mining*.

Bishop, C. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc.

Blockeel, H., and L. De Raedt. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101:285–297.

Boutilier, C., N. Friedman, M. Goldszmidt, and D. Koller. 1996. Context-specific independence in Bayesian Networks. In Proc. *Uncertainty in Artificial Intelligence*.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24:123–140.

Chambers, N., and D. Jurafsky. 2008. Jointly combining implicit constraints improves temporal ordering. In Proc. *Empirical Methods in Natural Language Processing*.

Chan, P., and S. Stolfo. 1998. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In Proc. *Knowledge Discovery and Data Mining*.

Chandola, V., A. Banerjee, and V. Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys* 41(3):1–58.

Chickering, D. 1996. Learning Bayesian networks is NP-Complete. In *Learning from Data: Artificial Intelligence and Statistics V*, 121–130. Springer-Verlag.

Comité, F., F. Denis, R. Gilleron, and F. Letouzey. 1999. Positive and unlabeled examples help learning. In Proc. *Algorithmic Learning Theory*.

Craven, M., D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. 1998. Learning to extract symbolic knowledge from the World Wide Web. In Proc. *Association for the Advancement of Artificial Intelligence Conference*.

Craven, M., and J. Shavlik. 1996. Extracting tree-structured representations of trained networks. In Proc. *Neural Information Processing Systems*.

Cussens, J. 1998. Using prior probabilities and density estimation for relational classification. In Proc. *Inductive Logic Programming*.

D'Amato, C., S. Staab, and N. Fanizzi. 2008. On the influence of description logics ontologies on conceptual similarity. In Proc. *Knowledge Engineering and Knowledge Management*.

Davis, J., and M. Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In Proc. *International Conference on Machine Learning*.

Davis, J., I. Ong, J. Struyf, E. Burnside, D. Page, and V.S. Costa. 2007. Change of representation for statistical relational learning. In Proc. *International Joint Conference on Artificial Intelligence*.

Dempster, A., N. Laird, and D. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* B.39:1–38.

Domingos, P., and D. Lowd. 2009. *Markov Logic: An Interface Layer for AI*. Morgan & Claypool.

Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty. 2010. Building Watson: An overview of the DeepQA Project. *AI Magazine* 31(3): 59.

Freund, Y., and R. Schapire. 1996. Experiments with a new boosting algorithm. In Proc. *International Conference on Machine Learning*.

Friedman, J. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 1189–1232.

Friedman, N. 1998. The Bayesian structural EM algorithm. In Proc. *Uncertainty in Artificial Intelligence*.

Friedman, N., D. Geiger, and M. Goldszmidt. 1997. Bayesian network classifiers. *Machine Learning* 29(2-3):131–163.

Galar, Mikel, Alberto Fernández, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. 2011. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition* 44:1761–1776.

Getoor, L., N. Friedman, D. Koller, and A. Pfeffer. 2001. Learning probabilistic relational models. *Relational Data Mining* 307–338.

Getoor, L., and B. Taskar, eds. 2007. *Introduction to Statistical Relational Learning*. MIT Press.

Haddawy, P. 1994. Generating Bayesian networks from probability logic knowledge bases. In Proc. *Uncertainty in Artificial Intelligence*.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorer Newsletter* 11(1):10–18.

Hastie, T., and R. Tibshirani. 1998. Classification by pairwise coupling. In Proc. *Neural Information Processing Systems*.

Hastie, T., R. Tibshirani, and J. Friedman. 2001. *The Elements of Statistical Learning*. Springer.

Heckerman, D., and J. Breese. 1994. A new look at causal independence. In Proc. *Uncertainty in Artificial Intelligence*.

Heckerman, D., D. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. 2001. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research* 1:49–75.

Horváth, T., S. Wrobel, and U. Bohnebeck. 2001. Relational instance-based learning with lists and terms. *Machine Learning* 43:53–80.

Hull, D., and S. Robertson. 1999. The TREC–8 filtering track final report. In Proc. *Text REtrieval Conference*.

Huynh, T., and R. Mooney. 2009. Max-margin weight learning for Markov logic networks. In Proc. *European Conference on Machine Learning*.

———. 2011. Online max-margin weight learning for Markov logic networks. In Proc. *SIAM International Conference on Data Mining*.

Huynh, T. N., and R. J. Mooney. 2008. Discriminative structure and parameter learning for Markov logic networks. In Proc. *International Conference on Machine Learning*.

Jaeger, M. 1997. Relational Bayesian networks. In Proc. *Uncertainty in Artificial Intelligence*.

———. 2007. Parameter learning for relational Bayesian networks. In Proc. *International Conference on Machine Learning*.

———. 2008. Model-theoretic expressivity analysis. In Proc. *Probabilistic Inductive Logic Programming*.

Jank, W. 2005. Stochastic variants of the EM algorithm: Monte Carlo, quasi-Monte Carlo and more. In Proc. *American Statistical Association*.

Jensen, D., and J. Neville. 2002. Linkage and autocorrelation cause feature selection bias in relational learning. In Proc. *International Conference on Machine Learning*.

Ji, H., R. Grishman, H. Dang, and K. Griffit. 2010. An overview of the TAC 2010 Knowledge Base Population track. In Proc. *Text Analysis Conference*.

Joshi, S., K. Kersting, and R. Khardon. 2011. Decision-theoretic planning with generalized first-order decision diagrams. *Artificial Intelligence* 175(18):2198–2222.

Kersting, K., B. Ahmadi, and S. Natarajan. 2009. Counting Belief Propagation. In Proc. *Uncertainty in Artificial Intelligence*.

Kersting, K., and L. De Raedt. 2007. Bayesian logic programming: Theory and tool. In Proc. *An Introduction to Statistical Relational Learning*, ed. L. Getoor and B. Taskar.

Kersting, K., and T. Raiko. 2005. ‘Say EM’ for selecting probabilistic models for logical sequences. In Proc. *Uncertainty in Artificial Intelligence*.

Khan, S., and M. Madden. 2009. A survey of recent trends in one class classification. In Proc. *Irish Conference on Artificial Intelligence and Cognitive Science*.

Khot, T., S. Natarajan, K. Kersting, and J. Shavlik. 2011. Learning Markov logic networks via functional gradient boosting. In Proc. *IEEE International Conference on Data Mining*.

———. 2013a. Learning relational probabilistic models from partially observed data - opening the closed-world assumption. In Proc. *Inductive Logic Programming*. Accepted.

———. 2014a. Gradient-based boosting for statistical relational learning : The Markov logic network and missing data cases. *Machine Learning*. Under review.

Khot, T., S. Natarajan, and J. Shavlik. 2014b. Relational one-class classification: A non-parametric approach. In Proc. *Association for the Advancement of Artificial Intelligence Conference*.

Khot, T., S. Srivastava, S. Natarajan, and J. Shavlik. 2012. Learning relational structure for temporal relation extraction. In Proc. *Statistical Relational AI Workshop*.

Khot, T., C. Zhang, S. Natarajan, C. Ré, and J. Shavlik. 2013b. Bootstrapping knowledge base acceleration. In Proc. *Text REtrieval Conference*.

Kim, J., T. Ohta, S. Pyysalo, Y. Kano, and J. Tsujii. 2009. Overview of BioNLP'09 shared task on event extraction. In Proc. *Biomedical Natural Language Processing Workshop*.

Kindermann, R., and J. Snell. 1980. *Markov random fields and their applications*. American Mathematical Society.

Kok, S., and P. Domingos. 2005. Learning the structure of Markov logic networks. In Proc. *International Conference on Machine Learning*.

———. 2009. Learning Markov logic network structure via hypergraph lifting. In Proc. *International Conference on Machine Learning*.

———. 2010. Learning Markov logic networks using structural motifs. In Proc. *International Conference on Machine Learning*.

Kok, S., M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, J. Wang, A. Nath, and P. Domingos. 2010. The Alchemy system for statistical relational AI. Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. Http://alchemy.cs.washington.edu.

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

Koller, D., and A. Pfeffer. 1997a. Learning probabilities for noisy first-order rules. In Proc. *International Joint Conference on Artificial Intelligence*.

———. 1997b. Object-oriented Bayesian networks. In Proc. *Uncertainty in Artificial Intelligence*.

Kononenko, I. 2001. Machine learning for medical diagnosis: History, state of the art and perspective. *Artificial Intelligence in Medicine* 23:89–109.

Kosala, R., and H. Blockeel. 2000. Web mining research: A survey. *ACM SIGKDD Exploration Newsletter* 2(1):1–15.

Landwehr, N., A. Passerini, L. De Raedt, and P. Frasconi. 2010. Fast learning of relational kernels. *Machine Learning* 78:305–342.

Lawrence, S., C. Giles, and K. Bollacker. 1999. Autonomous citation matching. In Proc. *AGENTS*.

Li, X., and Z. Zhou. 2007. Structure learning of probabilistic relational models from incomplete relational data. In Proc. *European Conference on Machine Learning*.

Liu, F., K. Ting, and Z. Zhou. 2012. Isolation-based anomaly detection. *Transactions on Knowledge Discovery from Data* 6(1):3.

Lowd, D., and P. Domingos. 2007. Efficient weight learning for Markov logic networks. In Proc. *Principles and Practice of Knowledge Discovery in Databases*.

Manning, C., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

McCallum, A., K. Nigam, J. Rennie, and K. Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3(2):127–163.

Mihalkova, L., and R. Mooney. 2007. Bottom-up learning of Markov logic network structure. In Proc. *International Conference on Machine Learning*.

Mintz, M., S. Bills, R. Snow, and D. Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In Proc. *Association for Computational Linguistics*.

Mitchell, T. 1997. *Machine Learning*. McGraw-Hill, Inc.

Moya, M., and D. Hush. 1996. Network constraints and multi-objective optimization for one-class classification. *Neural Networks* 9(3):463–474.

Muggleton, S. 1997. Learning from positive data. In Proc. *Inductive Logic Programming*.

Muggleton, S., and L. De Raedt. 1994. Inductive logic programming: Theory and methods. *Journal of Logic Programming* 19/20:629–679.

Muggleton, Stephen. 1996. Stochastic logic programs. In Proc. *Advances in Inductive Logic Programming*.

Natarajan, S., and E. Altendorf. 2005. First order conditional influence language. Technical Report, School of EECS, Oregon State University, USA.

Natarajan, S., S. Joshi, P. Tadepalli, K. Kristian, and J. Shavlik. 2011. Imitation learning in relational domains: A functional-gradient boosting approach. In Proc. *International Joint Conference on Artificial Intelligence*.

Natarajan, S., T. Khot, K. Kersting, B. Guttmann, and J. Shavlik. 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*.

Natarajan, S., T. Khot, D. Lowd, P. Tadepalli, K. Kersting, and J. Shavlik. 2010. Exploiting causal independence in Markov logic networks: Combining undirected and directed models. In Proc. *European Conference on Machine Learning*.

Natarajan, S., B. Saha, S. Joshi, A. Edwards, T. Khot, E. Davenport, K. Kersting, C. Whitlow, and J. Maldjian. 2013. Relational learning helps in three-way classification of Alzheimer patients from structural magnetic resonance images of the brain. *International Journal of Machine Learning and Cybernetics* 1–11.

Natarajan, S., P. Tadepalli, T. Dietterich, and A. Fern. 2008. Learning first-order probabilistic models with combining rules. *Annals of Mathematics and AI* 54(1-3):223–256.

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

Neville, J., and D. Jensen. 2007. Relational dependency networks. In *Introduction to Statistical Relational Learning*, ed. L. Getoor and B. Taskar, 653–692. MIT Press.

Neville, J., D. Jensen, L. Friedland, and M. Hay. 2003a. Learning relational probability trees. In Proc. *Knowledge Discovery and Data Mining*.

Neville, J., D. Jensen, and B. Gallagher. 2003b. Simple estimators for relational Bayesian classifiers. In Proc. *IEEE International Conference on Data Mining*.

Ngo, L., and P. Haddawy. 1996. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science* 171:147–177.

Niculescu-Mizil, A., and R. Caruana. 2005. Obtaining calibrated probabilities from boosting. In Proc. *Uncertainty in Artificial Intelligence*.

Nilsson, N. 1986. Probabilistic logic. *Artificial Intelligence* 28(1):71–88.

Niu, F., C. Zhang, C. Ré, and J. Shavlik. 2012a. Scaling inference for Markov logic via dual decomposition. In Proc. *IEEE International Conference on Data Mining*.

Niu, Feng, Ce Zhang, Christopher Ré, and Jude Shavlik. 2012b. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *IJSWIS Special Issue on Web-Scale Knowledge Extraction*.

Parzen, E. 1962. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics* 33(3):1065–1076.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc.

Platt, J. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Proc. *Advances in Large Margin Classifiers*.

Poole, D. 1993. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence Journal* 81–129.

Poon, H., and P. Domingos. 2007. Joint inference in information extraction. In Proc. *Association for the Advancement of Artificial Intelligence Conference*.

Pustejovsky, J., J. Castaño, R. Ingria, R. Saurí, R. Gaizauskas, A. Setzer, and G. Katz. 2003a. TimeML: Robust specification of event and temporal expressions in text. In Proc. *International Workshop on Computational Semantics*.

Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, and M. Lazo. 2003b. The TIMEBANK corpus. In Proc. *Corpus Linguistics*.

Raedt, L. De, A. Kimmig, and H. Toivonen. 2007. Problog: A probabilistic Prolog and its application in link discovery. In Proc. *International Joint Conference on Artificial Intelligence*.

Ramakrishnan, R., and J. Gehrke. 2003. *Database management systems*. McGraw-Hill Inc.

Ratliff, N., A. Bagnell, and M. Zinkevich. 2006. Maximum margin planning. In Proc. *International Conference on Machine Learning*.

Richardson, M., and P. Domingos. 2004. Markov logic networks. Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. Http://www.cs.washington.edu/homes/pedrod/mln.pdf.

———. 2006. Markov logic networks. *Machine Learning* 62:107–136.

de Ridder, D., D. Tax, and R. Duin. 1998. An experimental comparison of one-class classification methods. In Proc. *Conference of the Advanced School for Computing and Imaging*.

Russell, S., and P. Norvig. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.

Schölkopf, B., R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. 2000. Support vector method for novelty detection. In Proc. *Neural Information Processing Systems*.

Settles, B. 2012. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool.

Shavlik, J., and S. Natarajan. 2009. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In Proc. *International Joint Conference on Artificial Intelligence*.

Singla, P., and P. Domingos. 2005. Discriminative training of Markov logic networks. In Proc. *Association for the Advancement of Artificial Intelligence Conference*.

———. 2006. Entity resolution with Markov logic. In Proc. *IEEE International Conference on Data Mining*.

———. 2007. Markov logic in infinite domains. In Proc. *Uncertainty in Artificial Intelligence*.

———. 2008. Lifted first-order belief propagation. In Proc. *Association for the Advancement of Artificial Intelligence Conference*.

Srinivasan, A. 2004. *The Aleph Manual*.

Sun, L., R. Patel, J. Liu, K. Chen, T. Wu, J. Li, E. Reiman, and J. Ye. 2009. Mining brain region connectivity for Alzheimer's disease study via sparse inverse covariance estimation. In Proc. *Knowledge Discovery and Data Mining*.

Supekar, K., V. Menon, D. Rubin, M. Musen, and M. Greicius. 2008. Network analysis of intrinsic functional brain connectivity in Alzheimer's disease. *PLoS Computational Biology* 4(6).

Taskar, B., P. Abeel, and D. Koller. 2002. Discriminative probabilistic models for relational data. In Proc. *Uncertainty in Artificial Intelligence*.

Tax, D., and R. Duin. 1999. Support vector domain description. *Pattern Recognition Letters* 20:1191–1199.

UzZaman, N., and J. F. Allen. 2010. TRIPS and TRIOS system for TempEval-2: Extracting temporal information from text. In Proc. *International Workshop on Semantic Evaluation*.

UzZaman, N., H. Llorens, J. Allen, L. Derczynski, M. Verhagen, and J. Pustejovsky. 2012. Tempeval-3: Evaluating events, time expressions, and temporal relations. *CoRR* abs/1206.5333.

Van Laer, W., L. Dehaspe, and L. De Raedt. 1994. Applications of a logical discovery engine. In Proc. *Inductive Logic Programming*.

Verhagen, M., R. Gaizauskas, F. Schilder, M. Hepple, G. Katz, and J. Pustejovsky. 2007. SemEval-2007 Task 15: TempEval temporal relation identification. In Proc. *International Workshop on Semantic Evaluation*.

Verhagen, M., R. Sauri, T. Caselli, and J. Pustejovsky. 2010. SemEval-2010 Task 13: Tempeval-2. In Proc. *International Workshop on Semantic Evaluation*.

Viola, P., and M. Jones. 2001. Rapid object detection using a boosted cascade of simple features. In Proc. *Conference on Computer Vision and Pattern Recognition*.

Wei, G., and M. Tanner. 1990. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association* 85(411):699–704.

Wellman, M., J. Breese, and R. Goldman. 1992. From knowledge bases to decision models. *The Knowledge Engineering Review* 7:35–53.

Xu, Z., K. Kersting, and V. Tresp. 2009. Multi–relational learning with Gaussian Processes. In Proc. *International Joint Conference on Artificial Intelligence*.

Ye, J., K. Chen, T. Wu, J. Li, Z. Zhao, R. Patel, M. Bae, R. Janardan, H. Liu, G. Alexander, and E. Reiman. 2008. Heterogeneous data fusion for Alzheimer's disease study. *Knowledge Discovery and Data Mining*.

Yoshikawa, K., S. Riedel, M. Asahara, and Y. Matsumoto. 2009. Jointly identifying temporal relations with Markov logic. In Proc. *Association for Computational Linguistics*.

Zadrozny, B., and C. Elkan. 2002. Transforming classifier scores into accurate multiclass probability estimates. In Proc. *Knowledge Discovery and Data Mining*.

Zhang, C., and C. Ré. 2013. Towards high-throughput Gibbs sampling at scale: A study across storage managers. In Proc. *Special Interest Group on Management Of Data*.

Zhang, N., and D. Poole. 1996. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research* 5: 301–328.

Zhu, X., and A. Goldberg. 2009. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers.