MORGAN & CLAYPOOL PUBLISHERS

Markov Logic

An Interface Layer for Artificial Intelligence

Pedro Domingos
Daniel Lowd

SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Ronald J. Brachman and Thomas G. Dietterich, *Series Editors*

Markov Logic: An Interface Layer for Artificial Intelligence

Markov Logic: An Interface Layer for Artificial Intelligence

Pedro Domingos and Daniel Lowd

www.morganclaypool.com

ISBN: 9781598296921 paperback
ISBN: 9781598296938 ebook

DOI 10.2200/S00206ED1V01Y200907AIM007

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Lecture #7
Series Editors: Ronald J. Brachman, *Yahoo! Research*
Thomas Dietterich, *Oregon State University*

Series ISSN
Synthesis Lectures on Artificial Intelligence and Machine Learning
Print 1939-4608 Electronic 1939-4616

Contents

Acknowledgments

We are grateful to all the people who contributed to the development of Markov logic and Alchemy: colleagues, users, developers, reviewers, and others. We thank our families for their patience and support.

The research described in this book was partly funded by ARO grant W911NF-08-1-0242, DARPA contracts FA8750-05-2-0283, FA8750-07-D-0185, HR0011-06-C-0025, HR0011-07-C-0060 and NBCH-D030010, NSF grants IIS-0534881, IIS-0803481 and EIA-0303609, ONR grants N-00014-05-1-0313 and N00014-08-1-0670, an NSF CAREER Award (first author), a Sloan Research Fellowship (first author), an NSF Graduate Fellowship (second author) and a Microsoft Research Graduate Fellowship (second author). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, DARPA, NSF, ONR, or the United States Government.

Pedro Domingos and Daniel Lowd
Seattle, Washington

CHAPTER 1

Introduction

1.1 THE INTERFACE LAYER

Artificial intelligence (AI) has made tremendous progress in its first 50 years. However, it is still very far from reaching and surpassing human intelligence. At the current rate of progress, the crossover point will not be reached for hundreds of years. We need innovations that will permanently increase the rate of progress. Most research is inevitably incremental, but the size of those increments depends on the paradigms and tools that researchers have available. Improving these provides more than a one-time gain; it enables researchers to consistently produce larger increments of progress at a higher rate. If we do it enough times, we may be able to shorten those hundreds of years to decades, as illustrated in Figure 1.1.

If we look at other subfields of computer science, we see that in most cases progress has been enabled above all by the creation of an interface layer that separates innovation above and below it, while allowing each to benefit from the other. Below the layer, research improves the foundations (or, more pragmatically, the infrastructure); above it, research improves existing applications and invents new ones. Table 1.1 shows examples of interface layers from various subfields of computer science. In each of these fields, the development of the interface layer triggered a period of rapid progress above and below it. In most cases, this progress continues today. For example, new applications enabled by the Internet continue to appear, and protocol extensions continue to be proposed. In many cases, the progress sparked by the new layer resulted in new industries, or in sizable expansions of existing ones.

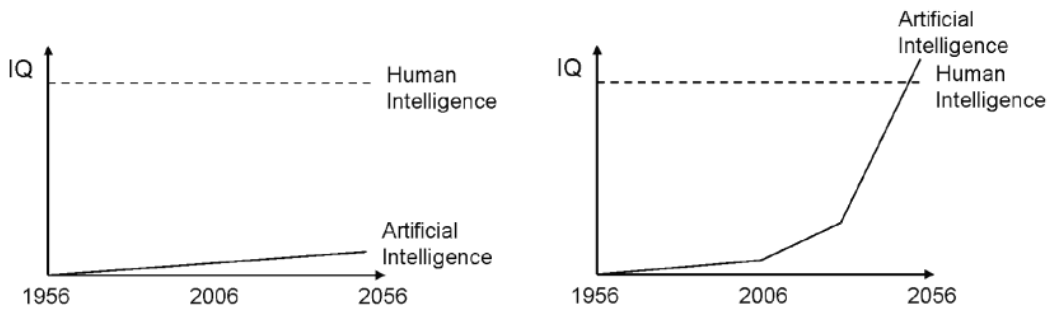

Figure 1.1: The first 100 years of AI. The graph on the right illustrates how increasing our rate of progress several times could bring about human-level intelligence much faster.

Table 1.1: Examples of interface layers.

Field	Interface Layer	Below the Layer	Above the Layer
Hardware	VLSI design	VLSI modules	Computer-aided chip design
Architecture	Microprocessors	ALUs, buses	Compilers, operating systems
Operating systems	Virtual machines	Hardware	Software
Programming systems	High-level languages	Compilers, code optimizers	Programming
Databases	Relational model	Query optimization, transaction mgmt.	Enterprise applications
Networking	Internet	Protocols, routers	Web, email
HCI	Graphical user interface	Widget toolkits	Productivity suites
AI	???	Inference, learning	Planning, NLP, vision, robotics

The interface layer allows each innovation below it to automatically become available to all the applications above it, without the "infrastructure" researchers having to know the details of the applications, or even what all the existing or possible applications are. Conversely, new applications (and improvements to existing ones) can be developed with little or no knowledge of the infrastructure below the layer. Without it, each innovation needs to be separately combined with each other, an $O(n^2)$ problem that in practice is too costly to solve, leaving only a few of the connections made. When the layer is available, each innovation needs to be combined only with the layer itself. Thus, we obtain $O(n^2)$ benefits with $O(n)$ work, as illustrated in Figure 1.2.

In each case, the separation between applications above the layer and infrastructure below it is not perfect. If we care enough about a particular application, we can usually improve it by developing infrastructure specifically for it. However, most applications are served well enough by the general-purpose machinery, and in many cases would not be economically feasible without it. For example, ASICs (application-specific integrated circuits) are far more efficient for their specific applications than general-purpose microprocessors; but the vast majority of applications are still run on the latter. Interface layers are an instance of the 80/20 rule: they allow us to obtain 80% of the benefits for 20% of the cost.

The essential feature of an interface layer is that it provides a language of operations that is all the infrastructure needs to support, and all that the applications need to know about. Designing it is a difficult balancing act between providing what the applications need and staying within what the infrastructure can do. A good interface layer exposes important distinctions and hides unimportant ones. How to do this is often far from obvious. Creating a successful new interface layer is thus not easy. Typically, it initially faces skepticism, because it is less efficient than the existing alternatives, and appears too ambitious, being beset by difficulties that are only resolved by later research. But once the layer becomes established, it enables innovations that were previously unthinkable.

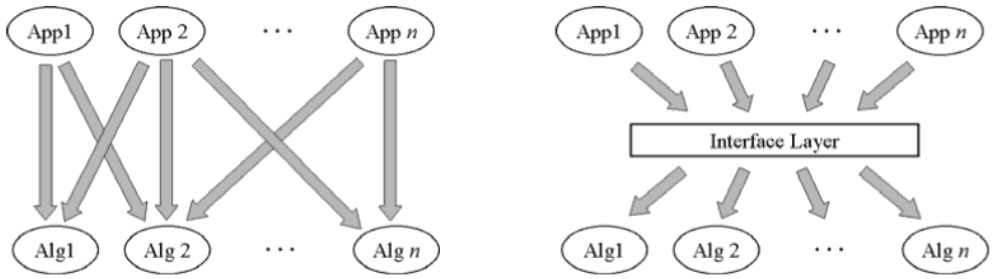

Figure 1.2: An interface layer provides the benefits of $O(n^2)$ connections between applications and infrastructure algorithms with just $O(n)$ connections.

1.2 WHAT IS THE INTERFACE LAYER FOR AI?

In AI, the interface layer has been conspicuously absent, and this, perhaps more than any other factor, has limited the rate of progress. An early candidate for this role was first-order logic. However, it quickly became apparent that first-order logic has many shortcomings as a basis for AI, leading to a long series of efforts to extend it. Unfortunately, none of these extensions has achieved wide acceptance. Perhaps the closest first-order logic has come to providing an interface layer is the Prolog language. However, it remains a niche language even within AI. This is largely because, being essentially a subset of first-order logic, it shares its limitations, and is thus insufficient to support most applications at the 80/20 level.

Many (if not most) of the shortcomings of logic can be overcome by the use of probability. Here, graphical models (i.e., Bayesian and Markov networks) have to some extent played the part of an interface layer, but one with a limited range. Although they provide a unifying language for many different probabilistic models, graphical models can only represent distributions over propositional universes, and are thus insufficiently expressive for general AI. In practice, this limitation is often circumvented by manually transforming the rich relational domain of interest into a simplified propositional one, but the cost, brittleness, and lack of reusability of this manual work is precisely what a good interface layer should avoid. Also, to the extent that graphical models can provide an interface layer, they have done so mostly at the conceptual level. No widely accepted languages or standards for representing and using graphical models exist today. Many toolkits with specialized functionality have been developed, but none that could be used as widely as (say) SQL engines are in databases. Perhaps the most widely used such toolkit is BUGS, but it is quite limited in the learning and inference infrastructure it provides. Although very popular with Bayesian statisticians, it fails the 80/20 test for AI.

It is clear that the AI interface layer needs to integrate first-order logic and graphical models. One or the other by itself cannot provide the minimum functionality needed to support the full range of AI applications. Further, the two need to be fully integrated, and not simply provided alongside each other. Most applications require simultaneously the expressiveness of first-order logic and the

Table 1.2: Examples of logical and statistical AI.

Field	Logical Approach	Statistical Approach
Knowledge representation	First-order logic	Graphical models
Automated reasoning	Satisfiability testing	Markov chain Monte Carlo
Machine learning	Inductive logic programming	Neural networks
Planning	Classical planning	Markov decision processes
Natural language processing	Definite clause grammars	Probabilistic context-free grammars

robustness of probability, not just one or the other. Unfortunately, the split between logical and statistical AI runs very deep. It dates to the earliest days of the field, and continues to be highly visible today. It takes a different form in each subfield of AI, but it is omnipresent. Table 1.2 shows examples of this. In each case, both the logical and the statistical approach contribute something important. This justifies the abundant research on each of them, but also implies that ultimately a combination of the two is required.

In recent years, we have begun to see increasingly frequent attempts to achieve such a combination in each subfield. In knowledge representation, knowledge-based model construction combines logic programming and Bayesian networks, and substantial theoretical work on combining logic and probability has appeared. In automated reasoning, researchers have identified common schemas in satisfiability testing, constraint processing and probabilistic inference. In machine learning, statistical relational learning combines inductive logic programming and statistical learning. In planning, relational MDPs add aspects of classical planning to MDPs. In natural language processing, work on recognizing textual entailment and on learning to map sentences to logical form combines logical and statistical components. However, for the most part these developments have been pursued independently, and have not benefited from each other. This is largely attributable to the $O(n^2)$ problem: researchers in one subfield can connect their work to perhaps one or two others, but connecting it to all of them is not practically feasible.

1.3 MARKOV LOGIC AND ALCHEMY: AN EMERGING SOLUTION

We have recently introduced Markov logic, a language that combines first-order logic and Markov networks. A knowledge base (KB) in Markov logic is a set of first-order formulas with weights. Given a set of constants representing objects in the domain of interest, it defines a probability distribution over possible worlds, each world being an assignment of truth values to all possible ground atoms. The distribution is in the form of a log-linear model: a normalized exponentiated weighted combination of features of the world.[1] Each feature is a grounding of a formula in the KB, with the corresponding weight. In first-order logic, formulas are hard constraints: a world that violates even a single formula

[1] Log-linear models are also known as, or closely related to, Markov networks, Markov random fields, maximum entropy models, Gibbs distributions, and exponential models; and they have Bayesian networks, Boltzmann machines, conditional random fields, and logistic regression as special cases.

Table 1.3: A comparison of Alchemy, Prolog, and BUGS.

Aspect	Alchemy	Prolog	BUGS
Representation	First-order logic + Markov networks	Horn clauses	Bayesian networks
Inference	Model checking, MCMC, lifted BP	Theorem proving	Gibbs sampling
Learning	Parameters and structure	No	Parameters
Uncertainty	Yes	No	Yes
Relational	Yes	Yes	No

is impossible. In Markov logic, formulas are soft constraints: a world that violates a formula is less probable than one that satisfies it, other things being equal, but not impossible. The weight of a formula represents its strength as a constraint. Finite first-order logic is the limit of Markov logic when all weights tend to infinity. Markov logic allows an existing first-order KB to be transformed into a probabilistic model simply by assigning weights to the formulas, manually or by learning them from data. It allows multiple KBs to be merged without resolving their inconsistencies, and obviates the need to exhaustively specify the conditions under which a formula can be applied. On the statistical side, it allows very complex models to be represented very compactly; in particular, it provides an elegant language for expressing non-i.i.d. models (i.e., models where data points are not assumed independent and identically distributed). It also facilitates the incorporation of rich domain knowledge, reducing reliance on purely empirical learning.

Markov logic builds on previous developments in knowledge-based model construction and statistical relational learning, but goes beyond them in combining first-order logic and graphical models without restrictions. Unlike previous representations, it is supported by a full range of learning and inference algorithms, in each case combining logical and statistical elements. Because of its generality, it provides a natural framework for integrating the logical and statistical approaches in each field. For example, DCGs and PCFGs are both special cases of Markov logic, and classical planning and MDPs can both be elegantly formulated using Markov logic and decision theory. With Markov logic, combining classical planning and MDPs, or DCGs and PCFGs, does not require new algorithms; the existing general-purpose inference and learning facilities can be directly applied.

AI can be roughly divided into "foundational" and "application" areas. Foundational areas include knowledge representation, automated reasoning, probabilistic models, and machine learning. Application areas include planning, vision, robotics, speech, natural language processing, and multi-agent systems. An interface layer for AI must provide the former, and serve the latter. We have developed the Alchemy system as an open-source embodiment of Markov logic and implementation of algorithms for it [60]. Alchemy seamlessly combines first-order knowledge representation, model checking, probabilistic inference, inductive logic programming, and generative/discriminative parameter learning. Table 1.3 compares Alchemy with Prolog and BUGS, and shows that it provides a critical mass of capabilities not previously available.

For researchers and practitioners in application areas, Alchemy offers a large reduction in the effort required to assemble a state-of-the-art solution, and to extend it beyond the state of the art.

The representation, inference and learning components required for each subtask, both logical and probabilistic, no longer need to be built or patched together piece by piece; Alchemy provides them, and the solution is built simply by writing formulas in Markov logic. A few lines of Alchemy suffice to build state-of-the-art systems for applications like collective classification, link prediction, entity resolution, information extraction, ontology mapping, and others. Because each of these pieces is now simple to implement, combining them into larger systems becomes straightforward, and is no longer a major engineering challenge. For example, we are currently beginning to build a complete natural language processing system in Alchemy, which aims to provide the functionality of current systems in one to two orders of magnitude fewer lines of code. Most significantly, Alchemy facilitates extending NLP systems beyond the current state of the art, for example by integrating probabilities into semantic analysis.

One of our goals with Alchemy is to support the growth of a repository of reusable knowledge bases in Markov logic, akin to the shareware repositories available today, and building on the traditional knowledge bases already available. Given such a repository, the first step of an application project becomes the selection of relevant knowledge. This may be used as is or manually refined. A new knowledge base is initiated by writing down plausible hypotheses about the new domain. This is followed by induction from data of new knowledge for the task. The formulas and weights of the supporting knowledge bases may also be adjusted based on data from the new task. New knowledge is added by noting and correcting the failures of the induced and refined KBs, and the process repeats. Over time, new knowledge is gradually accumulated, and existing knowledge is refined and specialized to different (sub)domains. Experience shows that neither knowledge engineering nor machine learning by itself is sufficient to reach human-level AI, and a simple two-stage solution of knowledge engineering followed by machine learning is also insufficient. What is needed is a fine-grained combination of the two, where each one bootstraps from the other, and at the end of each loop of bootstrapping the AI system's state of knowledge is more advanced than at the beginning. Alchemy supports this.

More broadly, a tool like Alchemy can help the focus of research shift from very specialized goals to higher-level ones. This is essential to speed progress in AI. As the field has grown, it has become atomized, but ultimately the pieces need to be brought back together. However, attempting to do this without an interface layer, by gluing together a large number of disparate pieces, rapidly turns into an engineering quagmire; systems become increasingly hard to build on for further progress, and eventually sheer complexity slows progress to a crawl. By keeping the pieces simpler and providing a uniform language for representing and combining them, even if at some cost in performance, an interface layer enables us to reach much farther before hitting the complexity wall. At that point, we have hopefully acquired the knowledge and insights to design the next higher-level interface layer, and in this way we can continue to make rapid progress.

Highly focused research is essential for progress, and often provides immediate real-world benefits in its own right. But these benefits will be dwarfed by those obtained if AI reaches and surpasses human intelligence, and to contribute toward this, improvements in performance in the

subtasks need to translate into improvements in the larger tasks. When the subtasks are pursued in isolation, there is no guarantee that this will happen, and in fact experience suggests that the tendency will be for the subtask solutions to evolve into local optima, which are best in isolation but not in combination. By increasing the granularity of the tasks that can be routinely attempted, platforms like Alchemy make this less likely, and help us reach human-level AI sooner.

1.4 OVERVIEW OF THE BOOK

In the remainder of this book, we will describe the Markov logic representation in greater detail, along with algorithms and applications.

In Chapter 2, we first provide basic background on first-order logic and probabilistic graphical models. We then define the Markov logic representation, building and unifying these two perspectives. We conclude by showing how Markov logic relates to some of the many other combinations of logic and probability that have been proposed in recent years.

Chapters 3 and 4 present state-of-the-art algorithms for reasoning and learning with Markov logic. These algorithms build on standard methods for first-order logic or graphical models, including satisfiability, Markov chain Monte Carlo, and belief propagation for inference; and inductive logic programming and convex optimization for learning. In many cases, these methods have been combined and extended to handle additional challenges introduced by the rich Markov logic representation.

Chapter 5 goes beyond the basic Markov logic representation to describe several extensions that increase its power or applicability to particular problems. In particular, we cover how Markov logic can be extended to continuous and infinite domains, combined with decision theory, and generalized to represent uncertain disjunctions and existential quantifiers. In addition to solving particular problems better, these extensions demonstrate that Markov logic can easily be adapted when necessary to explicitly support the features of new problems.

Chapter 6 is devoted to exploring applications of Markov logic to several real-world problems, including collective classification, link prediction, link-based clustering, entity resolution, information extraction, social network analysis, and robot mapping. Most datasets and models from this chapter can be found online at `http://alchemy.cs.washington.edu`.

We conclude in Chapter 7 with final thoughts and future directions. An appendix provides a brief introduction to Alchemy.

Sample course slides to accompany this book are available at
`http://www.cs.washington.edu/homes/pedrod/803/`.

CHAPTER 2

Markov Logic

In this chapter, we provide a detailed description of the Markov logic representation. We begin by providing background on first-order logic and probabilistic graphical models and then show how Markov logic unifies and builds on these concepts. Finally, we compare Markov logic to other representations that combine probability and logic.

2.1 FIRST-ORDER LOGIC

A *first-order knowledge base (KB)* is a set of sentences or formulas in first-order logic [37]. Formulas are constructed using four types of symbols: constants, variables, functions, and predicates. Constant symbols represent objects in the domain of interest (e.g., people: `Anna`, `Bob`, `Chris`, etc.). Variable symbols range over the objects in the domain. Function symbols (e.g., `MotherOf`) represent mappings from tuples of objects to objects. Predicate symbols represent relations among objects in the domain (e.g., `Friends`) or attributes of objects (e.g., `Smokes`). An *interpretation* specifies which objects, functions and relations in the domain are represented by which symbols. Variables and constants may be *typed*, in which case variables range only over objects of the corresponding type, and constants can only represent objects of the corresponding type. For example, the variable `x` might range over people (e.g., Anna, Bob, etc.), and the constant `C` might represent a city (e.g., Seattle, Tokyo, etc.).

A *term* is any expression representing an object in the domain. It can be a constant, a variable, or a function applied to a tuple of terms. For example, `Anna`, `x`, and `GreatestCommonDivisor(x, y)` are terms. An *atomic formula* or *atom* is a predicate symbol applied to a tuple of terms (e.g., `Friends(x, MotherOf(Anna))`). Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. If F_1 and F_2 are formulas, the following are also formulas: $\neg F_1$ (negation), which is true iff F_1 is false; $F_1 \wedge F_2$ (conjunction), which is true iff both F_1 and F_2 are true; $F_1 \vee F_2$ (disjunction), which is true iff F_1 or F_2 is true; $F_1 \Rightarrow F_2$ (implication), which is true iff F_1 is false or F_2 is true; $F_1 \Leftrightarrow F_2$ (equivalence), which is true iff F_1 and F_2 have the same truth value; $\forall \mathtt{x}\ F_1$ (universal quantification), which is true iff F_1 is true for every object `x` in the domain; and $\exists \mathtt{x}\ F_1$ (existential quantification), which is true iff F_1 is true for at least one object `x` in the domain. Parentheses may be used to enforce precedence. A *positive literal* is an atomic formula; a *negative literal* is a negated atomic formula. The formulas in a KB are implicitly conjoined, and thus a KB can be viewed as a single large formula. A *ground term* is a term containing no variables. A *ground atom* or *ground predicate* is an atomic formula all of whose arguments are ground terms. A *possible world* (along with an interpretation) assigns a truth value to each possible ground atom.

A formula is *satisfiable* iff there exists at least one world in which it is true. The basic inference problem in first-order logic is to determine whether a knowledge base *KB entails* a formula F, i.e.,

Table 2.1: Example of a first-order knowledge base and MLN. `Fr()` is short for `Friends()`, `Sm()` for `Smokes()`, and `Ca()` for `Cancer()`.

English and First-Order Logic	Clausal Form	Weight
"Friends of friends are friends." $\forall x \forall y \forall z\ \mathtt{Fr}(x, y) \wedge \mathtt{Fr}(y, z) \Rightarrow \mathtt{Fr}(x, z)$	$\neg\mathtt{Fr}(x, y) \vee \neg\mathtt{Fr}(y, z) \vee \mathtt{Fr}(x, z)$	0.7
"Smoking causes cancer." $\forall x\ \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$	$\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$	1.5
"If two people are friends and one smokes, then so does the other." $\forall x \forall y\ \mathtt{Fr}(x, y) \wedge \mathtt{Sm}(x) \Rightarrow \mathtt{Sm}(y)$	$\neg\mathtt{Fr}(x, y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$	1.1

if F is true in all worlds where KB is true (denoted by $KB \models F$). This is often done by *refutation*: KB entails F iff $KB \cup \neg F$ is unsatisfiable. (Thus, if a KB contains a contradiction, all formulas trivially follow from it, which makes painstaking knowledge engineering a necessity.) For automated inference, it is often convenient to convert formulas to a more regular form, typically *clausal form* (also known as *conjunctive normal form (CNF)*). A KB in clausal form is a conjunction of *clauses*, a clause being a disjunction of literals. Every KB in first-order logic can be converted to clausal form using a mechanical sequence of steps.[1] Clausal form is used in resolution, a sound and refutation-complete inference procedure for first-order logic [122].

Inference in first-order logic is only semidecidable. Because of this, knowledge bases are often constructed using a restricted subset of first-order logic with more desirable properties. The most widely used restriction is to *Horn clauses*, which are clauses containing at most one positive literal. The Prolog programming language is based on Horn clause logic [72]. Prolog programs can be learned from databases by searching for Horn clauses that (approximately) hold in the data; this is studied in the field of inductive logic programming (ILP) [65].

Table 2.1 shows a simple KB and its conversion to clausal form. Note that, while these formulas may be *typically* true in the real world, they are not *always* true. In most domains it is very difficult to come up with non-trivial formulas that are always true, and such formulas capture only a fraction of the relevant knowledge. Thus, despite its expressiveness, pure first-order logic has limited applicability to practical AI problems. Many *ad hoc* extensions to address this have been proposed. In the more limited case of propositional logic, the problem is well solved by probabilistic graphical models such as Markov networks, described in the next section. We will later show how to generalize these models to the first-order case.

[1] This conversion includes the removal of existential quantifiers by Skolemization, which is not sound in general. However, in finite domains an existentially quantified formula can simply be replaced by a disjunction of its groundings.

2.2 MARKOV NETWORKS

A *Markov network* (also known as *Markov random field*) is a model for the joint distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in \mathcal{X}$ [99]. It is composed of an undirected graph G and a set of potential functions ϕ_k. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real-valued function of the state of the corresponding clique. The joint distribution represented by a Markov network is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}}) \tag{2.1}$$

where $x_{\{k\}}$ is the state of the kth clique (i.e., the state of the variables that appear in that clique). Z, known as the *partition function*, is given by $Z = \sum_{x \in \mathcal{X}} \prod_k \phi_k(x_{\{k\}})$. Markov networks are often conveniently represented as *log-linear models*, with each clique potential replaced by an exponentiated weighted sum of features of the state, leading to

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x) \right) \tag{2.2}$$

A feature may be any real-valued function of the state. Except where stated, this book will focus on binary features, $f_j(x) \in \{0, 1\}$. In the most direct translation from the potential-function form (Equation 2.1), there is one feature corresponding to each possible state $x_{\{k\}}$ of each clique, with its weight being $\log \phi_k(x_{\{k\}})$. This representation is exponential in the size of the cliques. However, we are free to specify a much smaller number of features (e.g., logical functions of the state of the clique), allowing for a more compact representation than the potential-function form, particularly when large cliques are present. Markov logic will take advantage of this.

Inference in Markov networks is #P-complete [123]. The most widely used method for approximate inference in Markov networks is Markov chain Monte Carlo (MCMC) [40], and in particular Gibbs sampling, which proceeds by sampling each variable in turn given its Markov blanket. (The Markov blanket of a node is the minimal set of nodes that renders it independent of the remaining network; in a Markov network, this is simply the node's neighbors in the graph.) Marginal probabilities are computed by counting over these samples; conditional probabilities are computed by running the Gibbs sampler with the conditioning variables clamped to their given values.

Another popular method for inference in Markov networks is belief propagation [156], a message-passing algorithm that performs exact inference on tree-structured Markov networks. When applied to graphs with loops, the results are approximate and the algorithm may not converge. Nonetheless, loopy belief propagation is more efficient than Gibbs sampling in many applications.

Maximum-likelihood or MAP estimates of Markov network weights cannot be computed in closed form but, because the log-likelihood is a concave function of the weights, they can be found efficiently (modulo inference) using standard gradient-based or quasi-Newton optimization

methods [95]. Another alternative is iterative scaling [24]. Features can also be learned from data, for example by greedily constructing conjunctions of atomic features [24].

2.3 MARKOV LOGIC

A first-order KB can be seen as a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. The basic idea in MLNs is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal.

Definition 2.1. A *Markov logic network* L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ (Equations 2.1 and 2.2) as follows:

1. $M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground predicate is true, and 0 otherwise.

2. $M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.

The syntax of the formulas in an MLN is the standard syntax of first-order logic [37]. Free (unquantified) variables are treated as universally quantified at the outermost level of the formula. In this book, we will often assume that the set of formulas is in function-free clausal form for convenience, but our methods can be applied to other MLNs as well.

An MLN can be viewed as a *template* for constructing Markov networks. Given different sets of constants, it will produce different networks, and these may be of widely varying size, but all will have certain regularities in structure and parameters, given by the MLN (e.g., all groundings of the same formula will have the same weight). We call each of these networks a *ground Markov network* to distinguish it from the first-order MLN. From Definition 2.1 and Equations 2.1 and 2.2, the probability distribution over possible worlds x specified by the ground Markov network $M_{L,C}$ is given by

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right) = \frac{1}{Z} \prod_i \phi_i(x_{\{i\}})^{n_i(x)} \tag{2.3}$$

where $n_i(x)$ is the number of true groundings of F_i in x, $x_{\{i\}}$ is the state (truth values) of the predicates appearing in F_i, and $\phi_i(x_{\{i\}}) = e^{w_i}$. Note that, although we defined MLNs as log-linear models, they could equally well be defined as products of potential functions, as the second equality above shows. This will be the most convenient approach in domains with a mixture of hard and

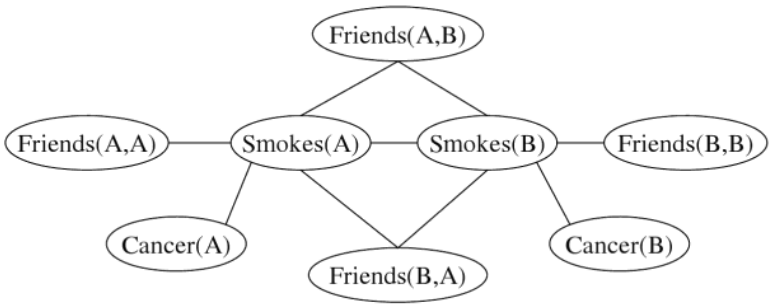

Figure 2.1: Ground Markov network obtained by applying the last two formulas in Table 2.1 to the constants `Anna(A)` and `Bob(B)`.

soft constraints (i.e., where some formulas hold with certainty, leading to zero probabilities for some worlds).

The graphical structure of $M_{L,C}$ follows from Definition 2.1: there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground predicates appear together in at least one grounding of one formula in L. Thus, the predicates in each ground formula form a (not necessarily maximal) clique in $M_{L,C}$. Figure 2.1 shows the graph of the ground Markov network defined by the last two formulas in Table 2.1 and the constants `Anna` and `Bob`. Each node in this graph is a ground predicate (e.g., `Friends(Anna, Bob)`). The graph contains an arc between each pair of predicates that appear together in some grounding of one of the formulas. $M_{L,C}$ can now be used to infer the probability that Anna and Bob are friends given their smoking habits, the probability that Bob has cancer given his friendship with Anna and whether she has cancer, etc.

Each state of $M_{L,C}$ represents a possible world. A possible world is a set of objects, a set of functions (mappings from tuples of objects to objects), and a set of relations that hold between those objects; together with an interpretation, they determine the truth value of each ground predicate. The following assumptions ensure that the set of possible worlds for (L, C) is finite, and that $M_{L,C}$ represents a unique, well-defined probability distribution over those worlds, irrespective of the interpretation and domain. These assumptions are quite reasonable in most practical applications, and greatly simplify the use of MLNs. For the remaining cases, we discuss below the extent to which each one can be relaxed.

Assumption 2.2. Unique names. *Different constants refer to different objects [37].*

Assumption 2.3. Domain closure. *The only objects in the domain are those representable using the constant and function symbols in* (L, C) *[37].*

Table 2.2: Construction of all groundings of a first-order formula under Assumptions 2.2–2.4.

function Ground(F)
  input: F, a formula in first-order logic
  output: G_F, a set of ground formulas
for each existentially quantified subformula $\exists x\ S(x)$ in F
  $F \leftarrow F$ with $\exists x\ S(x)$ replaced by $S(c_1) \vee S(c_2) \vee \ldots \vee S(c_{|C|})$,
    where $S(c_i)$ is $S(x)$ with x replaced by c_i
$G_F \leftarrow \{F\}$
for each universally quantified variable x
  for each formula $F_j(x)$ in G_F
    $G_F \leftarrow (G_F \setminus F_j(x)) \cup \{F_j(c_1), F_j(c_2), \ldots, F_j(c_{|C|})\}$,
      where $F_j(c_i)$ is $F_j(x)$ with x replaced by c_i
for each formula $F_j \in G_F$
  repeat
    for each function $f(a_1, a_2, \ldots)$ all of whose arguments are constants
    $F_j \leftarrow F_j$ with $f(a_1, a_2, \ldots)$ replaced by c, where $c = f(a_1, a_2, \ldots)$
  until F_j contains no functions
return G_F

Assumption 2.4. Known functions. *For each function appearing in L, the value of that function applied to every possible tuple of arguments is known, and is an element of C.*

This last assumption allows us to replace functions by their values when grounding formulas. Thus the only ground predicates that need to be considered are those having constants as arguments. The infinite number of terms constructible from all functions and constants in (L, C) (the "Herbrand universe" of (L, C)) can be ignored, because each of those terms corresponds to a known constant in C, and predicates involving them are already represented as the predicates involving the corresponding constants. The possible groundings of a predicate in Definition 2.1 are thus obtained simply by replacing each variable in the predicate with each constant in C, and replacing each function term in the predicate by the corresponding constant. Table 2.2 shows how the groundings of a formula are obtained given Assumptions 2.2–2.4.

Assumption 2.2 (unique names) can be removed by introducing the equality predicate (`Equals(x, y)`, or `x = y` for short) and adding the necessary axioms to the MLN: equality is reflexive, symmetric and transitive; for each unary predicate P, $\forall \mathtt{x} \forall \mathtt{y}\ \mathtt{x} = \mathtt{y} \Rightarrow (\mathtt{P(x)} \Leftrightarrow \mathtt{P(y)})$; and similarly for higher-order predicates and functions [37]. The resulting MLN will have a node for each pair of constants, whose value is 1 if the constants represent the same object and 0 otherwise; these nodes will be connected to each other and to the rest of the network by arcs representing the axioms above.

Note that this allows us to make probabilistic inferences about the equality of two constants. The example in Section 6.3 successfully uses this as the basis of an approach to entity resolution.

If the number u of unknown objects is known, Assumption 2.3 (domain closure) can be removed simply by introducing u arbitrary new constants. If u is unknown but finite, Assumption 2.3 can be removed by introducing a distribution over u, grounding the MLN with each number of unknown objects, and computing the probability of a formula F as $P(F) = \sum_{u=0}^{u_{max}} P(u) P(F|M^u_{L,C})$, where $M^u_{L,C}$ is the ground MLN with u unknown objects. Markov logic can also be applied to infinite domains; details are in Section 5.2.

Let $H_{L,C}$ be the set of all ground terms constructible from the function symbols in L and the constants in L and C (the "Herbrand universe" of (L, C)). Assumption 2.4 (known functions) can be removed by treating each element of $H_{L,C}$ as an additional constant and applying the same procedure used to remove the unique names assumption. For example, with a function `G(x)` and constants `A` and `B`, the MLN will now contain nodes for `G(A)` = `A`, `G(A)` = `B`, etc. This leads to an infinite number of new constants, requiring the corresponding extension of MLNs. However, if we restrict the level of nesting to some maximum, the resulting MLN is still finite.

To summarize, Assumptions 2.2–2.4 can be removed as long the domain is finite. Section 5.2 discusses how to extend MLNs to infinite domains. In the remainder of this book we proceed under Assumptions 2.2–2.4, except where noted.

A first-order KB can be transformed into an MLN simply by assigning a weight to each formula. For example, the formulas (or clauses) and weights in the last two columns of Table 2.1 constitute an MLN. According to this MLN, other things being equal, a world where n smokers don't have cancer is $e^{1.5n}$ times less probable than a world where all smokers have cancer. Note that all the formulas in Table 2.1 are false in the real world as universally quantified logical statements, but capture useful information on friendships and smoking habits, when viewed as features of a Markov network. For example, it is well known that teenage friends tend to have similar smoking habits [73]. In fact, an MLN like the one in Table 2.1 succinctly represents a type of model that is a staple of social network analysis [149].

It is easy to see that MLNs subsume essentially all propositional probabilistic models, as detailed below.

Theorem 2.5. *Every probability distribution over discrete or finite-precision numeric variables can be represented as a Markov logic network.*

Proof. Consider first the case of Boolean variables $(X_1, X_2, \ldots, X_n)$. Define a predicate of zero arity R_h for each variable X_h, and include in the MLN L a formula for each possible state of $(X_1, X_2, \ldots, X_n)$. This formula is a conjunction of n literals, with the hth literal being $R_h()$ if X_h is true in the state, and $\neg R_h()$ otherwise. The formula's weight is $\log P(X_1, X_2, \ldots, X_n)$. (If some states have zero probability, use instead the product form (see Equation 2.3), with $\phi_i()$ equal to the probability of the ith state.) Since all predicates in L have zero arity, L defines the same Markov network $M_{L,C}$ irrespective of C, with one node for each variable X_h. For any state, the corresponding

formula is true and all others are false, and thus Equation 2.3 represents the original distribution (notice that $Z = 1$). The generalization to arbitrary discrete variables is straightforward, by defining a zero-arity predicate for each value of each variable. Similarly for finite-precision numeric variables, by noting that they can be represented as Boolean vectors. □

Of course, compact factored models like Markov networks and Bayesian networks can still be represented compactly by MLNs, by defining formulas for the corresponding factors (arbitrary features in Markov networks, and states of a node and its parents in Bayesian networks).[2]

First-order logic (with Assumptions 2.2–2.4 above) is the special case of MLNs obtained when all weights are equal and tend to infinity, as described below.

Theorem 2.6. *Let KB be a satisfiable knowledge base, L be the MLN obtained by assigning weight w to every formula in KB, C be the set of constants appearing in KB, $P_w(x)$ be the probability assigned to a (set of) possible world(s) x by $M_{L,C}$, $\mathcal{X}_{KB}$ be the set of worlds that satisfy KB, and F be an arbitrary formula in first-order logic. Then:*

1. $\forall x \in \mathcal{X}_{KB} \ \lim_{w\to\infty} P_w(x) = |\mathcal{X}_{KB}|^{-1}$
   $\forall x \notin \mathcal{X}_{KB} \ \lim_{w\to\infty} P_w(x) = 0$
2. *For all F, $KB \models F$ iff* $\lim_{w\to\infty} P_w(F) = 1$.

Proof. Let k be the number of ground formulas in $M_{L,C}$. By Equation 2.3, if $x \in \mathcal{X}_{KB}$ then $P_w(x) = e^{kw}/Z$, and if $x \notin \mathcal{X}_{KB}$ then $P_w(x) \leq e^{(k-1)w}/Z$. Thus all $x \in \mathcal{X}_{KB}$ are equiprobable and $\lim_{w\to\infty} P(\mathcal{X} \setminus \mathcal{X}_{KB})/P(\mathcal{X}_{KB}) \leq \lim_{w\to\infty}(|\mathcal{X} \setminus \mathcal{X}_{KB}|/|\mathcal{X}_{KB}|)e^{-w} = 0$, proving Part 1. By definition of entailment, $KB \models F$ iff every world that satisfies KB also satisfies F. Therefore, letting $\mathcal{X}_F$ be the set of worlds that satisfy F, if $KB \models F$ then $\mathcal{X}_{KB} \subseteq \mathcal{X}_F$ and $P_w(F) = \sum_{x\in\mathcal{X}_F} P_w(x) \geq P_w(\mathcal{X}_{KB})$. Since, from Part 1, $\lim_{w\to\infty} P_w(\mathcal{X}_{KB}) = 1$, this implies that if' $KB \models F$ then $P_w(F) = 1$. The inverse direction of Part 2 is proved by noting that if $P_w(F) = 1$ then every world with non-zero probability must satisfy F, and this includes every world in $\mathcal{X}_{KB}$. □

In other words, in the limit of all equal infinite weights, the MLN represents a uniform distribution over the worlds that satisfy the KB, and all entailment queries can be answered by computing the probability of the query formula and checking whether it is 1. Even when weights are finite, first-order logic is "embedded" in MLNs in the following sense. Assume without loss of generality that all weights are non-negative. (A formula with a negative weight w can be replaced by its negation with weight $-w$.) If the knowledge base composed of the formulas in an MLN L (negated, if their weight is negative) is satisfiable, then, for any C, the satisfying assignments are the modes of the distribution represented by $M_{L,C}$. This is because the modes are the worlds x with maximum $\sum_i w_i n_i(x)$ (see Equation 2.3), and this expression is maximized when all groundings of

[2]While some conditional independence structures can be compactly represented with directed graphs but not with undirected ones, they still lead to compact models in the form of Equation 2.3 (i.e., as products of potential functions).

all formulas are true (i.e., the KB is satisfied). Unlike an ordinary first-order KB, however, an MLN can produce useful results even when it contains contradictions. An MLN can also be obtained by merging several KBs, even if they are partly incompatible. This is potentially useful in areas like the Semantic Web [5] and mass collaboration [116].

It is interesting to see a simple example of how MLNs generalize first-order logic. Consider an MLN containing the single formula $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ with weight w, and $C = \{\mathtt{A}\}$. This leads to four possible worlds: $\{\neg\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, $\{\neg\mathtt{R(A)}, \mathtt{S(A)}\}$, $\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}$, and $\{\mathtt{R(A)}, \mathtt{S(A)}\}$. From Equation 2.3 we obtain that $P(\{\mathtt{R(A)}, \neg\mathtt{S(A)}\}) = 1/(3e^w + 1)$ and the probability of each of the other three worlds is $e^w/(3e^w + 1)$. (The denominator is the partition function Z; see Section 2.2.) Thus, if $w > 0$, the effect of the MLN is to make the world that is inconsistent with $\forall \mathtt{x}\ \mathtt{R(x)} \Rightarrow \mathtt{S(x)}$ less likely than the other three. From the probabilities above we obtain that $P(\mathtt{S(A)}|\mathtt{R(A)}) = 1/(1 + e^{-w})$. When $w \to \infty$, $P(\mathtt{S(A)}|\mathtt{R(A)}) \to 1$, recovering the logical entailment.

A first-order KB partitions the set of possible worlds into two subsets: those that satisfy the KB and those that do not. An MLN has many more degrees of freedom: it can partition the set of possible worlds into many more subsets, and assign a different probability to each. How to use this freedom is a key decision for both knowledge engineering and learning. At one extreme, the MLN can add little to logic, treating the whole knowledge base as a single formula, and assigning one probability to the worlds that satisfy it and another to the worlds that do not. At the other extreme, each formula in the KB can be converted into clausal form, and a weight associated with each clause.[3] The more finely divided into subformulas a KB is, the more gradual the drop-off in probability as a world violates more of those subformulas, and the greater the flexibility in specifying distributions over worlds. From a knowledge engineering point of view, the decision about which formulas constitute indivisible constraints should reflect domain knowledge and the goals of modeling. From a learning point of view, dividing the KB into more formulas increases the number of parameters, with the corresponding tradeoff in bias and variance.

It is also interesting to see an example of how MLNs generalize commonly-used statistical models. One of the most widely used models for classification is logistic regression. Logistic regression predicts the probability that an example with features $f = (f_1, \ldots, f_i, \ldots)$ is of class c according to the equation:

$$\log\left(\frac{P(C = 1|F = f)}{P(C = 0|F = f)}\right) = a + \sum_{i=1}^{n} b_i f_i$$

This can be implemented as an MLN using a unit clause for the class, $\mathtt{C(x)}$, with weight a, and a formula of the form $\mathtt{F_i(x)} \wedge \mathtt{C(x)}$ for each feature, with weight b_i. This yields the distribution

$$P(C = c, F = f) = \frac{1}{Z}\exp\left(ac + \sum_i b_i f_i c\right)$$

[3] This conversion can be done in the standard way [37], except that, instead of introducing Skolem functions, existentially quantified formulas should be replaced by disjunctions, as in Table 2.2.

resulting in

$$\frac{P(C=1|F=f)}{P(C=0|F=f)} = \frac{\exp\left(a+\sum_i b_i f_i\right)}{\exp(0)} = \exp\left(a+\sum_i^n b_i f_i\right)$$

as desired.

In practice, we have found it useful to add each predicate to the MLN as a unit clause. In other words, for each predicate $R(x_1, x_2, \ldots)$ appearing in the MLN, we add the formula $\forall x_1, x_2, \ldots \; R(x_1, x_2, \ldots)$ with some weight w_R. The weight of a unit clause can (roughly speaking) capture the marginal distribution of the corresponding predicate, leaving the weights of the non-unit clauses free to model only dependencies between predicates.

When manually constructing an MLN or interpreting a learned one, it is useful to have an intuitive understanding of the weights. Consider a ground formula F with weight w. All other things being equal, a world where F is true is e^w times as likely as a world where F is false. Let U_t and U_f be the number of possible worlds in which F is true and false, respectively. If F is independent from all other ground formulas, then its probability is given by the following function:

$$P(F) = \frac{1}{1+\frac{U_f}{U_t}e^{-w}}$$

Solving for w:

$$w = \log\frac{P(F)}{P(\neg F)} - \log\frac{U_t}{U_f}$$

Therefore, w can be interpreted as the difference between the log odds of F according to the MLN and according to the uniform distribution. However, if F shares atoms with other formulas, as will typically be the case, it may not be possible to keep those formulas' truth values unchanged while reversing F's. In this case, there is no longer a one-to-one correspondence between weights and probabilities of formulas.[4] Nevertheless, the probabilities of all formulas collectively determine all weights if we view them as constraints on a maximum entropy distribution, or treat them as empirical probabilities and learn the maximum likelihood weights (the two are equivalent) [24]. Thus a good way to set the weights of an MLN is to write down the probability with which each formula should hold, treat these as empirical frequencies, and learn the weights from them using the algorithms in Section 4.1. Conversely, the weights in a learned MLN can be viewed as collectively encoding the empirical formula probabilities.

The size of ground Markov networks can be vastly reduced by having typed constants and variables, and only grounding variables to constants of the same type. However, even in this case the size of the network may still be extremely large. Fortunately, there are a number of ways to further reduce this size, as we will see in Chapter 3.

[4]This is an unavoidable side-effect of the power and flexibility of Markov networks. In Bayesian networks, parameters are probabilities, but at the cost of greatly restricting the ways in which the distribution may be factored. In particular, potential functions must be conditional probabilities, and the directed graph must have no cycles. The latter condition is particularly troublesome to enforce in relational extensions [141].

2.4 RELATION TO OTHER APPROACHES

There is a very large literature relating logic and probability; here we will focus only on the approaches most closely related to Markov logic.

EARLY WORK

Attempts to combine logic and probability in AI date back to at least Nilsson [94]. Bacchus [1], Halpern [43] and coworkers (e.g., [2]) studied the problem in detail from a theoretical standpoint. They made a distinction between statistical statements (e.g., "65% of the students in our department are undergraduate") and statements about possible worlds (e.g., "The probability that Anna is an undergraduate is 65%"), and provided methods for computing the latter from the former. In their approach, a KB did not specify a complete and unique distribution over possible worlds, leaving its status as a probabilistic model unclear. MLNs overcome this limitation by viewing KBs as Markov network templates.

Paskin [97] extended the work of Bacchus *et al.* [2] by associating a probability with each first-order formula, and taking the maximum entropy distribution compatible with those probabilities. This representation was still quite brittle, with a world that violates a single grounding of a universally quantified formula being considered as unlikely as a world that violates all of them. In contrast, in MLNs a rule like $\forall x\ \texttt{Smokes(x)} \Rightarrow \texttt{Cancer(x)}$ causes the probability of a world to decrease smoothly as the number of cancer-free smokers in it increases.

KNOWLEDGE-BASED MODEL CONSTRUCTION

Knowledge-based model construction (KBMC) is a combination of logic programming and Bayesian networks [151, 93, 55]. As in MLNs, nodes in KBMC represent ground predicates. Given a Horn KB, KBMC answers a query by finding all possible backward-chaining proofs of the query and evidence predicates from each other, constructing a Bayesian network over the ground predicates in the proofs, and performing inference over this network. The parents of a predicate node in the network are deterministic AND nodes representing the bodies of the clauses that have that node as the head. The conditional probability of the node given these is specified by a combination function (e.g., noisy OR or logistic regression). MLNs have several advantages compared to KBMC: they allow arbitrary formulas (not just Horn clauses) and inference in any direction, they sidestep the thorny problem of avoiding cycles in the Bayesian networks constructed by KBMC, and they do not require the introduction of *ad hoc* combination functions for clauses with the same consequent.

A KBMC model can be translated into an MLN by writing down a set of formulas for each first-order predicate $\texttt{P}_\texttt{k}(\ldots)$ in the domain. Each formula is a conjunction containing $\texttt{P}_\texttt{k}(\ldots)$ and one literal per parent of $\texttt{P}_\texttt{k}(\ldots)$ (i.e., per first-order predicate appearing in a Horn clause having $\texttt{P}_\texttt{k}(\ldots)$ as the consequent). A subset of these literals are negated; there is one formula for each possible combination of positive and negative literals. The weight of the formula is $w = \log[p/(1-p)]$, where p is the conditional probability of the child predicate when the corresponding conjunction of parent literals is true, according to the combination function used. If the combination function is

logistic regression, it can be represented using only a linear number of formulas, taking advantage of the fact that a logistic regression model is a (conditional) Markov network with a binary clique between each predictor and the response. Noisy OR can similarly be represented with a linear number of parents.

OTHER LOGIC PROGRAMMING APPROACHES

Stochastic logic programs (SLPs) [87, 17] are a combination of logic programming and log-linear models. Puech and Muggleton [111] showed that SLPs are a special case of KBMC, and thus they can be converted into MLNs in the same way. Like MLNs, SLPs have one coefficient per clause, but they represent distributions over Prolog proof trees rather than over predicates; the latter have to be obtained by marginalization. Similar remarks apply to a number of other representations that are essentially equivalent to SLPs, like independent choice logic [103] and PRISM [129].

MACCENT [23] is a system that learns log-linear models with first-order features; each feature is a conjunction of a class and a Prolog query (clause with empty head). A key difference between MACCENT and MLNs is that MACCENT is a classification system (i.e., it predicts the conditional distribution of an object's class given its properties), while an MLN represents the full joint distribution of a set of predicates. Like any probability estimation approach, MLNs can be used for classification simply by issuing the appropriate conditional queries.[5] In particular, a MACCENT model can be converted into an MLN simply by defining a class predicate, adding the corresponding features and their weights to the MLN, and adding a formula with infinite weight stating that each object must have exactly one class. (This fails to model the marginal distribution of the non-class predicates, which is not a problem if only classification queries will be issued.) MACCENT can make use of deterministic background knowledge in the form of Prolog clauses; these can be added to the MLN as formulas with infinite weight. In addition, MLNs allow uncertain background knowledge (via formulas with finite weights). As we demonstrate in Section 6.1, MLNs can be used for collective classification, where the classes of different objects can depend on each other; MACCENT, which requires that each object be represented in a separate Prolog knowledge base, does not have this capability.

Constraint logic programming (CLP) is an extension of logic programming where variables are constrained instead of being bound to specific values during inference [64]. Probabilistic CLP generalizes SLPs to CLP [121], and CLP($\mathcal{BN}$) combines CLP with Bayesian networks [128]. Unlike in MLNs, constraints in CLP($\mathcal{BN}$) are hard (i.e., they cannot be violated; rather, they define the form of the probability distribution).

PROBABILISTIC RELATIONAL MODELS

Probabilistic relational models (PRMs) [36] are a combination of frame-based systems and Bayesian networks. PRMs can be converted into MLNs by defining a predicate $\mathtt{S}(\mathtt{x}, \mathtt{v})$ for each (propositional or relational) attribute of each class, where $\mathtt{S}(\mathtt{x}, \mathtt{v})$ means "The value of attribute $\mathtt{S}$ in object $\mathtt{x}$ is $\mathtt{v}$."

[5] Conversely, joint distributions can be built up from classifiers (e.g., [44]), but this would be a significant extension of MACCENT.

CHAPTER 3

Inference

Inference in Markov logic lets us reason probabilistically about complex relationships. Since an MLN acts as a template for a Markov network, we can always answer probabilistic queries using standard Markov network inference methods on the instantiated network. However, due to the size and complexity of the resulting network, this is often infeasible. Instead, the methods we discuss here combine probabilistic methods with ideas from logical inference, including satisfiability and resolution. This leads to efficient methods that take full advantage of the logical structure.

We consider two basic types of inference: finding the most likely state of the world consistent with some evidence, and computing arbitrary conditional probabilities. We then discuss two approaches to making inference more tractable on large, relational problems: lazy inference, in which only the groundings that deviate from a "default" value need to be instantiated; and lifted inference, in which we group indistinguishable atoms together and treat them as a single unit during inference.

3.1 INFERRING THE MOST PROBABLE EXPLANATION

A basic inference task is finding the most probable state of the world y given some evidence x, where x is a set of literals. (This is known as MAP inference in the Markov network literature, and MPE inference in the Bayesian network literature.[1]) For Markov logic, this is formally defined as follows:

$$\begin{aligned} \arg\max_y P(y|x) &= \arg\max_y \frac{1}{Z_x} \exp\left(\sum_i w_i n_i(x, y)\right) \\ &= \arg\max_y \sum_i w_i n_i(x, y) \end{aligned} \tag{3.1}$$

The first equality is due to Equation 2.3, which defines of the probability of a possible world. The normalization constant is written as Z_x to reflect the fact that we are only normalizing over possible worlds consistent with x. In the second equality, we remove Z_x since, being constant, it does not affect the arg max operation. We can also remove the exponentiation because it is a monotonic function. Therefore, the MPE problem in Markov logic reduces to finding the truth assignment that maximizes the sum of weights of satisfied clauses.

This can be done using any weighted satisfiability solver, and (remarkably) need not be more expensive than standard logical inference by model checking. (In fact, it can be faster, if some hard constraints are softened.) The problem is NP-hard in general, but effective solvers exist, both exact and approximate. The most commonly used approximate solver is MaxWalkSAT, a weighted variant

[1] The term "MAP inference" is sometimes used to refer to finding the most probable configuration of a set of query variables, given some evidence. The necessity of summing out all non-query, non-evidence variables makes this a harder inference problem than the one we consider here, in which y is the *complete* state of the world.

3.2 COMPUTING CONDITIONAL PROBABILITIES

MLNs can answer arbitrary queries of the form "What is the probability that formula F_1 holds given that formula F_2 does?" If F_1 and F_2 are two formulas in first-order logic, C is a finite set of constants including any constants that appear in F_1 or F_2, and L is an MLN, then

$$\begin{aligned} P(F_1|F_2, L, C) &= P(F_1|F_2, M_{L,C}) \\ &= \frac{P(F_1 \wedge F_2|M_{L,C})}{P(F_2|M_{L,C})} \\ &= \frac{\sum_{x \in \mathcal{X}_{F_1} \cap \mathcal{X}_{F_2}} P(X=x|M_{L,C})}{\sum_{x \in \mathcal{X}_{F_2}} P(X=x|M_{L,C})} \end{aligned} \tag{3.2}$$

where $\mathcal{X}_{F_i}$ is the set of worlds where F_i holds, $M_{L,C}$ is the Markov network defined by L and C, and $P(X = x|M_{L,C})$ is given by Equation 2.3. Ordinary conditional queries in graphical models are the special case of Equation 3.2 where all predicates in F_1, F_2 and L are zero-arity and the formulas are conjunctions. The question of whether a knowledge base KB entails a formula F in first-order logic is the question of whether $P(F|L_{KB}, C_{KB,F}) = 1$, where L is the MLN obtained by assigning infinite weight to all the formulas in KB, and $C_{KB,F}$ is the set of all constants appearing in KB or F. The question is answered by computing $P(F|L_{KB}, C_{KB,F})$ by Equation 3.2, with $F_2 = \text{True}$.

Computing Equation 3.2 directly is intractable in all but the smallest domains. Since MLN inference subsumes probabilistic inference, which is #P-complete, and logical inference, which is NP-complete even in finite domains, no better results can be expected. However, many of the large number of techniques for efficient inference in either case are applicable to MLNs. Because MLNs allow fine-grained encoding of knowledge, including context-specific independences, inference in them may in some cases be more efficient than inference in an ordinary graphical model for the same domain. On the logic side, the probabilistic semantics of MLNs allows for approximate inference, with the corresponding potential gains in efficiency.

In principle, $P(F_1|F_2, L, C)$ can be approximated using an MCMC algorithm that rejects all moves to states where F_2 does not hold, and counts the number of samples in which F_1 holds. However, this may be too slow for arbitrary formulas. Instead, we focus on the case where F_2 is a conjunction of ground literals. While less general than Equation 3.2, this is the most frequent type of query in practice. In this scenario, further efficiency can be gained by applying a generalization of knowledge-based model construction [151]. The basic idea is to only construct the minimal subset of the ground network required to answer the query. This network is constructed by checking if the atoms that the query formula directly depends on are in the evidence. If they are, the construction is complete. Those that are not are added to the network, and we in turn check the atoms they depend on. This process is repeated until all relevant atoms have been retrieved. While in the worst case it yields no savings, in practice it can vastly reduce the time and memory required for inference.

Pseudocode for the network construction algorithm is shown in Table 3.2. See Figure 3.1 for an example of the resulting network. The size of the network returned may be further reduced, and the algorithm sped up, by noticing that any ground formula that is made true by the evidence can

A full proof can be found in Singla and Domingos [137].

Corollary 5.15. *Let* $\mathbf{K}$ *be a locally finite knowledge base. Let* α *be a first-order formula, and* $\mathbf{L}^{\alpha}_{\infty}$ *be the MLN obtained by assigning weight* $w \to \infty$ *to all clauses in* $\mathbf{K} \cup \{\neg\alpha\}$. *Then* $\mathbf{K}$ *entails* α *iff* $\mathbf{L}^{\alpha}_{\infty}$ *has no satisfying measure. Mathematically,*

$$\mathbf{K} \models \alpha \;\Leftrightarrow\; |\mathcal{S}(\gamma^{\mathrm{L}^{\alpha}_{\infty}})| = 0 \tag{5.11}$$

Thus, for locally finite knowledge bases with Herbrand interpretations, first-order logic can be viewed as the limiting case of Markov logic when all weights tend to infinity. Whether these conditions can be relaxed is a question for future work.

5.3 RECURSIVE MARKOV LOGIC

In Markov logic, the unification of logic and probability is incomplete. Markov logic only treats the top-level conjunction and universal quantifiers in a knowledge base as probabilistic, when in principle any logical combination can be viewed as the limiting case of an underlying probability distribution; disjunctions and existential quantifiers remain deterministic. Thus the symmetry between conjunctions and disjunctions, and between universal and existential quantifiers, is lost (except in the infinite-weight limit).

For example, an MLN with the formula $\mathtt{R(X)} \wedge \mathtt{S(X)}$ can treat worlds that violate both $\mathtt{R(X)}$ and $\mathtt{S(X)}$ as less probable than worlds that only violate one. Since an MLN acts as a soft conjunction, the groundings of $\mathtt{R(X)}$ and $\mathtt{S(X)}$ simply appear as distinct formulas. (As usual, we will assume that the knowledge base is converted to CNF before performing learning or inference.) This is not possible for the disjunction $\mathtt{R(X)} \vee \mathtt{S(X)}$: no distinction is made between satisfying both $\mathtt{R(X)}$ and $\mathtt{S(X)}$ and satisfying just one. Since a universally quantified formula is effectively a conjunction over all its groundings, while an existentially quantified formula is a disjunction over them, this leads to the two quantifiers being handled differently.

This asymmetry can be avoided by "softening" disjunction and existential quantification in the same way that Markov logic softens conjunction and universal quantification. The result is a representation in which MLNs can have nested MLNs as features. We call these recursive Markov logic networks, or *recursive random fields (RRFs)* for short.

RRFs have many desirable properties, including the ability to represent distributions like noisy DNF, rules with exceptions, and m-of-all quantifiers much more compactly than MLNs. RRFs also allow more flexibility in revising first-order theories to maximize data likelihood. Standard methods for inference in Markov networks are easily extended to RRFs, and weight learning can be carried out efficiently using a variant of the backpropagation algorithm.

RRF theory revision can be viewed as a first-order probabilistic analog of the KBANN algorithm, which initializes a neural network with a propositional theory and uses backpropagation to

improve its fit to data [145]. A propositional RRF (where all predicates have zero arity) differs from a multilayer perceptron in that its output is the joint probability of its inputs, not the regression of a variable on others (or, in the probabilistic version, its conditional probability). Propositional RRFs are an alternative to Boltzmann machines, with nested features playing the role of hidden variables. Because the nested features are deterministic functions of the inputs, learning does not require EM, and inference does not require marginalizing out variables.

A recursive random field is a log-linear model in which each feature is either an observable random variable or the output of another recursive random field. To build up intuition, we first describe the propositional case, then generalize it to the more interesting relational case. A concrete example is given in a later subsection, and illustrated in Figure 5.1.

PROPOSITIONAL RRFS

While our primary goal is solving relational problems, RRFs may be interesting in propositional domains as well. Propositional RRFs extend Markov random fields and Boltzmann machines in the same way multilayer perceptrons extend single-layer ones. The extension is very simple in principle, but allows RRFs to compactly represent important concepts, such as m-of-n. It also allows RRFs to learn features via weight learning, which could be more effective than current feature-search methods for Markov random fields.

The probability distribution represented by a propositional RRF is as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z_0} \exp\left(\sum_i w_i f_i(\mathbf{x})\right)$$

where Z_0 is a normalization constant, to ensure that the probabilities of all possible states $\mathbf{x}$ sum to 1. What makes this different from a standard Markov network is that the features can be built up from other subfeatures to an arbitrary number of levels. Specifically, each $f_i(\mathbf{x})$ is either:

$$\begin{aligned} f_i(\mathbf{x}) &= x_j \quad \text{(base case), or} \\ f_i(\mathbf{x}) &= \frac{1}{Z_i} \exp\left(\sum_j w_{ij} f_j(\mathbf{x})\right) \quad \text{(recursive case)} \end{aligned}$$

In the recursive case, the summation is over all features f_j referenced by the "parent" feature f_i. A child feature, f_j, can appear in more than one parent feature, and thus an RRF can be viewed as a directed acyclic graph of features. The attribute values are at the leaves, and the probability of their configuration is given by the root. (Note that the probabilistic graphical model represented by the RRF is still undirected.)

Since the overall distribution is simply a recursive feature, we can also write the probability distribution as follows:

$$P(\mathbf{X} = \mathbf{x}) = f_0(\mathbf{x})$$