LEARNING CLASSIFIERS WITH MARKOV LOGIC NETWORKS

Victor A. Silva*, Rodrigo B. Polastro*, Fabio G. Cozman*

*Laboratório de Tomada de Decisão
Escola Politécnica – Universidade de São Paulo
Av. Prof Mello de Morais, 2231, CEP 05508-900
São Paulo, SP, Brazil

Emails: victorsilva@usp.br, rodrigopolastro@usp.br, fgcozman@usp.br

Abstract— One of the central steps in building a pattern recognition system is the choice of the language that is employed to represent patterns and their classes — typical schemes involve rules, posterior probabilities, decision trees. In this paper we investigate the performance of *Markov logic*, a general language that combines first-order logic and probabilities, in the realm of pattern recognition; more specifically, in supervised learning of classifiers. We develop the theory necessary for this application of Markov logic, and present results with real data.

Keywords— Pattern recognition, supervised learning, Markov logic networks.

Resumo— Uma dos principais passos na construção de um sistema de reconhecimento de padrões é a escolha da linguagem para representar padrões e suas classes — esquemas típicos envolvem regras, probabilidades posteriores, árvores de decisão. Neste artigo investigamos o desempenho de *lógica de Markov*, uma linguagem geral que combina lógica de primeira-ordem e probabilidades, em reconhecimento de padrões; mais especificamente, no aprendizado supervisionado de classificadores. Desenvolvemos a teoria necessária para esta aplicação de lógica de Markov, e apresentamos resultados com dados reais.

Palavras-chave— Reconhecimento de padrões, aprendizado supervisionado, redes lógicas de Markov.

1 Introduction

To attain its goals, an intelligent system must select actions based on its past experiences. A crucial task in such systems is to classify data collected by sensors, so as to extract useful information out of them. In this paper we are interested in pattern recognition tasks (that is, to classify the nature of an observed pattern) where a classifier is learned from labeled data (Devroye et al., 1997; Jain et al., 1999). That is, we focus on supervised techniques: we are given a collection of labeled (pre-classified) patterns called *class labels*, and the problem consists of labeling a newly encountered, and yet unlabeled, pattern. We wish to focus on statistical learning of supervised classifiers (Hastie et al., 2001) — thus we wish to consider classifiers that have a strong probabilistic basis.

There are many "languages" that can be used to specify a classifier. Decision trees offer one such language; Naive Bayes classifiers offer a different language. Experience has shown that a trade-off is necessary in practice: if the underlying "language" leaves too many parameters free, the problem of *overfitting* appears; if the underlying "language" is too rigid, it may not be possible to capture relevant aspects of patterns (Friedman, 1997).

In the last decade a large number of statistical models has been proposed by combining well-known statistical methods with fragments of first-order logic. These models have appeared under the banners of (among others) *inductive logic programming* and *probabilistic relational models*; their purpose is to allow structured descriptions of observed data. The potential complexity of learned descriptions depends on the underlying language that is adopted. Thus one may adopt Horn clauses as the representation language (Lavrac and Dzeroski, 1994; Muggleton and Raedt, 1994), leveraging on their adequate tradeoff between complexity and expressivity. A different approach is to start with a language that is close to relational databases and statistical models, and to focus on learning relational structures from relational data (Dzeroski and Lavrac, 2001; Getoor and Taskar, 2007).

A considerable number of proposals for "relational" learning rely on *graph-based* models inspired by Bayesian networks and Markov random fields. For instance, *relational Bayesian networks* employ relations as nodes of a Bayesian network (Jaeger, 1997; Jaeger, 2001). *Markov logic networks (MLNs)* instead employ first-order formulas as the nodes of a Markov random field (Richardson and Domingos, 2006). These graph-based models are much more expressive than their propositional counterparts, and they have been used successfully in bioinformatics, language processing and diagnosis, to name a few applications (Getoor and Taskar, 2007).

Markov logic is a very flexible language that allows one to handle many aspects of real-world problems in a single representation. Inference algorithms have been implemented in an open tool called *Alchemy package*. It is currently a natural choice if one wishes to combine relational fragments of first-order logic with probabilities. How-

ever, the behavior of Markov logic as a basis for pattern recognition is not well explored. Due to the flexibility of the language, one can easily face overfitting, so the use of Markov logic is not entirely straightforward when learning classifiers.

In this paper we investigate the application of *Markov logic* to supervised learning of classifiers. We examine the necessary theory, propose sensible techniques to represent the dependencies between class and attributes, and describe experiments with real data. As it will be clear, our techniques offer a promising picture of Markov logic as a language for automatic pattern recognition.

Section 2 reviews relevant concepts about classification and knowledge bases. In Section 3 the overall problem and proposed approach are stated. Section 4 describes experiments and results obtained. The conclusion of the paper is left to Section 5.

2 Background

2.1 Classifiers and supervised learning

A classifier is a function $g(X_1, \ldots, X_N)$ from *attributes* to *labels*. The attributes are represented by variables X_i's, and the labels are the values that a *class variable* Y can assume. The goal, in constructing a classifier, is to obtain an estimator $\hat{Y} = g(X_1, \ldots, X_N)$ such that $\hat{Y} = Y$. So, to learn a classifier is to build a function g from data. This is often referred to as *training* the classifier. To evaluate classifiers, the usual metric is the *probability of error*, given by

$$\begin{aligned} e_g &= P(Y \neq g(\mathbf{X})) \\ &= E\left[I_{Y \neq g(\mathbf{X})}(\mathbf{X}, Y)\right]. \end{aligned} \tag{1}$$

The *empirical error rate* is

$$\hat{e}_g = \frac{1}{N} \sum_{i=1}^{N} I_{Y \neq g(\mathbf{X})}(\mathbf{X}, Y). \tag{2}$$

The optimal classifier is given by

$$\begin{aligned} g^* &= \arg\min_g e_g \\ &= \arg\min_g P(Y \neq g(\mathbf{X})). \end{aligned} \tag{3}$$

These expressions use the joint distribution $P(\mathbf{X}, Y)$. When the joint distribution is not available, it can be estimated from collected data, and the classifier is built using estimates of the joint distribution. Alternatively, the classifier can be directly built from data, without resort to the expression of the optimal classifier (Devroye et al., 1997).

Usually a classifier is learned using only a portion of available data, the *training data*. The remainder of the data are used to test the classifier (e.g., by estimating the probability of error), is the *testing data*. Testing data is important to detect *overfitting*; that is, the situation where a classifier is excellent for the training data but fails for other data. If no testing data are available, at least *cross-validation* must be used, in which one proceeds by separating a fraction of the data for testing, then repeating over the whole database. Five-fold or ten-fold cross-validation are very common procedures.

2.2 Basics of Markov Networks

A *Markov network* (MN) (also known as a *Markov random field*) is a statistical model for the joint probability distribution of a set of variables $X = (X_1, X_2, \ldots, X_n) \in X$ (Pietra et al., 1997). A MN 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 clique is a set of completely interconnected nodes, and a potential function is a non-negative real-valued function over the variables of the corresponding clique. The joint distribution represented by a MN is given by

$$P(X = x) = \frac{1}{Z} \prod_k \phi_k\left(x_{\{k\}}\right), \tag{4}$$

where x_k is the state of the kth clique (i.e., the state of the variables that appear in that clique). The denominator Z, known as the *partition function*, is given by

$$Z = \sum_{x \in \chi} \prod_k \phi_k\left(x_{\{k\}}\right).$$

MNs are often conveniently represented as *log-linear models*, with each clique potential replaced by a exponentiated weighted sum of *features* of the state, leading to

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_j w_j f_j(x)\right). \tag{5}$$

A feature may be any real-valued function of the state. There is one feature corresponding to each possible state x_k of each clique, with its weight being $\log \phi_k\left(x_{\{k\}}\right)$. This representation is exponential on the size of the cliques. However we are allowed 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).

Maximum a posteriori inference in MNs involves finding the most likely state of a set of query variables given the state of a set of evidence variables. Conditional inference involves computing the distribution of the query variables given the evidence. The most widely employed approximate solution to this problem is Markov chain Monte Carlo (MCMC) methods (Gilks, 1995).

2.3 First-Order Knowledge Bases

A *first-order knowledge base* (KB) is a set of sentences or formulas in first-order logic (Genesereth and Nislsson, 1987). Formulas are constructed using four types of symbols: constants, logical variables, functions and predicates. Constant symbols represent objects in a *domain* of interest (e.g., people: `Anna`, `Bob`, `Chris`, etc.). Logical variables 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`).

A *term* is any expression representing an object in the domain; it can be a constant, a logical 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))`). A *ground term* is a term containing no variables. A *ground atom* or a *ground predicate* is an atomic formula all of whose arguments are ground terms. Formulas are recursively constructed from atomic formulas using logical connectives and quantifiers. Some of the logical connectives in first-order logic, and their representation in the Alchemy package, are: not (`!`), and (`^`), or (`v`) implies (`=>`), if and only if (`<=>`).

A *positive literal* is as an atomic formula; a *negative literal* is a negated atomic formula. A KB in clausal form is a conjunction of *clauses*, a clause being a disjunction of literals. A *possible world* assigns a truth value to each possible ground atom.

In finite domains, first-order KBs can be propositionalized by grounding relations and functions, and by replacing each universally (existentially) quantified formula with a conjunction (disjunction) of all its groundings. In this paper we assume that domains are finite, as usually assumed in current work on Markov logic (recent investigations have tried to remove this restriction (Singla and Domingos, 2007)).

3 Markov Logic Networks for Classification

A first-order KB can be thought as set of hard constraints on the set of possible worlds. Consequently, if a world violates even one formula, it has probability equal to zero. The idea in Markov logic networks (MLNs) is to soften these constraints in a manner that when a world violates one formula in the KB it becomes less probable. 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.

An MLN (Richardson and Domingos, 2006) is defined as 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 $C = \{c_1, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ (Equations 4 and 5) as: (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 the F_i in L.

There is an edge between two nodes of $M_{L,C}$ if the corresponding ground predicates appear together in at least one grounding of one formula in L. From the definition of MLN and from Expressions 4 and 5, the probability distribution over possible worlds x specified by the ground MN $M_{L,C}$ is given by

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{i}^{F} w_i n_i(x)\right), \qquad (6)$$

where F is the number of formulas in the MLN and $n_i(x)$ is the number of true groundings of F_i in x. As formula weights increase, an MLN converges to a purely logical KB, being equal to one in the limit of all infinite weights (Richardson and Domingos, 2006).

Assuming domain closure and finiteness, we have that propositionlized MLNs are finite. In this case, the groundings of a formula are built simply by replacing its variables with constants in all possible combinations.

As a didactic example, consider a pattern recognition task involving two relevant features and a variable denoting the class. Assume the following formulas to be given:

$\forall$`x` $\texttt{Feature}_1\,(\texttt{x}) \Rightarrow \texttt{Class}\,(\texttt{x})$,
$\forall$`x` $\texttt{Feature}_2\,(\texttt{x}) \Rightarrow \texttt{Class}\,(\texttt{x})$,
$\forall\texttt{x}\forall\texttt{y}\ \texttt{Relation}\,(\texttt{x},\texttt{y}) \Rightarrow (\texttt{Feature}_2\,(\texttt{x}) \Leftrightarrow \texttt{Feature}_2\,(\texttt{y}))$.

If we have the constants $C =$ {`A`, `B`}, the MLN L yields the following features in the grounded MN $M_{L,C}$:

$\texttt{Feature}_1\,(\texttt{A}) \Rightarrow \texttt{Class}\,(\texttt{A})$,
$\texttt{Feature}_2\,(\texttt{A}) \Rightarrow \texttt{Class}\,(\texttt{A})$,
$\texttt{Feature}_1\,(\texttt{B}) \Rightarrow \texttt{Class}\,(\texttt{B})$,
$\texttt{Feature}_2\,(\texttt{B}) \Rightarrow \texttt{Class}\,(\texttt{B})$,
$\texttt{Relation}\,(\texttt{A},\texttt{B}) \Rightarrow (\texttt{Feature}_2\,(\texttt{A}) \Leftrightarrow \texttt{Feature}_2\,(\texttt{B}))$
$\texttt{Relation}\,(\texttt{B},\texttt{A}) \Rightarrow (\texttt{Feature}_2\,(\texttt{B}) \Leftrightarrow \texttt{Feature}_2\,(\texttt{A}))$
$\texttt{Relation}\,(\texttt{A},\texttt{A}) \Rightarrow (\texttt{Feature}_2\,(\texttt{A}) \Leftrightarrow \texttt{Feature}_2\,(\texttt{A}))$
$\texttt{Relation}\,(\texttt{B},\texttt{B}) \Rightarrow (\texttt{Feature}_2\,(\texttt{B}) \Leftrightarrow \texttt{Feature}_2\,(\texttt{B}))$

Figure 1 depicts the resulting propositionalized Markov network, where `C(U)` stands for `Class(U)`,

$\texttt{F}_\texttt{j}\,(\texttt{U})$ stands for $\texttt{Feature}_\texttt{j}(\texttt{U})$ and $\texttt{R}(\texttt{U},\texttt{V})$ stands for $\texttt{Relation}(\texttt{U},\texttt{V})$.

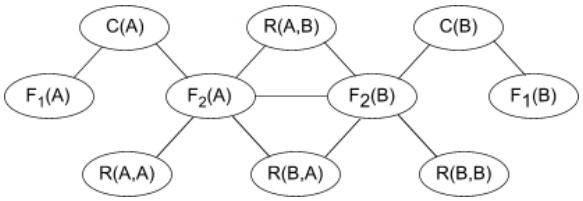

Figure 1: Propositionalized Markov network for a set of constants $C = \{\texttt{A},\ \texttt{B}\}$.

More details about MLNs can be found elsewhere (Richardson and Domingos (2006), Poon and Domingos (2006), Singla and Domingos (2006), Singla and Domingos (2005)).

Applying Markov logic in classification requires some care. An obvious idea is to build a classifier by learning weights for all formulas such as

$$C \ \Rightarrow \ A_1 \wedge A_2 \wedge \ldots \wedge A_n,$$

where we have a conjunction of atributes as an implication of the class (that is, by learning probabilities for Horn clauses). The Alchemy package allows one to specify, in one sentence, the set of all conjunctions over all attributes, thus it is easy to explore this alternative. However, as n grows the approach becomes unmanageable as it requires estimation of an exponential number of probability values. Indeed, if each atribute has m different values and we have c possible classes, Alchemy learns $c \cdot m^n$ parameters from data — overfitting then takes over and leads to poor classifiers.

A more concise alternative, and one that seems natural at first, is to learn probabilities for Horn clauses that mimic the structure of a Naive Bayes classifier:

$$\begin{aligned} C \ &\Rightarrow \ A_1 \\ C \ &\Rightarrow \ A_2 \\ &\ldots \\ C \ &\Rightarrow \ A_n \end{aligned}$$

The idea is to learn the probability that a single attribute implies the class, for each attribute. Taking the same example as in the last paragraph, we now have only $n\,(c \cdot m)$ parameters to learn. Despite the simplicity of this approach, experiments showed it to produce very poor classifiers. Further analysis revealed it to be actually far from a Naive Bayes classifier, because weights of formulas like $C \Rightarrow A_i$ are not related at all with the conditional probability of $P(A_i|C)$. Indeed, the probability of an implication to be true tells us nothing about the conditional probabilities, as the formula is always true if $\neg C \vee A_i$. Even though Park (2002) has proposed a way to express conditional probabilities from weighted clauses, it does not seem possible to create sensible classifiers on the basis of learning probabilities for Horn clauses.

The best results we obtained with MLN have been produced instead with conjuntions of groundings of the class and attributes, as we can see in the following formulas:

$$\begin{aligned} C \ &\wedge \ A_1 \\ C \ &\wedge \ A_2 \\ &\ldots \\ C \ &\wedge \ A_n \end{aligned}$$

Thus, the methodology adopted to express and validate a classifier in Markov logic based on these formulas is defined as follows:

- Define the class label variable;
- Extract suitable and meaningful features, i.e., the relevant attributes to avoid model incorrectness;
- Convert features into representative first-order predicates;
- Build conjunctions between pairs of class predicates and feature predicates;
- Build the traning and testing databases;
- Run generative weight learning;
- Cross-validate the classifier model obtained.

To understand this methodology, consider the following example. A database usually contains rows with observed values for attributes and class. Each row contains observed values for attributes that can be denoted by variables $X_1, \ldots, X_N$, where N is the problem's dimension. Suppose a database D contains a hundred rows with two binary features denoted by $feature_i$, $feature_j$, and with a *class* variable. A classifier representation for this example would then be constructed with three predicates and three logic formulas. In the syntax of the *Alchemy* tool, it would be as follows:

```
feature_i(row, value_i!)
feature_j(row, value_j!)
class(row, value_c!)
*class(row, +value_i) ^ *feature_i(row, +value_i)
*class(row, +value_j) ^ *feature_j(row, +value_j)
*class(row, +value_k) ^ *feature_k(row, +value_k)
```

The variable `row` ranges over each row number in database, and `value_i`, `value_j` range over the value the respective attribute can assume. The operator `!` placed after a variable allows one to specify variables that have mutually exclusive and exhaustive values. For example, in `class(row, value!)`, the `!` means that any row has exactly one class label value. When predicates in a formula are preceded by `*`, all possible ways in which `*` can be replaced by `!` are considered. If multiple variables are preceded by `+`, a weight is learned for each combination of their values in the formula. Together, formulas, their symbols and operators constitute the basis to construct (*grounding* and structure) a MN (see Richardson and

Table 1: Classification results (in %) for MLNC, rule, tree and Bayes-based classifiers. In Weka, information about standart deviation in classification accuracy results is not available.

UCI database	#lab	Rules			Trees			Bayes nets		MLNC
		ZeroR	DT	Prism	SC	DS	J48	NB	TAN	
Adult	1500	75.11	85.36	–	–	75.10	85.73	82.84	85.98	82.90 ± 1.35
Ionosphere	351	64.10	87.18	84.90	88.60	82.90	90.31	90.31	92.59	89.76 ± 3.75
Nursery	3087	34.20	96.00	99.52	99.58	67.98	98.07	92.49	96.45	90.96 ± 1.13
Breast Cancer (Diag.)	569	62.74	94.20	92.27	95.25	90.51	95.43	95.61	96.31	$\mathbf{95.79 \pm 3.22}$
Image Segmen.	210	14.28	83.81	87.62	93.33	28.10	93.81	89.05	95.71	75.24 ± 2.71
Car Evaluation	1728	70.02	91.49	89.70	97.11	70.02	92.36	85.53	94.61	88.25 ± 6.26
Balance Scale	609	45.76	71.04	37.44	69.6	56.32	69.6	70.72	71.36	$\mathbf{95.08 \pm 3.18}$
Iris	150	33.33	94.00	92.67	94.00	66.67	94.00	94.00	94.67	$\mathbf{95.33 \pm 6.32}$
Monk's	556	48.39	89.52	90.32	80.64	73.39	82.26	77.42	95.97	72.26 ± 7.12
Wine	178	39.89	96.08	95.51	93.82	58.43	93.82	98.87	98.31	$\mathbf{97.22 \pm 5.40}$

Domingos (2006, Table I, Figure 1), for details on this construction). After expressing an MN model of the database in Markov logic, the next step would be to build appropriate training and testing databases. Generative weight learning could be done afterwards. At last, ten or five-fold cross-validation should be perfomed.

4 Experiments

To investigate the performance of Markov logic classifiers, we have performed experiments with some of the UCI[1] databases. For inference and weight learning we have used an open software tool *Alchemy* (Kok et al., 2005), which provides a series of algorithms for statistical relational learning and probabilistic inference.

The methodology for tests was organized as follows: (1) specification of the problem's model (relational MN) in Markov logic; (2) preprocessing of database consisting of discretizing continuous variables (as Alchemy only works with categorical/discrete data), converting instances of variable into *atoms* (first-order logic) and building of appropriate training and testing data; (3) generative learning of weights associated to formulas in the model; and (4) running inference algorithms on query variables.

Table 1 summarizes the accuracy of our classifiers, denoted by MLNC (Markov Logic Network Classifier), and of other state-of-art classification methods. They are: Naive Bayes (NB), Tree-Augmented Naive Bayes (TAN), J48, Prism, Decision Table (DT), Simple Cart (SC), Decision Stump (DS) and ZeroR (these algorithms are included in *Weka* (Waikato Environment for Knowledge Analysis) (Witten and Frank, 1999)). The UCI databases used in our experiments were: *Adult*, *Breast Cancer Iris*, *Wine*, all of them amongst the most popular databases in the literature; and also the *Balance Scale*, *Car Evaluation*, *Image segmentation*, *Ionosphere*, *MONK's problems* and *Nursery*. Some of this databases were trimmed to reduce computation time. In this reduction the classes proportion was kept intact. The discretization of the databases was done with a supervised discretization algorithm included in *Weka*.

The results in the table for our classifier, MLNC, were obtained with ten-fold cross-validation over the selected databases. In comparison with the rule-based classifiers, MLNC presented superior accuracy values for most databases. Regarding tree-based classifiers, MLNC had better or very similar results in 5-out-of-10 databases. In comparison with Bayesian network classifiers such as Naive Bayes and TAN. MLNC presented competitive results. In several cases MLNC surpassed the TAN classifiers, generally considered the best available Bayesian network classifiers (Cohen et al., 2004; Friedman et al., 1997).

5 Conclusion

The results presented in this paper suggest that Markov logic is a flexible and useful contender as a language for describing patterns and classifiers. Supervised classifiers based on Markov logic are on a par with the state-of-art classifiers, sometimes surpassing their performance on real data. Besides, the main advantage of Markov logic classifiers seems to be the possibility of encoding deterministic relations about the domain of interest. Again, one must be careful to avoid an explosion of parameters to be learned while specifying a Markov logic classifier; the method described in Section 3 is the main contribution here.

As future work we intend to explore Markov logic in *semi-supervised learning under constraints*; that is, in situations where data may be related by deterministic constraints (Basu et al., 2004; Law et al., 2005; Shental et al., 2003). Markov logic is particularly well suited in this context as it can easily model the deterministic constraints within the language itself.

[1] Available in http://archive.ics.uci.edu/ml/

Acknowledgments

This work has received generous support from HP Brasil R& D. The third author is partially supported by CNPq.

References

Basu, S., Bilenko, M. and Mooney, R. J. (2004). A probabilistic framework for semi-supervised clustering, *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM Press, New York, NY, USA.

Cohen, I., Cozman, F. G., Sebe, N., Cirelo, M. C. and Huang, T. S. (2004). Semisupervised learning of classifiers: theory, algorithms, and their application to human-computer interaction, *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Devroye, L., Györfi, L. and Lugosi, G. (1997). *A Probabilistic Theory of Pattern Recognition (Stochastic Modelling and Applied Probability)*, Springer.

Dzeroski, S. and Lavrac, N. (2001). *Relational Data Mining*, Springer, Berlin.

Friedman, J. H. (1997). On bias, variance, 0/1-loss, and the curse-of-dimensionality, *Data Mining and Knowledge Discovery* **1**(1): 55–77.

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

Genesereth, M. R. and Nislsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers.

Getoor, L. and Taskar, B. (2007). *Introduction to Statistical Relational Learning*, MIT Press.

Gilks, W. R. (1995). *Markov Chain Monte Carlo in Practice*, Chapman & Hall/CRC.

Hastie, T., Tibshirani, R. and Friedman, J. H. (2001). *The Elements of Statistical Learning*, Springer.

Jaeger, M. (1997). Relational Bayesian networks, *in* D. Geiger and P. P. Shenoy (eds), *Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Francisco, California, pp. 266–273.

Jaeger, M. (2001). Complex probabilistic modeling with recursive relational Bayesian networks, *Annals of Mathematics and Artificial Intelligence* **32**: 179–220.

Jain, A. K., Murty, M. N. and Flynn, P. J. (1999). Data clustering: a review, *ACM Computing Surveys* **31**(3): 264–323.

Kok, S., Singla, P., Richardson, M. and Domingos, P. (2005). The alchemy system for statistical relational AI, *Technical report*, Department of Computer Science and Engineering, University of Washington.

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, New York.

Law, M. H. C., Topchy, A. P. and Jain, A. K. (2005). Model-based clustering with probabilistic constraints, *SIAM International Data Mining Conference*.

Muggleton, S. and Raedt, L. D. (1994). Inductive logic programming: Theory and methods, *Journal of Logic Programming* **20**: 629–679.

Park, J. D. (2002). Using weighted max-sat engines to solve mpe, *Eighteenth national conference on Artificial intelligence*, American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 682–687.

Pietra, S. D., Pietra, V. D. and Lafferty, J. (1997). Inducing features of random fields, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(4): 380–393.

Poon, H. and Domingos, P. (2006). Sound and efficient inference with probabilistic and deterministic dependencies, *AAAI*, AAAI Press.

Richardson, M. and Domingos, P. (2006). Markov logic networks, *Machine Learning* **62**(1-2): 107–136.

Shental, N., Bar-Hillel, A., Hertz, T. and Weinshall, D. (2003). Computing Gaussian mixture models with EM using equivalence constraints, *NIPS*.

Singla, P. and Domingos, P. (2005). Discriminative training of Markov logic networks., *AAAI*, AAAI Press / The MIT Press.

Singla, P. and Domingos, P. (2006). Memory-efficient inference in relational domains, *AAAI*, AAAI Press.

Singla, P. and Domingos, P. (2007). Markov logic in infinite domains, *Conference on Uncertainty in Artificial Intelligence*, AUAI Press, pp. 368–375.

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann.