

Chapter 2

Statistical Relational Learning

This chapter presents background on SRL models on which our work is based on. We start with a brief technical background on first-order logic and graphical models. In Sect. 2.2, we present an overview of SRL models followed by details on two popular SRL models. We then present the learning challenges in these models and the approaches taken to solve them in literature. In Sect. 2.3.3, we present functional-gradient boosting, an ensemble approach¹, which forms the basis of our learning approaches. Finally, We present details about the evaluation metrics and datasets we used.

2.1 Representing Structure and Uncertainty

We first define some notation that is used throughout this book. We 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. We use bold-faced letters to represents sets. Letters such as $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ represent sets of variables and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ represent sets of values. We 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 $p(X)$) which can be used to make logical statements about all objects in the domain (Russell and Norvig 2003). To avoid confusion with random variables, We use sans-serif capital letters X, Y , and Z

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

Table 2.1 First-order logic terminology

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

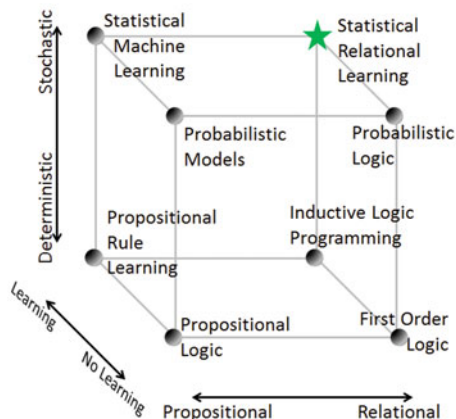
to represent logical variables. We use lower-case sans-serif letters such as *x*, *y* and *z* to represent values taken by logical variables i.e. objects in the domain. The common logical operators/quantifiers are \wedge (AND), \vee (OR), \Rightarrow (implication—condition implies consequence), \forall (for all) and \exists (existential 0 true for at least one value). used in this book are:

Furthermore, we assume that all logical variables are implicitly universally quantified (i.e. \forall) unless explicitly existentially quantified. Table 2.1 presents sample definitions of first-order logic terms that are used in the book.

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. 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 generally represented using the function ϕ and the normalization term using Z . Directed models such as Bayesian networks (Pearl 1988) represent the joint

Fig. 2.1 Different areas of AI with respect to learning, representation and uncertainty



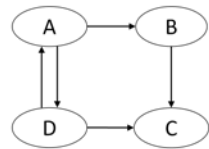
distributions as a product of conditional distributions for each variable given the parents of the variable. To ensure the product of conditional distributions represents the joint distribution, directed models require the model to be acyclic. The key idea is that the factored distributions can have exponentially smaller number of parameters compared to the full joint distribution.

Dependency networks (Heckerman et al. 2001) are directed graphical models that remove this acyclicity condition thereby allowing for faster learning of these models. But the product of the conditional distributions may not produce a consistent joint distribution.

2.2 Statistical Relational Models

Statistical Relational Learning (SRL) (Getoor and Taskar 2007) addresses the challenge of applying statistical inference and learning approaches to problems which involve rich collections of objects linked together in a complex, stochastic, and relational world. Figure 2.1 presents different areas in AI with respect to three dimensions: learning, representation and uncertainty. As can be observed, SRL extends statistical machine learning (Mitchell 1997) by using a richer representation, extends inductive logic programming (Lavrac and Dzeroski 1994; Raedt 2008) by modeling uncertainty and extends probabilistic logic (Nilsson 1986) by employing learning algorithms. The advantage of SRL models (Getoor et al. 2001; Heckerman et al. 2004; Jaeger 1997; Kersting and De Raedt 2007; Milch et al. 2004; Neville and Jensen 2007; Ngo and Haddawy 1995; Poole 1993; Sato and Kameya 2001; Domingos and Lowd 2009; Gutmann and Kersting 2006; Taskar et al. 2002) is that they can succinctly represent probabilistic dependencies among the attributes of different related objects, leading to a compact representation of learned models. We present two SRL

Fig. 2.2 A dependency network



models for which we will show how to use boosting to learn them automatically from data: one directed and one undirected relational model.

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) ($P(\mathbf{X}) \approx \prod_i P(X_i | Pa(X_i))$). Unlike Bayesian Networks, DNs allow cycles in the graphical model, as a result the joint distribution is approximated. The key advantage of this approximation is that each conditional distribution can be learned independently, which makes learning DNs much faster. Figure 2.2 shows a sample DN where $P(A, B, C, D)$ is approximated by the product $P(B|A)P(C|B, D)P(D|A)P(A|D)$. While these are approximate models, 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.

Relational Dependency Networks (RDNs) are relational extensions 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 Fig. 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 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 satisfaction of a student.

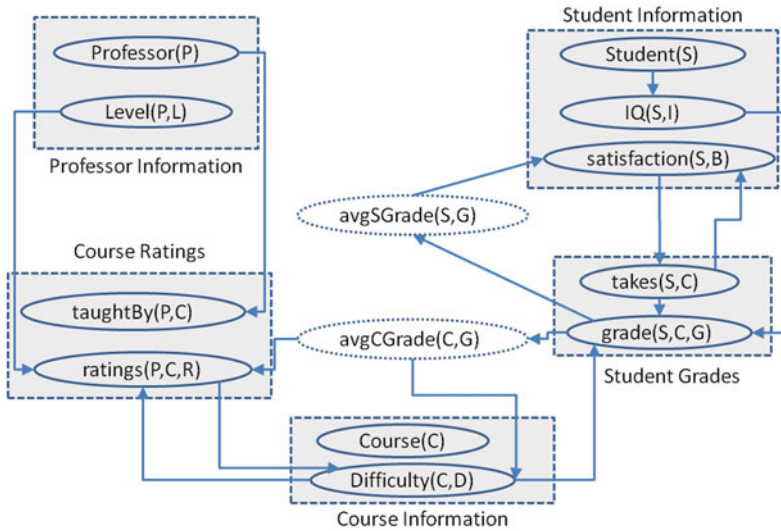


Fig. 2.3 A relational dependency network

2.2.2 Markov Logic Networks

Markov Logic Networks (MLNs) (Domingos and Lowd 2009) 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 constants² in the first-order logic rules. Each rule r_i forms a clique in the ground network and its weight w_i determine the potential for each clique. Fig. 2.4.

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

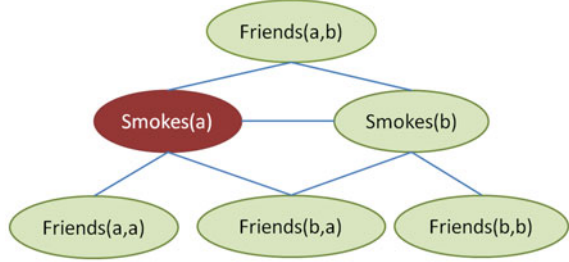
This rule presents a MLN clause from a simple cancer domain (adopted from Domingos and Lowd 2009). The corresponding ground Markov network generated from this rule for a domain with two constants $X, Y \in \{a, b\}$ is shown in Fig. 2.4.

The joint probability distribution in a MLN is given by the product of the potentials on each clique, similar to Markov networks. 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:

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

² We assume a finite set of constants throughout this document.

Fig. 2.4 A ground Markov network for the friends and smokes rule 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



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 Fig. 2.4, 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. For example in the sample MLN, the *Friends* predicate would have $O(n^2)$ groundings, where n is the number of people in the data set. Computation of the normalization terms requires summing over all these groundings. As a result, most learning methods use the approximate 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 blanket³ of the ground atom, X_i in the ground Markov network. An avid reader must have noted that the pseudo-likelihood term is similar to the probability distribution of RDNs, which is also defined as a product of the conditional distributions.

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 quantify this relationship.

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.

³ The Markov blanket of a node x_i is all the direct neighbors of x_i in the ground Markov network.

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. The standard approach in this research is to formulate the optimization problem as either maximizing the log-likelihood or minimizing the mean squared error when given some training data (Natarajan et al. 2005; Natarajan et al. 2008; Kersting and De Raedt 2007; Getoor et al. 2001). Then either gradient-descent or EM algorithm is adapted to fill in the parameter values. Since relational models can have multiple instantiations of a logical variable, either combining rules or aggregations are used to handle this issue. The algorithms are capable of learning the parameters of these combining rules as well.

Undirected Models

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 are computed. The issue is that this gradient computation requires computation of the expected number of groundings of the clause which in turn requires inference at each step. So approximations such as maximum a posteriori (MAP) estimates were used instead of the actual expectation. This work was later extended to second-order gradient descent approach (Lowd and Domingos 2007) and to margin-based approaches (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.

Most early structure learning methods employed the use of Inductive Logic Programming (ILP) (Muggleton and Raedt 1994) that learned logical theories such that these theories cover most of the positive examples and as few as possible of a set of negative examples. They perform a greedy search for the set of “clauses” that are

consistent with the training examples. Needless to say that these are deterministic clauses which do not handle noisy data well. However, these are the early techniques that inspired several structure learning methods inside SRL. One successful adaptation of this inductive learning method is the learning of TILDE trees (Blockeel and Raedt 1998). These trees upgrade the attribute representation of the classical decision trees by allowing for logical clauses as tests inside each node. Hence, learning a decision tree can be understood as learning a decision list of first-order clauses using some ILP learning method. TILDE trees extend this learning by computing regression values (or probabilities) in the leaves of these trees. This is one of the earliest successful statistical relational structure learning method. We now present the other more recent methods briefly.

Directed Models

Most directed models such as BLPs and PRMs 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/removing a parent from this set during the greedy search. The commonality is that 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.

The simplest among all directed models is the case of local models such as RDNs where learning the set of conditional distributions independently is sufficient to learn the structure. Neville and Jensen (2007) use two models: Relational Probability Trees (RPT; Neville et al. 2003a) and Relational Bayesian Classifiers (RBC; Neville et al. 2003b) to model the conditional distributions. A simple way to understand this learning is that at the start of learning process, all the other predicates are assumed to be parents of the target predicate. Then learning corresponds to simply finding the best tree that models the conditional distribution (similar to feature selection in a decision tree for propositional machine learning). Hence, an RDN is a set of RPTs learned one after another.

Undirected Models

For undirected models, since most of the prior work as well as our work focuses on MLNs, we 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. Bottom-up structure learning (Mihalkova and Mooney 2007) uses a propositional Markov network learning algorithm to identify paths of ground atoms. These form the templates that are generalized into first-order formulas. Hypergraph lifting (Kok and Domingos 2009) on the other hand clusters the constants and true atoms to construct a lifted (first-order) graph. Relational path-finding on this hypergraph is used to obtain the MLN clauses. Structural motif learning (Kok and Domingos 2010) uses random walks on the ground network to find symmetrical paths and cluster nodes with similar path distributions. All these methods obtain the candidate clauses first, learn the weights and modify the clauses.

2.3.3 Functional-Gradient Boosting

Recall from the introduction that the goal of this book is to bring the complexity of structure learning as close to parameter learning as possible. To this effect, we now present a learning algorithm that in the propositional case can learn the parameter and structure of the model simultaneously.

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,

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

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

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

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

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:

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

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.

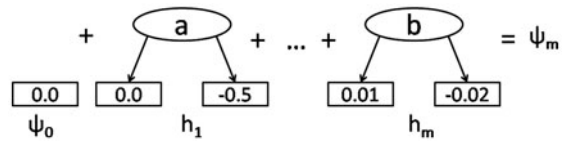
Consider the small dataset with three examples shown in Table 2.2 with two features (a and b) and the class label c . If we assume $P(y_i = 1|x_i; \psi) = 1/(1 + e^{-\psi(x_i)})$, the gradients can be shown to be $\Delta_m(x) = We(y_i = \hat{y}_i) - P(y_i = \hat{y}_i|x_i; \psi_m)$, where $We(y_i = \hat{y}_i)$ 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.

Table 2.3 Initial regression dataset

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

Table 2.4 Regression values after tree 1

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

Fig. 2.5 A sample model for predicting class label c after m iterations

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:

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

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 2.5 shows a sample model ψ_m after m iterations of boosting.

2.4 Benchmark Datasets

We now present details about the evaluation approach used. To show that the boosting 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. In addition to conditional log-likelihood (CLL), we use the areas under Precision-Recall curve (AUC-PR) and Receiver Operating Characteristic curve (AUC-ROC) (Davis and Goadrich 2006). It has been shown that CLL is not a perfect measure when dealing with skewed data sets. Relational data is inherently skewed as most of the relations are generally false (the number of friends of a person is much smaller than the entire population). In such cases, AUC-PR and AUC-ROC are considered better evaluation metrics.

Most of our learning approaches are evaluated on standard SRL data sets. We will present details about three of the most commonly used data sets in literature that are also used in our work. For other experiments, we refer to the corresponding paper.

2.4.1 UW-CSE

The UW-CSE dataset (Richardson and Domingos 2006) was created 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 dataset includes predicates such as *professor*, *student*, *publication*, *advisedBy*, *hasPosition*, *projectMember*, *yearsInProgram*, *courseLevel*, *taughtBy*, and *teachingAssistant* and equality predicates such as *samePerson*, *sameCourse* etc. The goal in this data set 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. Unless specified, we train on four areas and evaluated the results on the remaining area. This is the same approach taken in the MLN literature (Domingos and Lowd 2009) where each of the four areas form a “mega-example” that consists of all the inter-related objects of that area. Creating more folds would require breaking up the network of connected objects within each area. Hence each area is viewed as a single example. Our results are thus averaged over five mega-examples.

2.4.2 Cora

Cora dataset, now a standard dataset for citation matching, 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. A cluster is a set of citations that refer to the same paper, and a nontrivial cluster contains more than one citation. The Cora dataset has 1295 citations and 134 clusters where almost every citation in Cora belongs to a nontrivial cluster; the largest cluster contains 54 citations. Sets of clusters were combined to create five mega-examples by Poon and Domingos.

For each citation we have information about the various fields using predicates such as *author*, *title*, *venue*, *hasWordAuthor*, *hasWordTitle*, and *hasWordVenue*. This task has multiple target predicates (*sameAuthor*, *sameVenue*, *sameTitle*, and *sameBib*) for identifying matching authors, venues, titles and the complete citation. This dataset has five mega-examples and hence We perform 5-fold cross-validation to evaluate on this dataset.

2.4.3 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*, *director*, *workedUnder*, *genre*, and *gender*. The task is to predict the *workedUnder*, *genre*, and *gender* given all the other predicates. The *actor* and *director* predicates are mutually exclusive predicates (i.e., $actor(X) \Leftrightarrow \neg director(X)$) that provide type information for the people in the domain. Since the dataset is divided to five mega-examples by Mihalkova and Mooney (each mega-example contains four movies), we perform five-fold cross-validation in the experiments. Following Kok and Domingos (2009), we omitted the four equality predicates. The goal is to learn the conditional distribution to predict all the predicates except *actor* and *director*.

Boosted Statistical Relational Learners

From Benchmarks to Data-Driven Medicine

Natarajan, S.; Kersting, K.; Khot, T.; Shavlik, J.

2014, VIII, 74 p. 25 illus., Softcover

ISBN: 978-3-319-13643-1