KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING INFORMATICA
Celestijnenlaan 200 A — B-3001 Leuven

TRADING EXPRESSIVITY FOR EFFICIENCY IN STATISTICAL RELATIONAL LEARNING

Promotor :
Prof. Dr. L. DE RAEDT

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Niels LANDWEHR

February 2009

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING INFORMATICA
Celestijnenlaan 200 A — B-3001 Leuven

TRADING EXPRESSIVITY FOR EFFICIENCY IN STATISTICAL RELATIONAL LEARNING

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Niels LANDWEHR

U.D.C. 681.3*I26

February 2009

D/2009/7515/14
ISBN 978–94–6018–031–6

Abstract

Statistical relational learning (SRL) combines state-of-the-art statistical modeling with relational representations. It thereby promises to provide effective machine learning techniques for domains that cannot adequately be described using a propositional representation. Driven by new applications in which data is structured, interrelated, and heterogeneous, this area of machine learning has recently received increasing attention.

However, combining statistical modeling and relational representations also poses new challenges. There is a trade-off between the expressivity of a machine learning formalism and its computational efficiency, as a higher expressivity entails a larger search space during learning. Propositional machine learning techniques are at one end of this trade-off, while approaches that combine the full power of statistical and relational learning are at the other end. This thesis presents a collection of simple SRL techniques that focus on computational efficiency rather than maximum expressivity, and thereby occupy an intermediate position in the outlined expressivity-efficiency trade-off.

The thesis has three main contributions. We first introduce *dynamic propositionalization* approaches, which provide a simple but principled integration of relational and statistical learners. Dynamic propositionalization is shown to outperform more traditional static propositionalization approaches, while maintaining computational efficiency. A second part presents *Markov models for relational sequences*, where sequence elements can be logical atoms or complete logical interpretations. By restricting attention to fully observable data and employing a Markov assumption, inference and learning in the resulting formalisms is significantly easier than in more general SRL systems. In a final part, we present two structured probabilistic models that are tailored to particular application domains, namely *haplotype reconstruction* and *activity recognition*. These two domains could be modeled using general-purpose statistical relational sequence models; however, the restriction to a particular domain again allows us to derive more efficient special-purpose inference and learning algorithms.

The approaches presented throughout the thesis are evaluated in several relational real-world domains, including structure-activity prediction for chemical compounds, web page classification, modeling user behavior in mobile phone networks, and modeling massively multiplayer online games.

Acknowledgments

Working on this Ph.D. thesis has been a wonderful and exciting experience. I would not have been able to do the work I did and successfully finish this project if it had not been for the help, support, and advice of many people both within the university and beyond. First and foremost, I would like to thank my supervisor Luc De Raedt for his constant advice and support during the last four and a half years. He not only helped me with many technical questions and problems, but, more importantly, also showed me how science works in general: how to write papers and how to read them, how to give presentations and collaborate with fellow scientists, what is important and what is not. Whenever I got bogged down in the technical details of a problem, Luc helped me to concentrate on the general picture and identify the main issues that needed to be solved. He also actively encouraged me to develop my own ideas and venture into different subfields of machine learning, and helped me sort out what I had learned afterwards. Finally, Luc put me in touch with several other excellent researchers from around the world, with whom a fruitful collaboration developed.

My colleagues in the machine learning group, initially the smaller group in Freiburg and later the larger one in Leuven, also influenced and contributed to my work. Specifically, I would like to thank Kristian Kersting for the advice and guidance in particular during the early phase of my Ph.D. work. Working together with Ingo Thon, Bernd Gutmann, and Andreas Karwath has also been both fun and productive, and I am looking forward to continuing these collaborations. Björn Bringmann, Albrecht Zimmermann and Siegfried Nijssen helped me by giving me a different perspective on the field of machine learning in many interesting discussions.

A large part of my work has been done in close collaborations with other research groups from around the world. This has not only given me the opportunity to gather much new scientific knowledge, but also allowed me to go on enjoyable trips and make true friends. Specifically, I would like to thank Andrea Passerini and Paolo Frasconi for all the help they gave me with one of the main projects of the thesis, and many great evenings enjoying Italian cuisine. A significant part of my work was also done in collaboration with the machine learning group in Helsinki, and I would like to thank Taneli Mielikäinen, Heikki Manila, and Hannu Toivonen for many new insights and a great time in Helsinki. The collaboration with Matthai Philipose from Intel Research Seattle was also very fruitful.

Thanks are also due to all members of the general DTAI group in Leuven, specifically Maurice Bruynooghe, who leads the group, and my office mates Ingo, Kurt, Robby, Laura, and Martijn; the working environment in Leuven was superb and the size of the group meant that there was always an interesting presentation or discussion to join. I also thank Hendrik Blockeel, Johan Suykens, Paolo Frasconi and David Page for serving on my Ph.D. committee and their excellent comments on the initial version of this thesis text, and Dirk Vandermeulen for chairing my defense.

I gratefully acknowledge the financial support received for the work performed during this thesis from the following organizations: European Union FP6-508861 project on "Applications of Probabilistic Inductive Logic Programming II", the research foundation Flanders (FWO-Vlaanderen), the BOF-DOC grant of the K. U. Leuven, and the GOA/08/008 project on "Probabilistic Logic Learning".

Last but not least, I thank my family and personal friends for all their support and encouragement. At times of intensive work, dealing with me was probably not always easy. Special thanks go to Sybille, who has not only proofread this text but also managed to drag me away from my desk and cheer me up when it seemed there was nothing in life except work. Your support during the final months of writing up the text and defending the thesis was invaluable.

Niels Landwehr
Leuven, Belgium, January 2009

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

X	Random variable
$\mathbf{X}$	Set of random variables
$P(X)$	Distribution over random variable X
$P(x)$	Probability of value x
$\mathbb{E}(X)$	Expectation of random variable X
X	Logical variable
p/k	Predicate symbol of arity k
$p(t_1, ..., t_k)$	Logical atom
Σ	First-order alphabet
$hb(\Sigma)$	Herbrand base of the alphabet Σ
I	Herbrand interpretation
θ	Logical substitution
q	Definite clause
$\mathbf{q}$	Random variable over definite clause q
H	First-order hypothesis
B	First-order background knowledge
$\mathcal{L}$	First-order language bias

x	Vector-valued (propositional) example
e	Relational example
E	Set of training examples
ρ	Refinement operator
$\varphi_{H,B}$	Propositionalization based on hypothesis H
λ	Statistical classifier
P_λ	Distribution/probability estimate given by λ
f	Margin function of support vector machine
$K(\cdot,\cdot)$	Propositional kernel function
$K(\cdot,\cdot,H,B)$	Relational kernel function defined in terms of hypothesis H
$\langle u,v\rangle$	Scalar product of vectors u and v
$\langle M,N\rangle_F$	Frobenius product of matrices M and N
$S(\Sigma)$	Set of all ground sequences over the alphabet Σ
$\mathbb{S}_n(\Sigma)$	Set of equivalence classes induced on $S(\Sigma)$ modulo n-congruence
$s \preceq_\theta s'$	Sequence s subsumes sequence s' with substitution θ
σ	Selection for a CPT-theory
g	Genotype
$g[m]$	Allele pair at marker m on genotype g
h	Haplotype
$h[m]$	Allele at marker m on haplotype h
$\mathcal{G}$	Set of genotypes
$\mathcal{H}$	Set of haplotypes

Overture

Chapter 1

Introduction

1.1 Artificial Intelligence and Machine Learning

Artificial Intelligence, or *AI*, is often seen as the long-standing dream of building intelligent machines than can perceive, think, and act in similar ways as human beings. While this dream continues to fascinate and drive research efforts, AI today is also an important subfield of computer science with significant real-world applications. It is concerned with building computer programs that solve problems which would typically require intelligence if solved by a human. While the original goal of building machines that exhibit general human-level intelligence remains elusive, AI as a problem-solving discipline has been remarkably successful. In fact, computer programs that exhibit intelligence at solving a particular task are in wide use today, and their capabilities and scope expand continuously. Examples include control software for robots, face recognition from camera images, text analysis and information retrieval on the world wide web, systems that automatically process credit applications, detect fraudulent financial transactions, or provide medical assistance and suggest treatment options based on a patient's diagnostic record (Moore, 1990; Lawrence *et al.*, 1997; Chen, 1995; Carter and Catlett, 1987; Fawcett and Provost, 1997; Magoulas and Prentza, 2001).

A key component of AI is *machine learning*. Machine learning is usually defined as the study of how to make computer *learn*, that is, automatically improve their behavior with experience (Mitchell, 1997). Machine learning is applicable in situations where humans are unable—or unwilling—to explicitly program a procedure for solving a given problem, but example solutions (or a more indirect feedback mechanism) are available. Consider, for instance, the problem of diagnosing patients with cancer, based on the patients' records including information on symptoms, X-ray scans or patient history. It is hard to explicitly write down a procedure that correctly predicts whether a patient has cancer or not based on the available information, but many example cases (patient records for which the correct diagnosis is known) are available. From these examples, machine

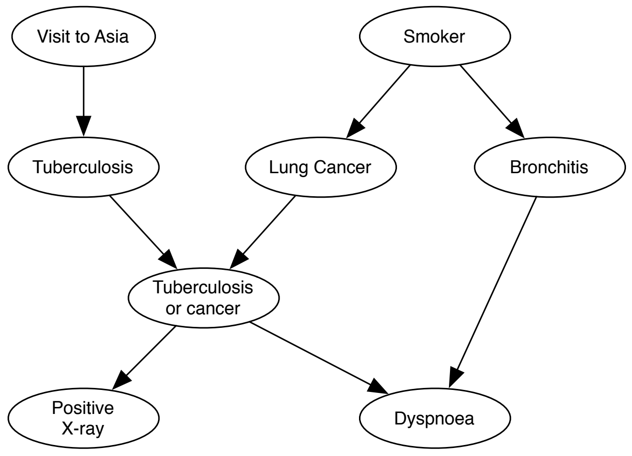


Figure 1.1: A Bayesian network model for lung cancer diagnosis, inspired by (Lauritzen and Spiegelhalter, 1988). The network relates diagnostic information, such as X-ray imaging or patient symptoms, to medical conditions, such as Tuberculosis or Cancer. Arrows indicate probabilistic influences between variables. See Section 2.1.1 for a more detailed discussion of Bayesian networks.

learning can automatically infer a *model*: a formal representation of the structure inherent in the data, which can, for example, be given by the joint probability distribution of all domain variables. Presented with a patient record, such a model can be used to predict the likelihood that this patient suffers from cancer (see Figure 1.1). In a similar manner, machine learning approaches can be used to learn how to classify email as legitimate or spam based on text statistics and user feedback, or to drive vehicles autonomously on public highways based on camera images and observing a human driver (Wu and Vapnik, 1999; Pomerleau, 1989).

A general definition of machine learning can be given as follows (Mitchell, 1997):

Definition 1.1.1 (Machine Learning)**.** *A computer program is said to* learn *from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

This definition not only encompasses a simple setup in which a computer program (or model) is learned once from a fixed set of example cases, but also a setting in which the program can continuously learn and improve its performance. The second setting has important applications in areas such as spam filtering, where a filter must continuously learn to adapt to new types of spam it will encounter over time (Gray and Haahr, 2004).

In addition to providing solutions for problems based on existing data or observations, machine learning techniques can also yield new insights into the domain under consideration. A learned model often summarizes the data in a helpful way, or uncovers regularities and associations which were previously unknown to the human observer. This process is often referred to as *knowledge discovery*. In this sense, machine learning is closely related to statistics and data analysis techniques such as *data mining*. Data mining aims at uncovering regularities in large data collections, such as sets of items appearing frequently together (Han and Kamber, 2001). In contrast to machine learning, data mining is typically focused more on the characterization of local structure in the data rather than providing a global model. Nevertheless, there are close connections and significant overlap between the two fields.

The rate at which data is collected and stored in our modern world is ever-increasing. The capacity of digital data storage has increased exponentially for many years (Fayyad and Uthurusamy, 2002), and the pervasiveness of modern IT infrastructure makes it possible to collect data at a very low cost. Moreover, modern science also generates vast amounts of experimental data, for example, in automated high-throughput experiments in biology. It becomes harder and harder for humans to sift through and understand this data without automation. There is a risk that data will remain in ”write-only” data storages and will never be used, or humans can only inspect a small part of the data and important connections will be missed. Machine learning and related data analysis methods are increasingly important for making sense of large and complex data collections in science, business, or the world wide web. They can easily surpass human capabilities in the sheer volume of data that can be analyzed.

To summarize, machine learning can both be used to solve a particular problem (for example, suggest a treatment for a patient based on a large set of medical records) and to gain insight into a domain (for example, understand how certain symptoms correlate with certain diseases). For the latter goal, *interpretability* of the learned model is of crucial importance—that is, the learned model should not be a black-box solution, but human experts should be able to inspect and understand both the model and how it makes predictions. As an example, consider the Bayesian network shown in Figure 1.1, that provides insight into the relationship between symptoms and diseases in a lung cancer diagnosis domain.

1.2 Statistical Relational Learning

Traditional machine learning is concerned with learning from independent examples represented in an attribute-value (or *propositional*) format. For instance, in the cancer domain described above, attributes are patient symptoms such as dyspnoea or a positive X-ray scan. Propositional learning has made much progress over the last decades, going from simple decision-rule induction to sophisticated and rigorous statistical modeling techniques such as probabilistic graphical models or support vector machines, which yield accurate models even in the presence of noise or uncertainty in the data (Bishop, 2006).

However, in many complex real-world domains a propositional representation is not appropriate, as instances are richly structured and/or interrelated. As an example, imagine that in the cancer diagnosis domain we are not only interested in the symptoms of an individual patient, but also in the patient's family history, as the likelihood of developing cancer is influenced by certain genetic traits shared with close relatives. In this case examples cannot be represented as simple attribute-value vectors, as the data exhibits a complex structure of relationships between patients, family members, and symptoms. Such *relational* domains are most easily described by (a subset of) first-order logic or related formalisms, for example, graphs and networks.

Learning from relational data has been extensively studied in the field of *Inductive Logic Programming* (ILP) (Muggleton and De Raedt, 1994). In ILP, logical representations are used for both the data and the model that is to be inferred from the data. More specifically, the goal of inductive logic programming is to characterize the data in terms of a logical theory (see Section 2.2 for more details). Advantages of using such a representation are that the learned model is easily understood by human experts, and background knowledge can be used to guide the learning process. ILP approaches have been successfully applied in a large number of relational domains, most notably in bio- and chemoinformatics (Bratko and Muggleton, 1995). On the negative side, classical ILP approaches do not take full advantage of recent advances in statistical machine learning. Accordingly, ILP methods are often not competitive with statistical modeling techniques if the available training data is noisy or only partially observable.

It is therefore not surprising that the study of *statistical relational learning* (SRL), that is, the combination of statistical modeling techniques with relational learning, has received much attention recently (Getoor and Jensen, 2000, 2003; Dietterich *et al.*, 2004; Fern *et al.*, 2006; De Raedt and Kersting, 2003; Getoor and Taskar, 2007; De Raedt *et al.*, 2008). Statistical relational learning promises to combine the advantages of the two constituent techniques, namely the expressivity and interpretability of ILP with the accuracy and robustness of statistical learning methods. This has been recognized as one of the central challenges in machine learning (Getoor and Taskar, 2007).

A large variety of statistical relational learning approaches have been developed recently (see Section 2.3 for a brief overview). A number of successful applications of SRL have also been reported, which can be attributed to the ability of SRL systems to model both rich relational structure and uncertainty (De Raedt *et al.*, 2008). However, the high expressivity in SRL comes at a computational price: there is an inherent trade-off between expressivity and efficiency in machine learning, as high expressivity leads to a large search space during learning. SRL approaches that try to combine the full power of statistical and relational learning can be seen as occupying one extreme in this expressivity-efficiency trade-off. Propositional learning systems are at the other extreme, as they are typically much more efficient but cannot handle the relational complexity of many real-world domains.

1.3 Thesis Contributions and Roadmap

The goal of this thesis is to explore approaches to statistical relational learning that occupy an intermediate position in the expressivity-efficiency trade-off inherent in machine learning. Simple statistical relational models will be developed that deliberately restrict expressivity, and thereby allow for more efficient learning and inference algorithms. This will extend the applicability of SRL techniques to application domains for which current approaches are computationally too demanding.

Different directions can be identified in existing approaches to SRL. The simplest category of existing approaches are *propositionalization* techniques. They map the relational instance space to a propositional space in which each instance is characterized by a fixed set of attribute-value pairs derived from its relational description, and apply statistical machine learning techniques in this simpler space (Kramer *et al.*, 2001). Two cases can be distinguished: instances can be either mapped to a small set of features obtained from a classical relational learning system, or (implicitly or explicitly) to a very large feature set. An example for the first case is post-processing of theories returned by ILP systems using probabilistic models (Pompe and Kononenko, 1995). An example for the second case are approaches that extract all frequent relational patterns from the data and use these as features in a statistical classifier (Kramer and De Raedt, 2001), or propositionalization systems such as Linus (Lavrac and Dzeroski, 1994) that use all features from a pre-defined language bias.

In both cases, the main drawback of propositionalization is that the construction of the mapping is decoupled from the actual statistical modeling. The feature set is thus not specifically optimized for the statistical classifier in which it will be used, and the relational part of learning cannot take advantage of advanced statistical techniques to effectively handle noise or guard against overfitting. In this sense, propositionalization avoids the problem of inducing a joint statistical-relational model, by decoupling it into the two separate steps of relational and statistical learning. Moreover, propositionalization systems typically need many features for accurate prediction, because the feature set is not specifically optimized. Therefore, one of the main advantages of relational learning is lost, namely that the final model is interpretable and yields insight into the domain under consideration. One of the major contributions presented in this thesis are *dynamic propositionalization* techniques that replace the two-step approach in existing (static) propositionalization systems by an integrated learning procedure, and typically induce small sets of interpretable first-order features. Such approaches have recently received significant attention (Landwehr *et al.*, 2005b; Davis *et al.*, 2005a; Landwehr *et al.*, 2007b; Rückert and Kramer, 2007).

More ambitious approaches to SRL directly define probabilistic models over relational data, often in a generative fashion. Most prominent are techniques based on *knowledge based model construction* (KBMC). Given relational data, they construct a ground probabilistic model by "unrolling" a probabilistic model template. Standard techniques for inference and (parameter) learning can then be used in the ground network. Examples for KBMC techniques include Probabilistic Relational Models (Koller, 1999), Bayesian Logic Programs (Kersting and De Raedt, 2001), Relational Bayesian Networks (Jaeger, 1997) and Markov Logic Networks (Richardson and Domingos, 2006). Furthermore, probabilistic models over relational data can also be defined via a (probabilistic) proof-based semantics, as in Stochastic Logic Programs (Muggleton, 1996) and PRISM (Sato and Kameya, 1997).

These more ambitious approaches try to combine the full power of statistical and relational modeling. While this allows for great expressivity and powerful probabilistic inference, their inherent complexity comes at a computational price. This is particularly relevant when learning models from data, which involves learning both the relational model structure and the model parameters. For parameter learning, it is relatively straightforward to extend standard algorithms for probabilistic models (such as the expectation-maximization algorithm) to KBMC approaches, but the resulting ground network can become very large. How structure learning can be addressed in an efficient and statistically sound way is often unclear. General structure learning algorithms, which take into account the statistical part of the model, are typically too inefficient to be applicable on real datasets. At the same time, the high expressivity of these formalisms requires large amounts of training data to reliably estimate the model structure. In most realistic settings, the structure therefore has to be pre-defined by the user. If structure learning is performed, it is typically again decoupled from the statistical modeling, for example, by running a more traditional ILP algorithm to infer a set of approximately correct rules (Richardson and Domingos, 2004). A notable exception is the heuristic approach by (Kok and Domingos, 2005), in which a simplified form of statistical learning is carried out to score candidate first-order models. If statistical and relational learning are decoupled, this has the same drawbacks as the static propositionalization approaches outlined above. As one promising direction to alleviate these problems, this thesis will discuss KBMC approaches which are tailored to a particular setting—such as fully observable, sequential data—or application domain. This significantly reduces expressivity compared to fully-fledged general-purpose SRL systems, but also yields significant computational savings.

The idea of trading expressivity for efficiency in statistical relational learning will be explored in different settings throughout the course of the thesis. While there will be a variety of techniques and problem settings considered, the general theme underlying all presented approaches is to provide principled but simple solutions for statistical relational learning problems. Specifically, the thesis is organized into three parts. Part I discusses statistical relational learning for classification, Part II statistical relational sequence models, and Part III *embedded* SRL approaches that are tailored to a particular application domain. In the following, we give a brief outline of the organization and main contributions of each part.

Part I proposes *dynamic propositionalization* techniques for relational classification problems. Dynamic propositionalization is an approach for tightly integrating statistical and relational learning, by jointly learning a small, interpretable set of first-order logical features and a statistical classifier in which these features are used. This contrasts with traditional (static) propositionalization approaches, where feature selection and statistical learning are decoupled. **Chapter 3** explores dynamic propositionalization based on probabilistic models. More specifically, we present the nFOIL and tFOIL systems. nFOIL integrates the simplest probabilistic model, naïve Bayes, with the simplest relational rule learner, FOIL. TFOIL extends the naïve Bayes model used in NFOIL to tree augmented naïve Bayes. **Chapter 4** explores dynamic propositionalization based on kernels. In the resulting KFOIL system, a relational kernel function is learned that is defined in terms of a small set of interpretable first-order features. Different learning settings and objective functions are discussed. In particular, we show how multi-task learning can be addressed by sharing the kernel function across tasks.

In both the graphical model and kernel setting, efficient incremental algorithms for dynamic propositionalization are derived that have little or no computational overhead compared to equivalent static propositionalization approaches. Empirically, we show that dynamic propositionalization outperforms both static propositionalization and (non-statistical) relational learning techniques in challenging real-world domains. Part I is based on material that has appeared in the following publications:

N. Landwehr, K. Kersting, and L. De Raedt. *Integrating naïve Bayes and FOIL*, Journal of Machine Learning Research 8, pp. 481-507, 2007.

N. Landwehr, A. Passerini, L. De Raedt and P. Frasconi. *kFOIL: Learning Simple Relational Kernels*, in Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006), Boston, Massachusetts, USA., 2006.

N. Landwehr, K. Kersting, and L. De Raedt. *nFOIL: Integrating naïve Bayes and FOIL*, in Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005), Pittsburgh, Pennsylvania, USA., 2005.

Part II deals with probabilistic models for relational sequences. We extend the simplest propositional sequence models, namely Markov chains, to the relational case. Two settings are considered: sequence elements can be individual logical facts, or complete logical interpretations. **Chapter 5** considers extensions of n*-gram* models to sequences of logical facts. n-grams are simple but effective propositional sequence models based on mixtures of Markov chains. The resulting r*-gram* model defines a Markov chain where smoothed distributions can be obtained by decreasing the order of the Markov chain as well as by relational generalization of the r-gram. To avoid sampling object identifiers in sequences, r-grams are generative models only at the level of variablized sequences with local object identity constraints, and thereby abstract away from a particular set of identifiers occurring in the sequence. **Chapter 6** considers Markov chains over sequences of complete logical interpretations. Transition probabilities are defined using CP-logic, an expressive probabilistic logic for modeling causality. By restricting CP-logic to the sequential case, employing a Markov assumption, and requiring fully observable data, inference and learning become more tractable than in general CP-logic. Moreover, efficient special-purpose algorithms for inference and learning based on binary decision diagrams are developed.

The models presented in Part II are applied to modeling traces of user behavior in different domains, such as mobile phone usage and massively multiplayer online games. Part II is based on material that has appeared in the following publications:

I. Thon, N. Landwehr and L. De Raedt. *A Simple Model for Sequences of Relational State Descriptions*, in Proceedings of the 19th European Conference on Machine Learning (ECML-2008), Antwerp, Belgium, 2008.

N. Landwehr and L. De Raedt. *r-grams: Relational Grams*, in Proceedings of the Twentieth Joint International Conference on Artificial Intelligence (IJCAI-2007), Hyderabad, India, 2007.

Part III is concerned with *embedded* SRL approaches: systems that employ statistical relational learning principles, but are tailored to a particular application domain. In this setting, the generality of the underlying SRL framework is sacrificed for a more efficient system restricted to perform a particular task in a particular domain. We introduce two application-specific models, namely for the *haplotyping* and *activity recognition* problems. These application domains exhibit a structure that can easily be represented using relations, although the data under consideration is not explicitly relational. Accordingly, the two proposed models can be represented as statistical relational models. Specifically, they are instances of Logical Hidden Markov Models, an SRL framework upgrading hidden Markov models to the relational case (Kersting *et al.*, 2006). However, by specializing to a particular application domain—and thus, a subclass of models—domain-specific inference and learning algorithms can be derived which are orders of magnitude more efficient than the general-purpose algorithms employed in LoHMMs.

Specifically, **Chapter 7** proposes an application-specific model for the haplotyping domain based on sparse hidden Markov models. For this model a domain-specific structure learning algorithm can be derived, which, in contrast to the general-purpose LoHMM structure learning algorithm, provides efficient structure learning for real-world haplotyping datasets. In terms of reconstruction accuracy the resulting system is competitive with state-of-the-art haplotyping systems developed in the bioinformatics community. **Chapter 8** presents an application-specific structured probabilistic model for activity recognition. The model is tailored towards recognition of *interleaved* activities, in which observations are generated by several interleaving Markov processes. Again, the model can be represented as a logical hidden Markov model. However, domain-specific approximate inference techniques are required to achieve the necessary efficiency for real-world activity recognition scenarios. Specifically, we will employ extensions of the chainwise Viterbi algorithm used in factorial hidden Markov models (Ghahramani and Jordan, 1997). Part III is based on material that has appeared in the following publications:

N. Landwehr. *Modeling Interleaved Hidden Processes*, in Proceedings of the 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland, 2008.

N. Landwehr, T. Mielikäinen, L. Eronen, H. Toivonen, and H. Mannila. *Constrained Hidden Markov Models for Population-based Haplotyping*, BMC Bioinformatics 8 (Suppl. 2), 2007.

Finally, some of the work that was performed during the course of my Ph.D. research has not been included in this thesis text. It will be briefly summarized now. In the context of the activity recognition problem discussed in Chapter 8, we also investigated the use of *relational transformation-based tagging* approaches to several activity recognition tasks. This approach was prompted by the observation that activity recognition problems are similar to tagging problems in natural language processing, as a given sequence of observations needs to be tagged with an activity label at every point in time. Based on inductive logic programming principles, we extended the work on transformation-based tagging by Brill (1995) to deal with relational representations. The approach has been shown to be competitive with probabilistic methods such as hidden Markov models. More details can be found in (Landwehr *et al.*, 2008, 2007c). Some additional work was also performed in the haplotyping domain discussed in Chapter 7, where we investigated how to *combine* haplotype reconstructions produced by different haplotyping approaches. This can be achieved by defining appropriate distance functions between haplotype pairs, and corresponding algorithms to compute mean or median reconstructions based on these distances. More details can be found in (Kääriäinen *et al.*, 2007; Landwehr and Mielikäinen, 2008). The work on modeling sequences of interpretations described in Chapter 6 employs massively multiplayer online games as a benchmark domain. Further experiments in this domain within a *sequence classification* setting were carried out using *relational sequence alignments* (Karwath *et al.*, 2008). Finally, earlier work on *Logistic Model Trees* (see Landwehr *et al.*, 2003) was extended, resulting in the publication (Landwehr *et al.*, 2005a).

Chapter 2

Statistical and Relational Machine Learning

This chapter provides some background on Statistical Relational Learning (SRL). We roughly categorize existing work in the field, thereby providing a context for the contributions of the thesis. The chapter is organized as follows: we first present a brief review of (1) the statistical perspective on machine learning and (2) relational learning techniques in Section 2.1 and Section 2.2. Afterwards, Section 2.3 discusses different classes of existing SRL techniques: propositionalization, knowledge-based model construction, and proof-based models. We also outline some of the open questions and challenges in the field. Finally, we propose a learning setting that extends inductive logic programming towards a joint statistical-logical model in Section 2.4.

2.1 Statistical Machine Learning

This section will briefly outline some fundamental concepts of statistical machine learning, and introduce notation and terminology that will be used throughout the thesis.

The general setup in statistical machine learning is as follows. We are given a set $E = \{x_1, ..., x_N\}$ of *training examples*, which are points in an M-dimensional *instance space* $\mathcal{X} = D_1 \times ... \times D_M$, where D_i is the domain of the i-th *feature* or *attribute* describing the example. A feature domain D_i can be *binary*, $D_i = \{true, false\}$, *categorical*, $D_i = \{v_1, ..., v_l\}$ for a fixed set of l distinct values, or *continuous*, $D_i = \mathbb{R}$. E will also be called the *training data* and the x_i *training instances*. It is assumed that the x_i are independently drawn from a fixed distribution $P(X)$; thus $P(x_1, ..., x_N) = \prod_{i=1}^{N} P(x_i)$. Such examples are said to be *i.i.d.*, independently drawn from identical distributions.

Unless noted otherwise, we will denote random variables with uppercase letters (such as X), and values of random variables with lowercase letters (such as x). Note that x can be vector-valued, as in the training examples $x_1, ..., x_N$. Furthermore, $P(x, y)$ is used as a short form for $P(X = x, Y = y)$.

Different problem settings can now be formulated. Sometimes, the goal is simply to estimate the unknown density function $P(X)$, which is called unsupervised learning. More frequently, we are interested in making predictions about a particular *class attribute* y given the remaining attributes; this is called supervised learning. In a supervised learning setting, E is a set of examples $\{x_1, ..., x_N\}$ as above, plus a set of labels $\{y_1, ..., y_N\}$ with $y_i \in Y$, that is, Y is the domain of the class attribute. Instances and labels are assumed to be independently drawn from a joint distribution $P(X, Y)$, such that $P(x_1, ..., x_N, y_1, ..., y_N) = \prod_{i=1}^{N} p(x_i, y_i)$. Labels can be binary, categorical, or continuous. We will mostly focus on *classification problems*, where labels are binary or categorical. Supervised learning problems with continuous labels are called *regression* problems.

The goal in prediction is to infer from the available data E a model $\lambda : \mathcal{X} \to Y$ that has minimum error on unseen instances $x \notin E$. More formally, we want to minimize the *generalization error*

$$Err(\lambda) = \mathbb{E}_{x,y}(\ell(\lambda(x), y)) = \int_{\mathcal{X}} \int_{\mathcal{Y}} \ell(\lambda(x), y) P(x, y) dx dy \tag{2.1}$$

where $\mathbb{E}_{x,y}$ denotes the expectation with respect to the distribution $P(X, Y)$, and ℓ is a loss function that measures the loss of predicting $\lambda(x)$ if the true label is y. Note that if the domain of x or y is finite, the corresponding integral is replaced with a finite sum. For classification problems, $\ell(y', y)$ is typically misclassification error given by

$$\ell(i, j) = \begin{cases} 0 & : \quad i = j \\ 1 & : \quad \text{otherwise} . \end{cases}$$

These considerations can be formalized in the following learning setting:

Problem 2.1.1 (Statistical Learning)**.**

Given

- *A set of training instances $E \subset \mathcal{X} \times Y$;*
- *A space of possible classifiers $\Lambda \subset \{\lambda \mid \lambda : \mathcal{X} \to Y\}$;*
- *A loss function $\ell : Y \times Y \to \mathbb{R}$*

Find

$$\lambda^* = \arg\min_{\lambda} Err(\lambda)$$

where $Err(\lambda)$ is defined by Equation (2.1).

Unfortunately this is not an operational definition as the true generalization error $Err(\lambda)$ of a model λ cannot be determined. Instead, λ is typically selected based on a combination of the training error

$$Err_E(\lambda) = \frac{1}{N}\sum_{i=1}^{N} \ell(\lambda(x_i), y)$$

and some bias or regularization term that specifies a preference relation over models in Λ (see Bishop, 2006, for a more detailed discussion).

Note that so far we have only assumed that the classifier λ returns a class label $\lambda(x) \in Y$. For several reasons, it can be beneficial if λ returns an estimate $P_\lambda(Y \mid X = x)$ of the conditional distribution $P(Y \mid X = x)$ over labels given an instance x. First, this is needed to make rational decisions in the context of varying misclassification costs, as it enables us to trade off the probability of error and the expected cost of a particular misclassification. Second, a probabilistic classification provides a confidence value for any prediction. This is useful in many domains, for instance, to refer ambiguous cases to a human expert for further inspection.

A natural approach to statistical modeling is estimating the joint distribution $P(X, Y)$ from E. Given such an estimate $P_\lambda(X, Y)$, a predictive distribution is obtained by

$$\begin{aligned} P_\lambda(y \mid x) &= \frac{P_\lambda(x, y)}{P_\lambda(x)} \\ &= \frac{P_\lambda(x, y)}{\sum_{y' \in Y} P_\lambda(x, y')}. \end{aligned}$$

This approach is called *generative modeling*, and the resulting model λ is called a *generative model*:

Definition 2.1.1 (Generative Model)**.** *A generative model λ provides an estimate $P_\lambda(X, Y)$ of the full joint probability distribution $P(X, Y)$ based on the available training data E.*

Generative models thus capture the full probabilistic structure of the domain, and in particular allow to sample new data from the learned model (that is, they allow to predict not only labels but also instances that are likely to be encountered). Generative models are the most widely used statistical models; examples include Bayesian networks, hidden Markov models, and Markov networks.

An alternative class of models that has received increasing attention recently are *discriminative* models. The idea behind discriminative modeling is to directly model the conditional distribution $P(Y \mid X)$, avoiding the (for prediction unnecessary) effort of estimating $P(X)$:

Definition 2.1.2 (Discriminative Model)**.** *A discriminative model λ provides an estimate $P_\lambda(Y \mid X)$ of the conditional probability distribution $P(Y \mid X)$ based on the available training data E.*

While the relative merits of generative and discriminative modeling are still being discussed (Ng and Jordan, 2001; Liang and Jordan, 2008), discriminative models have been shown to outperform generative models empirically in many domains (see, for instance, the discussion in Nallapati, 2004).

There are also important classes of models which can be statistically motivated, and will thus be called statistical models in this thesis, but do not explicitly manipulate probability distributions. The most important class we will be concerned with are many *kernel machines*. Kernels generalize the notion of inner product to arbitrary domains, in that the kernel function corresponds to an inner product in an induced reproducing kernel Hilbert space (RKHS). Inner product spaces are the basis of principled and powerful classification algorithms such as the support vector machine, which fits a linear max-margin decision boundary in the RKHS (Cortes and Vapnik, 1995). Kernel machines will be discussed in more detail in Chapter 4. Note that while standard kernel machines do not explicitly represent probability distributions, there are other kernel-based learning approaches that do combine kernels with explicit probabilistic frameworks, for example, Gaussian processes (Rasmussen and Williams, 2005).

2.1.1 Probabilistic Graphical Models

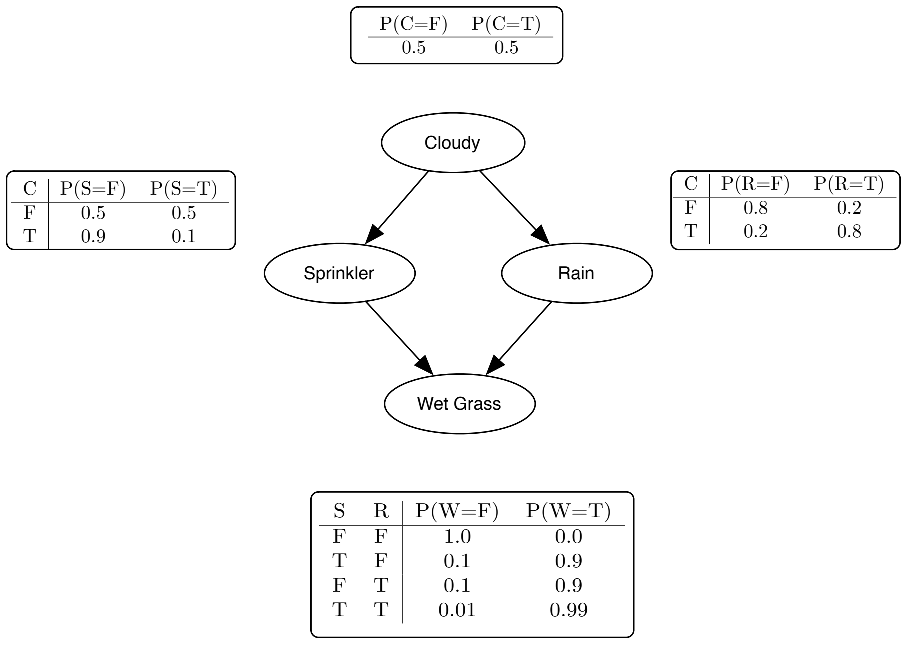
A unifying framework for describing generative probabilistic models are *graphical models*. They represent a joint probability distribution over a set of random variables, and use a graph structure to encode a probabilistic dependency structure (or, equivalently, a set of independence assumptions). Graphical models have become one of the most popular tools in statistical machine learning, as they combine a principled, compact, and intuitive representation language for probability distributions with powerful inference and learning algorithms. Two classes of graphical models can be distinguished: directed models (known as Bayesian networks) and undirected models (known as Markov networks). In this thesis, we will mostly focus on directed graphical models, that is, (variants of) Bayesian networks:

Definition 2.1.3 (Bayesian Network). *A Bayesian network consists of*

- *a directed acyclic graph structure G over a set $\mathbf{X} = \{X_1, .., X_M\}$ of nodes representing random variables;*
- *for every $X_i \in \mathbf{X}$, a specification of the probability distribution $P(X_i \mid pa(X_i))$, where $pa(X_i)$ denotes the set of parent nodes of X_i in G.*

Here, a node X_i is called a parent *of a node X_j if there is an arc from X_i to X_j in G. The Bayesian network encodes the independence assumption that every node is conditionally independent of all non-descendant nodes given the state of its parents. The joint distribution represented by the network thus factorizes to*

$$P(X_1, ..., X_M) = \prod_{i=1}^{M} P(X_i \mid pa(X_i)). \tag{2.2}$$



P(C=F)	P(C=T)
0.5	0.5

C	P(S=F)	P(S=T)
F	0.5	0.5
T	0.9	0.1

C	P(R=F)	P(R=T)
F	0.8	0.2
T	0.2	0.8

S	R	P(W=F)	P(W=T)
F	F	1.0	0.0
T	F	0.1	0.9
F	T	0.1	0.9
T	T	0.01	0.99

Figure 2.1: A Bayesian network for the *Sprinkler* domain (inspired by Pearl, 1988).

In the following, we will assume the X_i to be categorical, although extensions to numerical variables exist (Bishop, 2006). Markov networks implement similar ideas using undirected graphs:

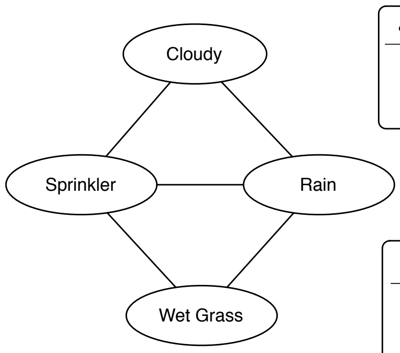
Definition 2.1.4 (Markov Network)**.** *A Markov network consists of*

- *an undirected graph structure G over a set $\mathbf{X} = \{X_1, .., X_M\}$ of nodes representing random variables;*
- *a set of potential functions Φ_k, one for each clique k in G. A clique k is a subset of nodes in G that is fully connected, and a potential function Φ_k is a mapping from the joint states of variables in k to non-negative real numbers.*

A Markov network encodes the independence assumption that a node is independent of any other node in the network given the state of all its direct neighbors in the graph (the Markov blanket*). The joint probability distribution is given by*

$$P(X_1, ..., X_M) = \frac{1}{Z} \prod_k \Phi_k \tag{2.3}$$

where Z is a normalizing constant called the partition function.



$\Phi_{C,S,R}$	0.2	0.09	0.2	0.01	0.05	0.36	0.05	0.4
C	F	T	F	T	F	T	F	T
S	F	F	T	T	F	F	T	T
R	F	F	F	F	T	T	T	T

$\Phi_{W,S,R}$	1.0	0.0	0.1	0.9	0.1	0.9	0.01	0.99
W	F	T	F	T	F	T	F	T
S	F	F	T	T	F	F	T	T
R	F	F	F	F	T	T	T	T

Figure 2.2: A Markov network for the *Sprinkler* domain (inspired by Pearl, 1988).

An important advantage of Bayesian and Markov networks is their ability to represent full joint distributions in a sparse way by exploiting independence assumptions according to Equation (2.2) and Equation (2.3). While a naïve specification of the joint distribution requires a number of parameters exponential in the number of variables, the number of network parameters is exponential only in the size of the largest set of parents of a node (Bayesian network) or the size of the largest clique (Markov network). Furthermore, the graphical representation is intuitive for human users, as arcs basically indicate influences between random variables. For both Bayesian and Markov networks, sophisticated algorithms exists for learning models from data and for performing inference (probability calculations), as described, for instance, by Jordan and Weiss (2002).

Figure 2.1 shows a Bayesian network for the toy *Sprinkler* domain, inspired by Pearl (1988). The fact that it is *Cloudy* influences the probability that there is *Rain* or the *Sprinkler* is turned on. Both *Rain* and *Sprinkler* influence the probability of observing *Wet Grass*. The same joint distribution over the four variables involved can be expressed using the Markov network given in Figure 2.2. Note that the additional (undirected) dependency between *Sprinkler* and *Rain* needs to be added, as the network would otherwise encode the assumption that *Rain* was independent of *Sprinkler* given information on both *Cloudy* and *Wet Grass*, an assumption not encoded in the Bayesian network. Note furthermore that the conditional probability distributions in the Bayesian network have been replaced by two clique potentials $\Phi_{C,S,R}$ and $\Phi_{W,S,R}$.

We will encountered Bayesian networks again—as a general probabilistic representation formalism—in techniques discussed later (in Section 2.1.2, Chapter 3, Chapter 8).

2.1.2 Markov Processes

A further important scenario in statistical learning is the modeling of *time series* or *sequential* data. Time series models describe the dynamic state of a system over time by capturing its (statistical) transition behavior. We will be specifically interested in *Markov* *processes*, which form the basis of much of our discussion in Part II and Part III of the thesis.

Assume there is a process $\mathcal{R}$ whose state over time is described by a series of random variables $Z_1, Z_2, Z_3, ...$, where Z_t denotes the state of $\mathcal{R}$ at time t. $\mathcal{R}$ is called *Markov* if $P(Z_{t+1} \mid Z_1, ..., Z_t) = P(Z_{t+1} \mid Z_t)$, and *stationary* if this transition distribution is time-invariant, that is, $P(Z_{t+1} \mid Z_t) = P(Z_{t+k+1} \mid Z_{t+k})$. A Markov process can thus be defined as follows:

Definition 2.1.5 (Markov Process). *A stationary Markov process $\mathcal{R}$ consists of*

- *a set $\mathcal{Z}$ of possible states the process can be in at any point in time;*
- *an initial-state distribution $P(Z_1)$, where the variable Z_1 with domain $\mathcal{Z}$ represents the initial state of $\mathcal{R}$;*
- *a transition distribution $P(Z_{t+1} \mid Z_t)$ which determines the probability to transition from state $Z_t \in \mathcal{Z}$ to $Z_{t+1} \in \mathcal{Z}$, where $P(Z_{t+1} \mid Z_t) = P(Z_t \mid Z_{t-1})$ for all t.*

Given $T \in \mathbb{N}$, the joint distribution over state variables $Z_1, ..., Z_T$ is given by

$$P(Z_1, ..., Z_T) = P(Z_1) \prod_{t=1}^{T} P(Z_{t+1} \mid Z_t). \tag{2.4}$$

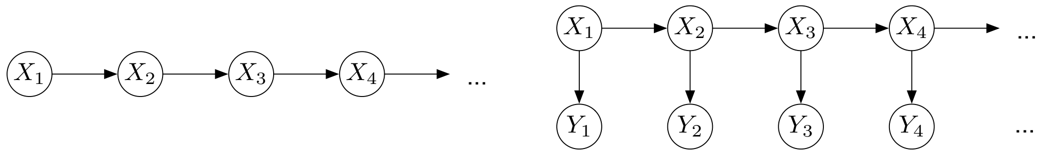
To learn time series models from data, we are given a set $E = \{s_1, ..., s_N\}$ of sequences that describe the observed behavior of the process. That is, $s_i = z_{i1}, ..., z_{iT_i}$ for $i = 1, ..., N$ where z_{it} is the observed state of $\mathcal{R}$ at time t in sequence s_i. We will also assume that sequences are drawn independently from the distribution $P(Z_1, ..., Z_T)$ defined by $\mathcal{R}$ (Equation (2.4))[1]. Note that, in contrast to general machine learning setting described above, the individual states z_{i1},...,z_{iT_i} are not i.i.d. as they are correlated over time, while the sequences $s_1, ..., s_N$ are.

In general, the random variable Z_t describing the state of $\mathcal{R}$ can be factored into several variables that describe different aspects of the current state, that is, Z_t can in general be a set of random variables. A particularly important case is that of $Z_t = \{X_t, Y_t\}$, where X_t is an unobservable internal state of $\mathcal{R}$, while Y_t is some observable indirect evidence of X_t. An important class of models in this setting are hidden Markov processes:

Definition 2.1.6 (Hidden Markov Process). *A stationary hidden Markov process $\mathcal{R}$ consists of*

- *a set $\mathcal{S}_X$ of possible hidden states the process can be in;*
- *a set $\mathcal{S}_Y$ of possible observations;*

[1]Here, we have not introduced an explicit distribution over the sequence length—this can be implemented, for example, using *sink states* from which no further transition is possible.



(a) Markov process. (b) Hidden Markov process.

Figure 2.3: Graphical model representation of a Markov and hidden Markov process. Note that the X_i nodes in the hidden Markov model are hidden, meaning that their state is never directly observable.

- *an initial-state distribution $P(X_1)$, where the variable X_1 with domain $\mathcal{S}_X$ represents the initial hidden state of $\mathcal{R}$;*
- *a transition distribution $P(X_{t+1} \mid X_t)$ which determines the probability to transition from hidden state $X_t \in \mathcal{S}_X$ to $X_{t+1} \in \mathcal{S}_X$, where $P(X_{t+1} \mid X_t) = P(X_t \mid X_{t-1})$ for all t.*
- *a distribution $P(Y_t \mid X_t)$ for the observational variable Y_t given the hidden state X_t, where $P(Y_t \mid X_t) = P(Y_{t-1} \mid X_{t-1})$ for all t.*

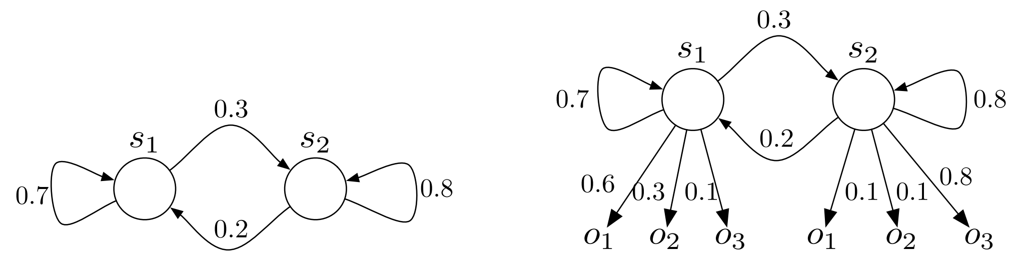
The joint distribution over hidden variables $X_1, ..., X_T$ and observed variables $Y_1, ..., Y_T$ is given by

$$P(X_1, ..., X_T, Y_1, ..., Y_T) = P(X_1)P(Y_1 \mid X_1) \prod_{t=1}^{T} P(X_{t+1} \mid X_t)P(Y_{t+1} \mid X_{t+1}). \tag{2.5}$$

Hidden Markov models are important tools in a wide range of application domains such as bioinformatics (Winters-Hilt, 2006), speech recognition (Rabiner, 1989), finance (Bahr, 2004), or activity recognition (Patterson *et al.*, 2005).

Markov and hidden Markov processes can be easily represented using the Bayesian network formalism introduced above. Figure 2.3 shows the graphical model representation of a Markov (left) and hidden Markov (right) process. Note that a Markov model essentially corresponds to a linear chain of variables describing the current process state, while a hidden Markov model is represented by two chains, one for the hidden state and one for the observations. Another popular representation for (hidden) Markov processes is the automaton notation. Figure 2.4 shows examples for a Markov and a hidden Markov process in this notation. In the automaton notation, nodes indicate possible states the process can be in, and arrows denote possible transitions and emission for each state.

In general, (hidden) Markov processes can be generalized to complex *dynamic Bayesian networks* (DBNs), which are graphical models for time series data that involve complex structures of interacting variables to describe the state of a process at any given



(a) Markov process. (b) Hidden Markov process.

Figure 2.4: Automaton representation of a Markov and a hidden Markov process. The set of states is $\mathcal{S}_X = \{s_1, s_2\}$, and the set of observations (for the hidden Markov process) is $\mathcal{S}_Y = \{o_1, o_2, o_3\}$. Transition and emission probabilities are indicated as arrow labels, for instance, $P(X_{t+1} = s_1 \mid X_t = s_2) = 0.2$, and $P(Y_t = o_1 \mid X_t = s_1) = 0.6$.

time. More formally, a dynamic Bayesian network defines a Bayesian network template $\mathcal{B}$ which is "unrolled" over time, such that there is one copy $\mathcal{B}(t)$ of the template for every point t in time. The template $\mathcal{B}$ defines probabilistic dependencies both between variables within $\mathcal{B}$ (as in a normal Bayesian network) and between time slices. The latter are arcs $X_t^i \rightarrow X_{t+1}^j$ which connect a variable $X_t^i \in \mathcal{B}(t)$ to a variable $X_{t+1}^j \in \mathcal{B}(t+1)$. Parameters are shared across time slices, thus DBNs are stationary in the sense outlined above. As an example for a complex DBN model, Figure 2.5 shows a network for recognition of interleaved activities. In this network, only nodes of the form Y_t (for $t = 1, ..., T$) are observable, while S_t^m and Z_t are hidden (see Chapter 8 for more details). Note that both Markov processes and hidden Markov processes are simple instances of DBNs.

From a statistical perspective, three interesting problems can be considered for time series models:

Problem 2.1.2 (Inference and Learning Tasks for Time Series Models)**.** *Let $\mathcal{R}$ denote a time series model over the state variables $Z = X \cup Y$, where X denotes the set of hidden state variables and Y denotes the set of observable state variables.*

- ***Inference****: Given a sequence of observations $y_1, ..., y_T$, where y_t denotes the joint state of variables in Y, find the probability*

$$P(y_1, ..., y_T \mid \mathcal{R}) = \sum_{x_1,...,x_T} P(x_1, ..., x_T, y_1, ..., y_T \mid \mathcal{R})$$

  of the given sequence of observations.

- ***Decoding****: Given a sequence of observations $y_1, ..., y_T$, where y_t denotes the joint state of variables in Y, find the most likely sequence*

$$\arg\max_{x_1,...,x_T} P(x_1, ..., x_T \mid y_1, ..., y_T, \mathcal{R})$$

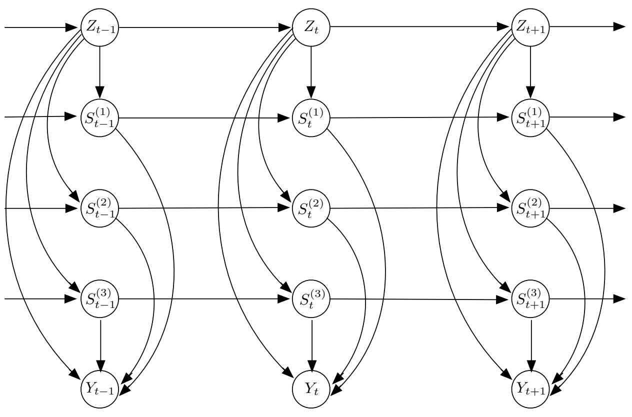


Figure 2.5: A dynamic Bayesian network model for recognition of interleaved activities. Only nodes of the form Y_t are observable (see Chapter 8 for a more detailed discussion).

of hidden states to have generated the observations.

- ***Parameter Learning**: Given a set $E = s_1, ..., s_N$ of sequences of observations with $s_i = y_{i1}, ..., y_{iT_i}$, find (new) parameters for $\mathcal{R}$ maximizing the likelihood*

$$L(E) = \prod_{i=1}^{N} P(y_{i1}, ..., y_{iT_i} \mid \mathcal{R})$$

of the observed data.

Time series models and dynamic Bayesian networks in particular will be discussed in more detail in Chapter 7 and Chapter 8.

Statistical machine learning techniques have been remarkably successful, and a wide variety of different approaches have been developed in the last two decades. The statistical perspective on machine learning offers thorough mathematical foundations, and a principled way to deal with uncertainty, noisy, or hidden data. Moreover, statistical machine learning techniques yield state-of-the-art accuracies in many application domains, often surpassing more traditional methods such as symbolic rule learners. On the other hand, drawing on statistics and probability theory means that these approaches are very much focused on working with data that is represented in the simple vector format introduced in the beginning of this section. It is thus typically not straightforward to upgrade statistical learning techniques to other, more flexible, data representations. This can be a severe limitation for several application domains. The next section discusses relational learning techniques, which can easily deal with complex and flexible data representations. Finally, in Section 2.3, we will discuss how statistical relational learning approaches try to combine the advantages of statistical and relational learning.

2.2 Relational Learning and Inductive Logic Programming

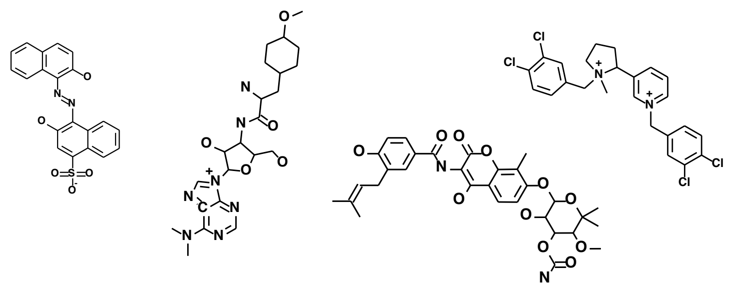
This section introduces relational data representations and inductive logic programming as a general framework for learning from relational data. More specifically, we will start by introducing and motivating relational representations as a way to cope with the complexity of real-world data collections. Afterwards, we briefly review some logic programming concepts, and give an overview of problem settings and algorithmic techniques employed in the field of inductive logic programming.

2.2.1 Relational Data Representations

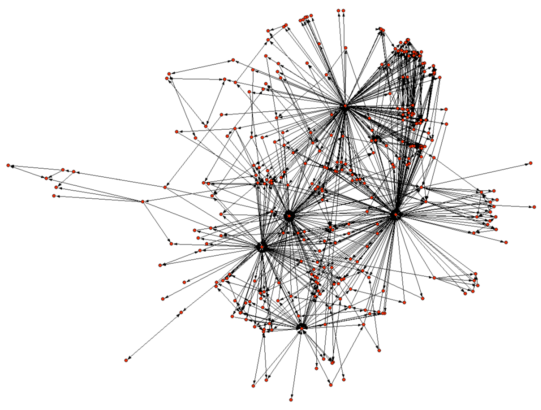
Classical machine learning—as discussed in the previous section—assumes that data instances x_i are points in an M-dimensional space defined by a set of M features, and that instances are independently drawn from identical distributions. However, even a cursory look at the real world shows that most data encountered in practice does not adhere to this simple attribute-value representation. Instead, data collections are often complex and structured, and individual data points cannot be considered independently because there are links or relations, and thus dependencies, between points.

The most ubiquitous example for complex data collections are *relational databases* and *graphs*. The former is the preferred format for large-scale data storage in cooperations, scientific knowledge bases, and public administration. Graph structures are found whenever data consists of individual entities connected by a link structure. Figure 2.6 shows examples of graph-based data from three real-world domains. The first domain is concerned with a *structure-activity relationship* (SAR) problem for chemical compounds. In SAR problems, the goal is to predict a certain property of interest (for instance, activation or blocking of a receptor in the human body, toxicity, or suppression of tumor growth) given the structure of a molecule (Srinivasan *et al.*, 1994). Such problems are of central importance in many areas of bio- and chemoinformatics. SAR problems will be further discussed in Chapter 3 and Chapter 4. The second example is concerned with the analysis of the hyperlink structure in the world wide web. A possible task would be to automatically classify web pages based on the words appearing on the page and the hyperlink neighborhood (see Chapter 4). This can be helpful for solving information retrieval problems (Slattery and Craven, 1998). The final example visualizes game graphs from a massively multiplayer online game, which relate game entities such as players, cities, and alliances (Thon *et al.*, 2008). This domain will be discussed in more detail in Chapter 6.

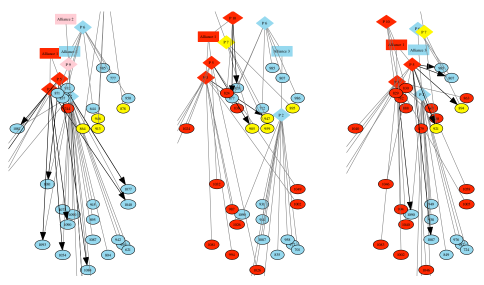
We now discuss how *relational data representations* can elegantly account for the complexity of real-world data. The main idea behind relational representations is to organize data in *relations*, that is, sets of tuples. There are two major formalisms: the entity-relationship (or *ER*) model from databases and first-order predicate logic. Throughout this thesis, we will use logic, as is offers a slightly more general framework. To this end, we will first briefly review key elements of first-order predicate logic.



(a) Structure-activity relationship prediction for molecules.



(b) Hyperlink graph on the world wide web.



(c) Game graphs in a massively multiplayer online game.

Figure 2.6: Examples for relational data from three real-world domains (hyperlink graph taken from Cothey *et al.*, 2006).

An *atom* is an expression of the form $p(t_1, ..., t_k)$ where p/k is a *predicate symbol* of arity n and the t_i are *terms*. Terms are either constants (typically denoted by lowercase letters or identifiers), variables (typically denoted by uppercase letters) or structured terms of the form $f(t_1, .., t_k)$, where f/k is a functor symbol of arity k and $t_1, ..., t_k$ are again terms. A *literal* is an atom or its negation. *Ground* expressions do not contain variables. Ground atoms will also be called *facts*. A first-order alphabet Σ is a set of predicate symbols, constant symbols, and functor symbols. The set of all ground atoms that can be constructed from Σ is called the *Herbrand base* of Σ, denoted $hb(\Sigma)$. A subset $I \subseteq hb(\Sigma)$ of the Herbrand base is called a *Herbrand interpretation*; I defines which of the possible atomic statements are true in a given situation. A *substitution* $\theta = \{V_1/t_1, ..., V_k/t_k\}$ is an assignment of terms t_i to variables V_i. Applying θ to a term or atom f simultaneously replaces all occurrences of the variables $V_1, ..., V_k$ by the respective terms $t_1, ..., t_k$. The resulting term or atom will be denoted by $f\theta$.

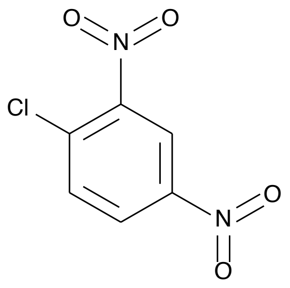
Example 2.2.1. *Consider the domain of natural numbers, represented as* 0*,* $s(0)$*,* $s(s(0))$*,* $s(s(s(0)))$*, etc. A first-order alphabet for this domain is given by the constant* 0*, the functor* $s/1$ *mapping a number to its successor, and predicates* $even/1$ *and* $odd/1$ *indicating even and odd numbers. Structured terms of the form* $s(...s(s(0))..)$ *are built from the constant* 0 *and (repeated) application of the functor* $s/1$*. The following Herbrand interpretation represents even and odd number of up to size four:*

$$I = \{even(0), even(s(s(0)), even(s(s(s(s(0))))), odd(s(0)), odd(s(s(s(0))))\}.$$

The substitution $\theta = \{X/s(s(0))\}$*, applied to the atom* $f = even(X)$*, yields the fact* $f\theta = even(s(s(0))$*.*

First-order logic can naturally represent complex structured data, such as graphs and networks: nodes are represented by constants and edges by ground atoms. Node or edge labels can also be incorporated using additional predicates or arguments in predicates.

Example 2.2.2. *Consider representing the graph structure of a molecule, as in the mutagenicity experiments presented by Srinivasan* et al. *(1994).*



```
molecule(d26).
mutagenic(d26).
lumo(d26, -2.072).
logp(d26, 2.17).

atm(d26,d26_1,c,22,-0.093).        bond(d26,d26_1,d26_2,7).
atm(d26,d26_2,c,22,-0.093).        bond(d26,d26_2,d26_3,7).
atm(d26,d26_3,c,22,-0.093).        bond(d26,d26_3,d26_4,7).
atm(d26,d26_4,c,22,-0.093).        bond(d26,d26_4,d26_5,7).
atm(d26,d26_5,c,22,-0.093).        bond(d26,d26_5,d26_6,7).
atm(d26,d26_6,c,22,-0.093).        bond(d26,d26_6,d26_1,7).
atm(d26,d26_10,cl,93,-0.163).      bond(d26,d26_10,d26_5,1).
atm(d26,d26_11,n,38,0.836).        bond(d26,d26_4,d26_11,1).
atm(d26,d26_12,n,38,0.836).        bond(d26,d26_2,d26_12,1).
atm(d26,d26_13,o,40,-0.363).       bond(d26,d26_13,d26_11,2).
atm(d26,d26_14,o,40,-0.363).       bond(d26,d26_11,d26_14,2).
atm(d26,d26_15,o,40,-0.363).       bond(d26,d26_15,d26_12,2).
atm(d26,d26_16,o,40,-0.363).       bond(d26,d26_12,d26_16,2).
```

The constant `d26` *identifies the molecule under consideration. Atoms in the molecule are represented with constants such as* `d26_1`*. A fact* `atom(d26,d26_1,c,22,-0.093)` *denotes that atom* `d26_1` *is a carbon atom of type* `22` *and charge* `-0.093`*. A fact* `bond(d26,d26_2,d26_3,7)` *represents a bond of type* `7` *(aromatic) between atoms* `d26_2` *and* `d26_3`.

However, more general structures such as hypergraphs can also easily be represented. Furthermore, there is a straightforward mapping from the ER model used in relational databases to first-order predicate logic (Das, 1992), meaning that any relational database can be converted to a logical representation. In the following we will refer to the large class of complex data collections that can be represented this way as *relational data*, and distinguish them from simple attribute-value datasets which we will call *propositional*.

As for propositional data, different machine learning problems such as supervised and unsupervised learning can be considered for relational data. Typical prediction problems include classifying whole graphs (such as molecules in structure-activity prediction), or individual nodes within a graph (such as web pages in a hyperlink structure graph). Moreover, new problems such as *link prediction* can be considered (Getoor, 2003): predicting new links in a given (partial) graph based on the available graph structure. As an example, consider a model for hyperlink graphs, which could be trained to propose additional appropriate hyperlinks from a given page to other related pages based on the known graph structure. As another example for link prediction, we will consider predicting player actions in a massively multiplayer online game in Chapter 6.

Note that from a logical perspective, these different prediction problems are rather similar: they all involve the prediction of some *target predicate* $p(X_1, ..., X_k)$; such a target predicate can represent node or graph labels as well as links or even more complex structures of interest.

2.2.2 Inductive Logic Programming: Problem Settings

Inductive logic programming (ILP) is a machine learning technique that uses logical representations for both the data and the model that is to be inferred from the data. More specifically, the goal of inductive logic programming in a classification setting is to infer a *hypothesis* H in the form of a logic program from a given set of positive (E^+) and negative (E^-) examples, which together make up the training data E. The building blocks of logic programs are *definite clauses*:

Definition 2.2.1 (Definite Clause)**.** *A definite clause q is an expression of the form*

$$h \leftarrow b_1, ..., b_n$$

where h and the b_i are atoms, and "comma" is read as conjunction, that is, $b_1 \wedge ... \wedge b_n$. The atom h is called the head of the clause (denoted $head(q)$) and $b_1, ..., b_n$ its body (denoted $body(q)$) [2].

[2]A clause *q* is called *range-restricted* if all variables appearing in the head also appear in the body, and we will limit our attention to range-restricted clauses.

The intuitive semantics of q is that whenever the body $b_1, ..., b_n$ holds, the head h will also be true. Note that the definition of a clause subsumes logical facts, as a clause with an empty body ($n = 0$) is a fact.

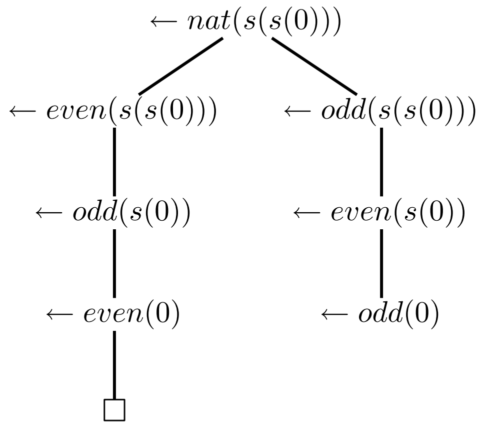
Definition 2.2.2 (Logic Program)**.** *A logic program $\mathcal{P}$ is a set of definite clauses, that is, $\mathcal{P} = \{q_1, ..., q_m\}$ with $q_i = h_i \leftarrow b_{i1}, ..., b_{in_i}$ for $i \in \{1, ..., m\}$.*

A Herbrand interpretation I is a model of a clause q if and only if for all substitutions θ such that $body(q)\theta \subseteq I$ holds, it also holds that $head(q)\theta \subseteq I$. I is a model of a logic program $\mathcal{P} = \{q_1, ..., q_m\}$ if it is a model of all clauses $q_i \in \mathcal{P}$. A logic program $\mathcal{P}$ entails another logic program $\mathcal{P}'$, denoted $\mathcal{P} \models \mathcal{P}'$, if and only if every model of $\mathcal{P}$ is also a model of $\mathcal{P}'$.

We will say that the program $\mathcal{P}$ logically entails a fact f, denoted $\mathcal{P} \models f$, if f holds in any model of $\mathcal{P}$. Algorithmically, this can be determined using *SLD resolution*, which is implemented in logic programming languages such as Prolog. The idea in SLD resolution is to negate the fact f to be proven to form a so-called *goal clause* $\{\neg f\}$. SLD resolution then tries to derive from $\mathcal{P} \cup \{\neg f\}$ the empty clause through a series of *resolution steps*, proving that $\mathcal{P} \cup \{\neg f\}$ is unsatisfiable and thus $\mathcal{P} \models f$. A resolution step basically consist of unifying a literal l in the current goal clause with the head of a clause $q \in \mathcal{P}$ using a substitution θ. The current goal clause is then replaced by a new goal clause in which the literal l is replaced by the negated literals in the body of q and the substitution θ is applied. SLD resolution is *sound* and *refutation complete* in the sense that starting from $\mathcal{P} \cup \{\neg f\}$ there is a series of resolution steps yielding the empty clause if and only if $\mathcal{P} \models f$. For a more detailed discussion of logic programming and SLD resolution, the reader is referred to Flach (1994). The least Herbrand model $LH(P)$ of a logic program $\mathcal{P}$ corresponds to the set of all facts entailed by $\mathcal{P}$: $LH(\mathcal{P}) = \{f \in hb(\Sigma) \mid \mathcal{P} \models f\}$ where Σ is the first-order alphabet for $\mathcal{P}$. Finally, note that logic programming systems such as Prolog support reasoning with *program clauses*, which generalize definite clauses by allowing atoms in the body to be negated. Thus, a program clause is an expression $h \leftarrow b_1, ..., b_n$ where the b_i are (positive or negative) literals. In Prolog and related systems, negation is often implemented as *negation as failure*, meaning that an atom is false if it cannot be proven to be true (Flach, 1994).

Example 2.2.3. *Reconsider the domain of natural numbers introduced above.*

$$
\begin{aligned}
&nat(X) \leftarrow even(X).\\
&nat(X) \leftarrow odd(X).\\[1em]
&even(s(X)) \leftarrow odd(X).\\
&odd(s(X)) \leftarrow even(X).\\[1em]
&even(0).
\end{aligned}
$$



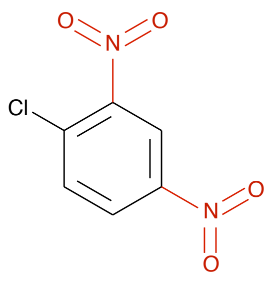
The five definite clauses shown on the left constitute a logic program for this domain. The program states that natural numbers are either even or odd, the successor of an even number is odd and vice-versa, and 0 *is an even number. The* SLD tree *shown on the right summarizes the resolution process for deriving the fact* $nat(s(s(0)))$*. An edge in the tree corresponds to a resolution step, and the adjacent nodes show the old and new goal clause. A successful series of resolution steps corresponds to a branch in the SLD tree that ends in the empty clause* $\square$*. There is exactly one such series for this goal clause (left branch).*

In inductive logic programming, the goal of learning is to find a logic program H that *covers* all positive and no negative examples, meaning that H should logically entail the correct classification for all examples. Depending on the exact specification of examples and the $covers$ relation, this can be realized in different settings, which are discussed in more detail below. An important feature of learning in logic-based representations is the possibility to employ *background knowledge* (Lavrač and Džeroski, 1992). Background knowledge is any form of prior knowledge relevant for the problem at hand, such as characteristic chemical groups in classification tasks involving molecules (Srinivasan *et al.*, 1994). More formally, the background knowledge B is again a logic program that is supplied to the learning system by the user, and which is assumed to be correct a priori. In the presence of background knowledge, the goal of learning is to find a hypothesis H that together with the background knowledge B covers all positive and no negative examples. Background knowledge provides a flexible way to influence the process of model induction, by defining new features and views on the (raw) logical data, which can be used as building blocks in the logical theory H.

Example 2.2.4. *Continuing Example 2.2.2, consider the following background knowledge:*

```
nitro(X,[Atom0,Atom1,Atom2,Atom3]) :-
    atm(X,Atom1,n,38,_),
    bondd(X,Atom0,Atom1,1),
    bondd(X,Atom1,Atom2,2),
    atm(X,Atom2,o,40,_),
    bondd(X,Atom1,Atom3,2),
    Atom3 @> Atom2,
    atm(X,Atom3,o,40,_).

bondd(X,Atom1,Atom2,Type) :-
    bond(X,Atom1,Atom2,Type).
bondd(X,Atom1,Atom2,Type) :-
    bond(X,Atom2,Atom1,Type).
```



The background predicate `bondd(X,Atom1,Atom2,Type)` *represents an (undirected) chemical bond between atoms* `Atom1` *and* `Atom2` *in terms of the (directed) logical predicate* `bond(X,Atom1,Atom2,Type)`*. The logical predicate* `nitro(X,[Atom0,Atom1,Atom2,Atom3])` *represents a nitro group, highlighted in the molecule structure on the right. Note that clauses are given in Prolog notation, where* $h \leftarrow b_1, ..., b_n$ *is replaced by* `h :- b1,...,bn`.

The two most important learning settings in ILP are *learning from entailment* and *learning from interpretations*, which will now be discussed in more detail.

Learning from entailment is the most popular setting in inductive logic programming, and it is implemented in a number of well-known systems such as FOIL (Quinlan, 1990), GOLEM (Muggleton and Feng, 1990), and PROGOL (Muggleton, 1995). When learning from entailment, examples are ground facts[3]. The background knowledge B is a set of definite clauses, and a hypothesis H covers an example e wrt. the background knowledge B if and only if $B \cup H \models e$.

Problem 2.2.1 (Learning from Entailment)**.**

Given

- *a background theory B, in the form of a set of definite clauses $h \leftarrow b_1, \cdots, b_n$;*
- *a set of examples E, in the form of ground facts, classified into positive and negative examples;*
- *a language of clauses $\mathcal{L}$, which specifies the clauses that are allowed in hypotheses;*
- *the $covers(e, H, B)$ function defined by $covers(e, H, B)$ if and only if $B \cup H \models e$;*
- *a $score(E, H, B)$ function, which specifies the quality of the hypothesis H w.r.t. the data E and the background theory;*

[3]or, more generally, definite clauses—but we will restrict ourselves to the case of ground facts.

Find

$$\arg\max_{H \subset \mathcal{L}} score(E, H, B) \ .$$

Many well-known learning algorithms in ILP implement this learning setting. They differ in the choice of the scoring function $score(E, H, B)$, the language bias $\mathcal{L}$ and the way the search for the highest-scoring hypothesis H is carried out (greedy search, branch-and-bound search, etc).

When learning from entailment, little information is provided at the level of examples as they are represented by individual facts. In the setting of learning from interpretations, examples provide more information as they are complete interpretations:

Problem 2.2.2 (Learning from Interpretations)**.**

Given

- *a background theory B, in the form of a set of definite clauses $h \leftarrow b_1, \cdots, b_n$;*
- *a set of examples E, in the form of Herbrand interpretations, classified into positive and negative examples;*
- *a language of clauses $\mathcal{L}$, which specifies the clauses that are allowed in hypotheses;*
- *the $covers(e, H, B)$ function defined by $covers(e, H, B)$ if and only if e is a model of $B \cup H$, that is, $e \cup B \cup H \not\models \perp$;*
- *a $score(E, H, B)$ function, which specifies the quality of the hypothesis H w.r.t. the data E and the background theory;*

Find

$$\arg\max_{H \subset \mathcal{L}} score(E, H, B) \ .$$

In addition to the different amount of information contained in the examples, a crucial difference between the two settings is the notion of *generality* implied by the covers function.

Definition 2.2.3 (Generality Relation)**.** *A hypothesis G is called* more general than *another hypothesis S, denoted $G \preceq S$, if all examples covered by S are also covered by G.*

From the definition of coverage in Problem 2.2.1 and Problem 2.2.2 it follows that when learning from entailment, G is more general than S if and only if $G \models S$, but when learning from interpretations, G is more general than S if and only if $S \models G$.

2.2.3 Inductive Logic Programming: Techniques

Many different learning techniques have been developed in the field of inductive logic programming. This section briefly discusses some basic concepts to outline the general principles of ILP algorithms. Moreover, the FOIL algorithm will be presented as a very simple instance of an ILP rule learning system, which forms the basis for much of the work presented in later chapters. In general, we are concerned with *simple* relational learning techniques to be used as a starting point for efficient statistical relational learning approaches. A detailed discussion of more advanced topics in ILP is thus beyond the scope of this thesis; for more information, the reader is referred to De Raedt (2008).

The goal of learning is to find an optimal hypothesis according to some scoring function, which involves searching through a very large space of possible hypotheses $\mathcal{H} = 2^{\mathcal{L}}$ (cf. Problem 2.2.1). A central idea in most ILP systems is to structure the search space according to generality. The $\preceq$ relation induces a lattice on the space $\mathcal{H}$, and thus provides a way to systematically search in $\mathcal{H}$. This lattice can be searched either *top-down* or *bottom up*. In the former case, the search starts with the most general hypothesis and is then specialized; in the latter case it starts from the most specific hypotheses and is then generalized. There are also some hybrid approaches, for instance, for theory revision systems (De Raedt and Bruynooghe, 1994). In the following we will mostly discuss top-down approaches as they are often simpler than bottom-up search strategies.

A key concept for top-down search are *refinement operators*:

Definition 2.2.4 (Refinement Operator). *A refinement operator ρ under θ-subsumption is a function $\rho : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ such that*

$$\forall q' \in \rho(q) : q \text{ } \theta\text{-subsumes } q'$$

A clause $h \rightarrow b_1, ..., b_n$ θ-subsumes a clause $h' \rightarrow b'_1, ..., b'_l$ if and only if there is a substitution θ such that $\{h, \neg b_1, ..., \neg b_n\}\theta \subseteq \{h', \neg b'_1, ..., \neg b'_l\}\theta$.

A refinement operator thus returns a set of specializations of a given clause $h \leftarrow b_1, \cdots, b_n$. This is typically realized by either adding a new literal b_{n+1} to the clause yielding $h \leftarrow b_1, \cdots, b_n, b_{n+1}$ or by applying a substitution θ yielding $h\theta \leftarrow b_1\theta, \cdots, b_n\theta$. When learning an individual clause, simple top-down search techniques start with the most general clause and greedily search through the generality lattice using a refinement operator.

We will now present one of the simplest ILP learners, namely the *First Order Inductive Learning*, or *FOIL*, algorithm (Quinlan, 1990). This serves to outline the general principles of ILP algorithms, and will also form the basis for work presented later in Chapter 3 and Chapter 4. FOIL, like many inductive logic programming systems, follows a *greedy* and *incremental* approach to induce a hypothesis. Such an algorithm is described in Algorithm 1. It repeatedly searches for clauses that score well with respect to the data set and the current hypothesis, and adds them to the hypothesis. In the *update* function, the set of training examples can be updated after adding a clause to the current hypothesis. In

Algorithm 1 Generic FOIL algorithm.

```
Initialize $H := \emptyset$
**repeat**
    Initialize $c := p(X_1, \cdots, X_k) \leftarrow$
    **repeat**
        **for** all $c' \in \rho(c)$ **do**
            compute $score(E, H \cup \{c'\}, B)$
        **end for**
        let $c$ be the $c' \in \rho(c)$ with the best score
    **until** stopping criterion
    add $c$ to $H$
    $E := update(E, H)$
**until** stopping criterion
output $H$
```

the inner loop, the algorithm greedily searches for a clause that scores well. To this end, it employs a general-to-specific hill-climbing search strategy. The search is initialized with the most general clause $p(X_1, \cdots, X_k) \leftarrow$ where p/k is the predicate being learned and the X_i are different variables. To generate specializations of the current clause q, a refinement operator ρ under θ-subsumption is employed (Definition 2.2.4). From the set of refinements $\rho(q)$, the highest-scoring clause is selected and further refined, until a stopping criterion is met. The algorithm employs a stopping criterion to determine when to stop adding clauses to the current hypothesis.

This algorithm has been successfully applied to a wide variety of problems in inductive logic programming. In the original FOIL algorithm, the technique is further simplified to a *separate-and-conquer* approach: examples that are covered by a learned clause are removed from the training data, and the score of a clause can be computed without respect to the current hypothesis; that is, $score(E, H \cup \{c'\}, B)$ simplifies to $score(E, c', B)$. Many different scoring functions and stopping criteria have been employed. The original FOIL algorithm uses information gain based on the number of positive and negative tuples bound to clauses as the scoring function (Quinlan, 1990), and a stopping criterion based on the minimum description length principle (Rissanen, 1978). MFOIL, a variant of FOIL that was particularly designed to cope with noise in the training data, uses the m-estimate for scoring clauses and a significance-based stopping criterion (Lavrac and Dzeroski, 1994).

It should be stressed that the search problem in ILP is difficult, and the huge size of the search space requires heuristic search approaches for most real-world domains. As in other optimization problems, heuristic search strategies in ILP can suffer from local optima and premature convergence. The basic concepts present in FOIL, such as a search based on a refinement operator under θ-subsumption, also appear in many other ILP algorithms, and are therefore of general importance in ILP. However, the greedy search strategy employed by FOIL is only one, simple option, and many more advanced approaches have been explored in the literature. Other important classes of techniques include: *first-order decision trees*, that upgrade propositional tree-based learners to relational representations (Blockeel and Raedt, 1998); bottom-up approaches, that focus the search for clauses based on the available examples (Muggleton and Feng, 1990); and approaches based on branch-and-bound search (Muggleton, 1995). Local optima problems can also be alleviated by *lookahead* search techniques, which search several levels ahead in the refinement lattice for a candidate clause with a higher score. For example, in a level-one lookahead search all secondary refinements $\rho(\rho(q))$ of a clause q are considered as candidate clauses during search.

A further key component of practical ILP systems is the so-called *language bias*: a set of definitions that define which subset of all clauses in the first-order alphabet are syntactically valid. The language bias effectively constrains the search space by reducing the number of refinements that have to be considered. A popular way of implementing a language bias is to use a combination of *type* and *rmode* declarations. Type declarations associate a type to every argument of a predicate and functor in the first-order alphabet Σ. Variables appearing as arguments in predicates and functors take on the respective argument type. Valid clauses are type-conform, meaning that a variable can assume only one type. rmode declarations further constrain the valid clauses that can be formed. When adding a new literal of the form $l(X_1, ..., X_k)$ to a clause in the refinement operator, the rmode declaration defines how the variables $X_1, ..., X_k$ are instantiated: they can be new variables that do not occur in the clause so far, unified with an existing variable (of the right type), or one of a set of constants defined by the rmode declaration.

Example 2.2.5. *Reconsider the mutagenesis domain outlined in Example 2.2.2 and Example 2.2.4. A possible language bias for this domain is given below.*

```
type(atm(molecule,atom,element,atomtype,charge)).
type(bond(molecule,atom,atom,bondtype)).
type(bbond(molecule,atom,atom,bondtype)).
type(nitro(molecule,nitro_group)).

rmode(atm(+,-,c,22,-0.093)).        rmode(bbond,+,+,+,1).
...                                 rmode(bbond,+,+,+,2).
rmode(atm(+,-,o,40,0.836)).         rmode(bbond,+,+,+,7).
rmode(atm(+,-,cl,93,-0.163)).       rmode(nitro(+,-)).
```

The "+" symbol indicates that a new variable is introduced at this position when adding the respective literal. The "-" symbol indicates that an existing variable—that is, a variable already appearing in the partial clause to be refined—will be used.

The choice of language bias can significantly affect the effectiveness of the overall hypothesis search. Furthermore, there can be intricate interactions between the search technique, background knowledge, and language bias definition; thus careful design of these three elements is crucial to obtain good search results.

2.2.4 Inductive Logic Programming: Discussion

Inductive logic programming approaches have been remarkably successful, for example, in application problems in bio-and chemoinformatics (see Bratko and Muggleton, 1995, for an overview). These successes can, to a large extent, be explained by the use of an expressive general purpose representation formalism that can elegantly account for complex data, the easy incorporation of prior domain knowledge in the learning process, and the fact that the induced rules are easy to interpret by domain experts.

At the same time, the classical ILP settings and techniques suffer from some significant limitations. On the one hand, the use of logic as a unifying representation language for examples, background knowledge and hypotheses is elegant and flexible. On the other hand, logic as a representation and reasoning formalism is tailored to domains that are deterministic and discrete. In contrast, many domains in which machine learning is applied are strongly stochastic, and observations often noisy, hidden, or incomplete. In propositional machine learning, approaches based on sophisticated probabilistic models and statistical characterizations of data have been receiving increasing attention (Bishop, 2006). It has been recognized that the inability to (directly) cope with uncertain or probabilistic information is a major drawback of standard logic, and thus standard ILP approaches (De Raedt and Kersting, 2003). ILP techniques—by definition—search for a deterministic, ”crisp” logical theory that characterizes the data under consideration. For many domains this will lead to suboptimal results simply because there exist no hard-and-fast rules to characterize the data or make predictions.

2.3 Statistical Relational Learning: A Brief Overview

It has been realized that to model large-scale real-world domains, systems need to be able to handle both inherent uncertainty and relational structure. By restricting their attention to simple attribute-value representations, propositional machine learning approaches ignore much of the relational complexity of the real world. As an example, consider the task of building a filter for incoming e-mail. Traditional approaches are able to classify messages as spam or non-spam or sort them into a small set of pre-defined categories. However, we might want to detect that a message is not only non-spam but a request from our supervisor to meet with a number of colleagues in his office to discuss a particular project, and that this requires immediate attention. Producing and manipulating such structured representations of objects and relationships is one of the central challenges for machine learning systems (Getoor and Taskar, 2007). Statistical machine learning easily handles uncertainty, while inductive logic programming and related techniques address learning in relational domains. Simply speaking, statistical relational learning (or SRL) is the science of combining these two streams of research into techniques that perform statistical learning and inference in relational domains. Statistical Relational Learning is a relatively young but burgeoning field, and it has received much attention in recent years (Getoor and Jensen, 2000, 2003; Dietterich *et al.*, 2004; Fern *et al.*, 2006; De Raedt and Kersting, 2003; Domingos and Richardson, 2004; De Raedt *et al.*, 2008). Moreover, many promising application domains have been identified that require SRL techniques (De Raedt *et al.*, 2008).

This thesis is concerned with the integration of statistical and relational learning techniques as described above. In contrast to most approaches described in the literature, we will specifically focus on relatively simple combinations of statistical and relational techniques. By deliberately restricting expressivity, the resulting techniques occupy an intermediate position in the trade-off between propositional techniques and fully fledged statistical relation learning systems. As shown below, such a restriction has advantages in terms of computational complexity and in particular allows for efficient and principled structure learning. Before discussing these contributions in more detail in subsequent chapters, we briefly review existing, related approaches in the field.

The definition of SRL as the integration of statistical and relational learning is rather broad, and the problem can be approached from different directions. Accordingly, there are a number of different perspectives on the field. Two general perspectives are (1) to start from statistical techniques (such as graphical models) and extend these formalisms to relational data representations, and (2) to start from relational learning techniques and extend them with probabilistic elements, which includes probabilistic extensions of logic programming languages (Sato and Kameya, 1997; Muggleton, 1996). Other directions include so-called *linked data* settings (Macskassy and Provost, 2007) where individual data items are propositional but links make them non-i.i.d., kernels methods for structured data (Gärtner, 2003), and application-specific techniques such as recommender systems (Adomavicius and Tuzhilin, 2005). It is therefore not easy to categorize all existing work in SRL. However, we will briefly review three main approaches that can be distinguished, and which, although non-exhaustive, encompass many of the frameworks proposed in statistical relational learning. These are *propositionalization*, *knowledge-based model construction*, and *proof-based* methods.

2.3.1 Propositionalization

Propositionalization addresses relational learning problems by mapping the original relational data to a propositional representation, on which standard statistical machine learning techniques can be applied (Kramer *et al.*, 2001). More specifically, a set of *relational features* $\mathcal{F} = \{f_1, ..., f_m\}$ is employed to map a relational example e to an m-dimensional (binary or numeric) vector

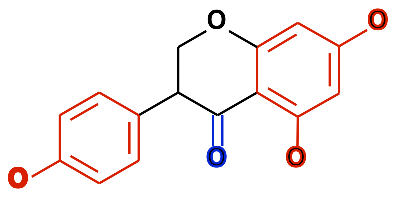
$$\varphi_{\mathcal{F},B}(e) = \begin{pmatrix} f_1(e) \\ \dots \\ f_m(e) \end{pmatrix}. \tag{2.6}$$

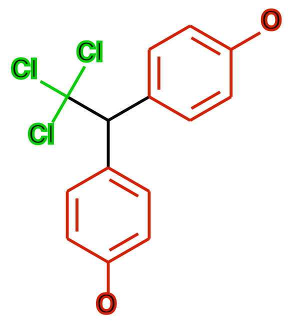
This vector summarizes the original relational structure and possibly links to other instances. Throughout the thesis, we will assume that the features f_i are definite clauses of the form $p(X_1, ..., X_k) \leftarrow b_1, ..., b_n$ where $p(X_1, ..., X_k)$ is the target predicate. An example is mapped onto the vector $\varphi_{\mathcal{F},B}(e) \in \{0,1\}^m$ with $\varphi_{\mathcal{F},B}(e)_i = 1$ if $B \cup \{f_i\} \models e$. That is, the vector $\varphi_{\mathcal{F},B}$ indicates which of the features covers the example e. To summarize, propositionalization defines a feature map from the space of relational examples to the space of binary vectors $\varphi_{\mathcal{F},B}(e) \in \{0,1\}^m$. Note that the dimension m of this feature map is the number of clauses in the logical theory, thus, this dimension will eventually depend on the model that is learned from data.

Example 2.3.1. *Reconsider the mutagenicity domain from Example 2.2.2, and the following set $\mathcal{F} = \{f_1, f_2, f_3\}$ of relational features:*

$$mutagenic(M) \leftarrow aromatic\_ring(M, [A0, A1, A2, A3, A4, A5]),$$
$$bond(X, A5, A6, 1), atm(M, A6, o, \_, \_)$$
$$mutagenic(M) \leftarrow atm(M, A, o, \_, \_), bond(M, A,, 2)$$
$$mutagenic(M) \leftarrow atm(M, A1, cl, \_, \_), bond(M, A1, A0, 1),$$
$$atm(M, A2, cl, \_, \_), bond(M, A2, A0, 1), atm(M, A3, cl, \_, \_), bond(M, A3, A0, 1)$$

where $aromatic\_ring(M, [A0, A1, A2, A3, A4, A5])$ is a background predicate indicating that M contains an aromatic ring structure involving molecules $A0, ..., A5$. These clauses constitute features. Applying these features to a set of molecules yields the following propositionalized dataset:





$$\varphi_{\mathcal{F},B}(e_1) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad \varphi_{\mathcal{F},B}(e_2) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Relational (sub)structures matched by the three features are highlighted in red, blue and green.

Different propositionalization techniques can be distinguished. In the simplest setting, there is a fixed user-supplied set of features $\mathcal{F}_{user}$ that is used to transform the relational to a propositional problem. This is effectively a data preprocessing step that avoids tackling the full relational learning problem. Alternatively, an existing relational learner (e.g.

an ILP system or a pattern miner) can be used to infer the set of features $\mathcal{F}$, that is then used to map examples to a propositional space. This can be seen as a post-processing step, where the result of the relational learner is post-processed by a statistical classifier. Finally, another class of approaches considers (explicitly or implicitly) all features within a large feature space $\mathcal{F}_{lang}$ defined by a language bias. For instance, many kernel functions for graph data are based on the idea of (approximately) counting the number of co-occurring substructures between two graphs (Gärtner, 2003). This can be seen as implicitly constructing a very large feature space on which a propositional kernel is applied. Propositionalization systems such as 1BC (Lachiche and Flach, 2003) and LINUS (Lavrač *et al.*, 1991) also essentially follow this strategy. The approach typically results in a very large feature set, which has the advantage of representing the original relational instances relatively accurately. On the downside, the mapping is also opaque, and the model will not yield domain insights because it is too hard to interpret.

Propositionalization approaches have been successfully applied to a variety of problems. However, from a theoretical perspective the decomposition of the overall learning problem into a feature construction and a separate statistical modeling step is not appealing. The main drawback is that the construction of the features is decoupled from the actual statistical modeling and uses a different selection criterion. Thus, no jointly optimal combination of the feature set and statistical classifier will be found. Note that as statistical classifiers, propositionalization approaches are discriminative (Definition 2.1.2) because they do not define a distribution over the original relational instance space. More specifically, let X denote a random variable for relational instances, and Y a random variable for the corresponding class label. Propositionalization (together with a statistical classifier built on the propositionalized data) yields a model for the distribution $P(Y \mid X)$. Thus, propositionalization avoids the problem of specifying a (generative) distribution $P(X)$ over the relational data. The latter is a much harder problem because of the size and complexity of the space of relational instances. We will revisit propositionalization approaches when discussing simple SRL models for classification in Chapter 3 and Chapter 4.

2.3.2 Knowledge-based Model Construction

Knowledge-based model construction, or KBMC, is an approach to upgrade probabilistic graphical models to relational domains by dynamically constructing propositional models to answer probabilistic queries (Wellmann, 1992). The relational structure of the domain is encoded in a knowledge base which uses some relational representation format (typically, the ER model or logic). Probabilistic influences between objects and their properties are specified at an abstract relational level in probabilistic "templates", which can be grounded to define probability distributions at the level of actual objects in the domain. Given a probabilistic query in the relational domain, the knowledge base can then be used to construct a query-specific graphical model by "unrolling" the probabilistic templates into a so-called *ground network*.

For instance, *Markov logic networks* define a generative model over logical interpretations by combining Markov networks (Definition 2.1.4) with first order logic (Richardson and Domingos, 2006). A Markov logic network consists of a set of first-order logical formulae $H = C_1, ..., C_m$ with associated weights $w_{C_1}, ..., w_{C_m}$. Given a finite domain Σ (basically, a set of predicates and constants), the formulae $C_1, ..., C_m$ serve as a template for building a propositional Markov network G. G has a binary node for every ground atom in the domain, and an edge between two nodes if the corresponding (ground) atoms appear in a possible grounding of a formula C_i. Clique potentials in G are defined by the corresponding weights w_{C_i}. The probability of an interpretation over Σ is now defined by

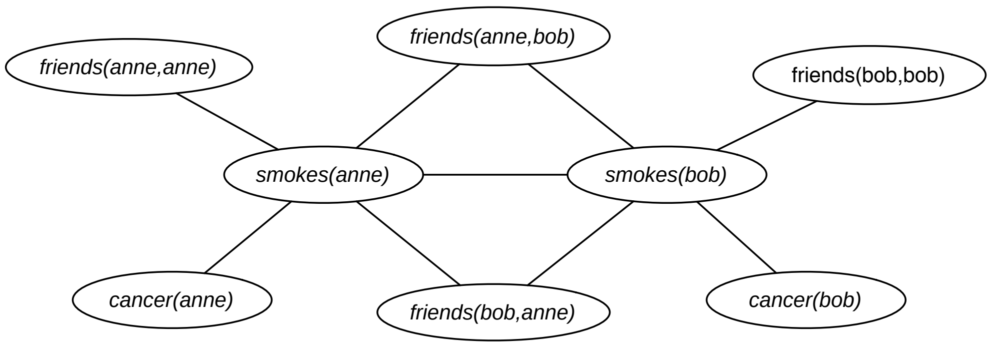
$$P(I \mid H) = \frac{1}{Z(I)} \prod_{C \in H} e^{n_C(I) w_C} = \frac{1}{Z(I)} \prod_{C \in H} \Phi_C(I_{\{C\}})^{n_C(I)}$$

where $n_C(I)$ is the number of true groundings of formula C in I, and $I_{\{C\}}$ is the state of the clique formed by the formula C in I.

Example 2.3.2. *Consider a domain consisting of the predicates $friends(X, Y)$, $smokes(X)$, and $cancer(X)$; and the constants $anna$ and bob (taken from Richardson and Domingos, 2006). Let*

$$\forall X : smokes(X) \Rightarrow cancer(X)$$
$$\forall X, Y : friends(X, Y) \Rightarrow (smokes(X) \Leftrightarrow smokes(Y))$$

denote formulae in a Markov logic network. The resulting ground network is as follows (Richardson and Domingos, 2006):



It defines a distribution over interpretations I over Σ, as there is a one-to-one mapping between joint states of the binary nodes in the network and interpretations over Σ: a node has state $true$ if $p \in I$ for the corresponding ground atom p, and false otherwise.

Many SRL approaches that extend probabilistic graphical models to relational domains use KBMC principles. Examples include Markov Logic Networks (MLNs), Bayesian Logic Programs (BLPs) (Kersting and De Raedt, 2001), Probabilistic Relational

Models (PRMs) (Koller, 1999) and Relational Bayesian Networks (RBNs) (Jaeger, 1997). They can be distinguished according to the type of probabilistic model employed (e.g. undirected as in MLNs vs. directed as in BLPs, PRMs, and RBNs), and their underlying relational framework (e.g. the ER model as in PRMs, or logic-based formalisms as in BLPs, RBNs, and MLNs). Note that KBMC-based approaches constitute generative statistical models (Definition 2.1.1), as they specify a full joint distribution over relational data. Nevertheless, they can be used for prediction; in fact, special-purpose algorithms for discriminatively training e.g. Markov logic networks have been investigated (Huynh and Mooney, 2008).

KBMC-based formalisms typically feature powerful probabilistic inference techniques inherited from the underlying probabilistic graphical model framework. They are also principled, as they directly define distributions over relational domains. However, the powerful probabilistic inference comes at a computational cost. In particular, structure learning (that is, learning the "templates" that define the probabilistic influences in the domain under consideration) is hard. KBMC approaches have thus been mostly applied in scenarios where the domain structure is known, and humans can easily define structural dependencies manually. Parameters can then be learned by lifted versions of standard algorithms such as expectation-maximization or gradient-based methods. We will revisit KBMC strategies when discussing Markov models for relational sequences and sequences of interpretations in Chapter 5 and Chapter 6.

2.3.3 Proof-based Approaches

Finally, *proof-based* approaches to SRL are rooted in logic programming principles. Their general idea is to add some probabilistic element to a logic programming formalism, such as Prolog, and thereby define a probabilistic logical resolution technique. From this, different distributions can be obtained: for example, distributions over derivation and refutations (as in Stochastic Logic Programs, Muggleton, 1996), Herbrand interpretation (as in PRISM, Sato and Kameya, 1997), or possible worlds (as in ICL, Poole, 1997). Approaches such as stochastic logic programs can also be seen as an extension of stochastic context-free grammars to first-order logic. The techniques we derive later on are mostly influenced by propositionalization and KBMC-based approached to SRL; thus, a detailed discussion of proof-based approaches is beyond the scope of the thesis. For more details, the reader is referred to Muggleton (2000).

2.3.4 Statistical Relational Learning: Discussion

Recently, the field of statistical relational learning has matured to a degree where a thorough theoretical understanding, robust implementations of SRL systems, and significant real-world applications are available (De Raedt *et al.*, 2008). Furthermore, a synthesis of the different perspectives on and approaches to SRL has started to develop.

By combining statistical and relational learning techniques, SRL has the appealing potential to combine the advantages of the two underlying frameworks, namely the robustness and accuracy of statistical learning with the representational power of relational learning. It is widely recognized that this will be needed in order to apply machine learning techniques to complex and challenging application domains in which data is structured, linked, or otherwise interrelated.

On the other hand, SRL also poses new challenges. By combining statistical and relational learning, we also risk combining the complexity of the two underlying frameworks, as both the relational and the statistical component of the model now need to be learned from data. For instance, in knowledge-based model construction approaches (see Section 2.3.2), first-order models need to be grounded into a ground network to perform inference or parameter learning, and such ground networks can grow quickly in the size of the model and domain under consideration, making inference and parameter learning expensive. For structure learning the problem is even more severe, as typically parameter learning needs to be performed in a large number of candidate first-order models, meaning that a large number of such ground networks need to be built and learned from data. From a practical perspective, this high computational complexity of most existing SRL systems is probably one of the main reason still preventing SRL from being widely applied in large-scale real-world domains, where the benefits of relational models could be most significant. Reducing the computational complexity of SRL approaches is therefore one of the most important research directions in the field.

One way of reducing complexity is to explore sophisticated (approximate) algorithms for inference and learning. There are several promising directions in this area, including lifted inference (De Salvo Braz *et al.*, 2007) or relational upgrades of variational methods (Jordan, 1999). While being worthwhile research directions, these approaches are usually complicated to develop and implement, not always applicable, and their effectiveness depends strongly on the considered domain.

This thesis will take a different approach to reducing computational complexity of SRL techniques, namely by considering the expressivity-efficiency trade-off inherent in all machine learning approaches. Propositional learning techniques, which are relatively restricted in terms of representational power, but for which efficient learning algorithms are available, can be seen as occupying one extreme in this trade-off. Existing SRL techniques have mostly tried to combine the full power of statistical and relational learning, making them very expressive modeling frameworks, but limiting computational efficiency. They are thus at the other extreme of the expressivity-efficiency trade-off. This thesis will explore approaches that occupy an intermediate position in the outlined trade-off, by proposing relatively *simple* combinations of statistical and relational learning which are less expressive than more general SRL systems but typically much more efficient. This will be achieved by deliberately restricting modeling power, making assumptions that simplify learning and inference (for example, that data is fully observable), or focusing on particular application domains. We show in subsequent chapters how this approach can yield principled yet efficient algorithmic techniques for learning and inference, and good empirical results in several real-world domains.

2.4 A Framework for Integrating ILP and Statistical Learning

In inductive logic programming, the learning setting involves examples, hypotheses, and predictions which are discrete and deterministic, as outlined in Section 2.2.2. As a first step towards integrating logical learning with statistical modeling, this section presents an adapted learning setting that accounts for a combined statistical-logical model. The logical part of the model captures the relational domain structure, while the statistical part of the model accounts for probabilistic aspects of the data. Specifically, we will extend the classical setting of learning from entailment (Problem 2.2.1) to the setting of learning from statistical entailment. This extension is closely related to the exposition by De Raedt and Kersting (2004), where a probabilistic version of entailment is introduced. In the setting we derive, the probabilistic model employed by De Raedt and Kersting (2004) is replaced by an arbitrary statistical classifier. This enables us to also naturally consider, for instance, kernel machines (see Chapter 4), and to tackle classification and regression problems in a uniform framework.

The key step to achieve the proposed integration is to replace the deterministic coverage function employed in ILP by a statistical model. Traditionally, the $covers$ function determines how an example e is classified given a logical theory H by employing logical entailment. In learning from statistical entailment, the statistical part of the model determines how a classification is derived from the logical theory. We will refer to this process as *statistical coverage*. Note that the scoring function given in Problem 2.2.1 also needs to be adapted, to take into account the statistical nature of the overall model. All other elements, such as the examples, background theory and language of clauses $\mathcal{L}$ will be untouched.

In contrast to standard learning from entailment, learning from statistical entailment will enable us to naturally address multi-class classification as well as regression problems. It is therefore useful to introduce a general notation for representing relational examples and their (classification or regression) labels. For standard binary classification, there is a target predicate $p(X_1, ..., X_k)$, and examples are of the form $p(x_1, ..., x_k) = true$ or $p(x_1, ..., x_k) = false$, where the variables $X_1, ..., X_k$ have been instantiated to terms $x_1, ..., x_k$. Multi-class and regression problems can be represented in logic with an additional argument giving the class or regression value, as in $p(x_1, ..., x_k, y)$, where $y \in \{c_1, ..., c_l\}$ or $y \in \mathbb{R}$ (here, l is the number of classes in the multi-class problem). Merging this setting with the more standard notation typically employed in statistical learning, we treat the tuple $(x_1, ..., x_k)$ as an example identifier, and $p(x_1, ..., x_k)$ as a label associated to this identifier. Abusing notation, we also say that $p(x_1, .., x_k) \in \{true, false\}$ for binary classification, $p(x_1, ..., x_k) \in \{1, ..., c\}$ for multi-class problems, and $p(x_1, ..., x_k) \in \mathbb{R}$ for regression problems. Note that, logically speaking, the target predicate $p(X_1, ..., X_k)$ is instantiated to a particular example label $p(x_1, ..., x_k)$ by applying a substitution $\theta = \{X_1/x_1, ..., X_k/x_k\}$. We shall thus also use θ as an example identifier and write $p\theta$ to denote the label of the example θ.

Example 2.4.1. *Reconsider the mutagenicity domain outlined in Example 2.2.2. In the standard binary classification setting, the target predicate to be predicted is $mutagenic(X)$, which can be true or false.*

```
molecule(d26).                                       molecule(d28).
mutagenic(d26).          molecule(d27).              mutagenic(d28).
lumo(d26,-2.072).        lumo(d27,-2.072).           lumo(d28,-2.072).
...                      ...                         ...
```

The facts `mutagenic(d26)` *and* `mutagenic(d28)` *indicate that the molecules with the identifiers* `d26` *and* `d28` *are positive examples. Logically speaking,* `mutagenic(X)` *is instantiated to the example* `d26` *by the substitution $\theta = \{\texttt{X}/\texttt{d26}\}$, and we identify the example* `d26` *with this substitution. Consider now a multi-class setup in which there are three levels of mutagenicity: strong, medium, and low. Examples are now represented by*

```
molecule(d26).         molecule(d27).        molecule(d28).
mutagenic(d26)=high.   mutagenic(d27)=low.   mutagenic(d28)=medium.
lumo(d26,-2.072).      lumo(d27,-2.072).     lumo(d28,-2.072).
...                    ...                   ...
```

and we will say that the example $\theta = \{\texttt{X}/\texttt{d26}\}$ has the label $p\theta =$ `mutagenic(d26)` $\in \{low, medium, high\}$.

To implement statistical coverage in the general form, assume a generic function $\lambda(\theta, H, B)$; given a hypothesis $H = \{q_1, ..., q_m\}$ as a set of clauses q_i and the background knowledge B, $\lambda(\cdot, H, B) : \mathcal{X} \rightarrow \mathcal{Y}$ maps a relational example $\theta \in \mathcal{X}$ to the label $\lambda(\theta, H, B) \in \mathcal{Y}$. For (probabilistic) classification problems, we assume that $\mathcal{Y} = [0, 1]^l$, and $y = \lambda(\theta, H, B)$ gives a distribution over c possible classes and has to satisfy $\sum_{i=1}^{l} y_i = 1$. Other choices for $\mathcal{Y}$ are also possible, for instance, $\mathcal{Y} = \mathbb{R}$ for regression problems (this is discussed in more detail in subsequent chapters).

The objective of learning is still to maximize a generic scoring function S as in Problem 2.2.1. However, the scoring function needs to be adapted to account for the statistical part of the model. Thus, $S(E, H, B)$ is replaced by $S(E, H, \lambda, B)$, where the function λ implements statistical coverage. The exact choice of S depends on the model and learning setting under consideration. Several possible choices will be discussed in detail in Chapter 3 and Chapter 4. The generic maximization problem can now be stated as follows: find

$$\max_{H \in \mathcal{H}} \max_{\lambda \in \Lambda_H} S(E, H, \lambda, B) \tag{2.7}$$

where Λ_H is the space of all statistical coverage functions that can be defined given hypothesis H.

Problem (2.7) consists of jointly optimizing the logical hypothesis H and the statistical coverage function $\lambda(\cdot, H, B)$. In the following, we will refer to the outer optimization problem as *hypothesis* or *structure* learning, and the inner optimization problem as *parameter* or *function* learning. As discussed in Section 2.2, hypothesis learning implies searching in a discrete space of candidates, which is a complex task in general. Thus, heuristic strategies in the spirit of Algorithm 1 will be employed. In contrast, parameter learning takes place in a continuous space, for which other classes of search techniques are available (such as expectation-maximization for probabilistic models and convex optimization algorithms for kernel machines). It is thus not clear whether the scoring functional employed for function learning is also suitable for hypothesis learning. In fact, statistical relational learning systems often employ different scoring functions for learning the logical model structure and the statistical part of the model. Problem (2.7) should therefore be generalized to the following formulation:

$$\max_{H \in \mathcal{H}} S_O(E, H, \arg\max_{\lambda \in \Lambda_H} S_I(E, H, \lambda, B), B), \tag{2.8}$$

where S_O and S_I are the scoring functions used for hypothesis and parameter learning respectively. This is a nested optimization problem, in which the outer optimization of the hypothesis H (according to S_O) involves an inner optimization of the statistical coverage function (according to S_I).

To summarize, Problem 2.2.1 can be extended to account for statistical coverage as follows:

Problem 2.4.1 (Learning from Statistical Entailment)**.**

Given

- *a background theory B, in the form of a set of definite clauses $h \leftarrow b_1, \cdots, b_n$;*
- *a set of examples E, in the form of ground facts with associated (classification or regression) labels;*
- *a language of clauses $\mathcal{L}$, which specifies the clauses that are allowed in hypotheses;*
- *for every $H \subset \mathcal{L}$, a class of functions $\Lambda_H(\theta, H, B)$ to specify the statistical coverage functions under consideration;*
- *an outer scoring function $S_O(E, H, \lambda, B)$, which specifies the quality of the hypothesis H w.r.t. the data E and the background theory;*
- *an inner scoring function $S_I(E, H, \lambda, B)$, which specifies the quality of a statistical coverage function λ w.r.t. the data E and the background theory;*

Find

$$\max_{H \in \mathcal{H}} S_O(E, H, \arg\max_{\lambda \in \Lambda_H} S_I(E, H, \lambda, B), B).$$

Example 2.4.2. *Assume we want to implement the statistical coverage function $\lambda(\cdot, H, B)$ as a naïve Bayes model in a classification setting. Given a hypothesis $H = \{q_1, ..., q_m\}$, we can define a joint probability over clauses in H and class labels according to a naïve Bayes assumption:*

$$\hat{P}(\mathbf{q_1}, ..., \mathbf{q_m}, \mathbf{p}) = \prod_i \hat{P}(\mathbf{q_i}|\mathbf{p})\hat{P}(\mathbf{p}),$$

where $\mathbf{q_i}$ denotes a random variable indicating whether clause q_i succeeds on an example, $\mathbf{p}$ denotes a random variable denoting the class label, and $\hat{P}$ is our estimate of the true joint distribution. The statistical coverage function $\lambda(\cdot, H, B)$ can now be defined as

$$\lambda(\theta, H, B) = \hat{P}(\mathbf{p} \mid q_1\theta, \ldots, q_n\theta)$$

where $q_i\theta$ represents the state of $\mathbf{q_i}$ for example θ. This approach will be discussed in more detail in Chapter 3. Conditional likelihood and standard likelihood will be proposed as outer and inner scoring functions S_O and S_I.

Let us briefly discuss how the standard setting of (logical) learning from entailment given by Problem 2.2.1 can be recovered in the presented setting. This is achieved by fixing λ to be the coverage function,

$$\lambda(\theta, H, B) = \begin{cases} (1,0)^T & \text{if } H \cup B \models p\theta \\ (0,1)^T & \text{otherwise} \end{cases}$$

where $H \cup B \models p\theta$ denotes that H and B together entail a positive classification of the example θ, and using training set accuracy as the score:

$$S(E, H, \lambda, B) = -\frac{1}{m} \sum_{\theta \in E} \mathbb{I}[prediction(\lambda(\theta, H, B)) = p\theta]$$

$$prediction(\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}) = \begin{cases} true : \lambda_1 \geq \lambda_2 \\ false : otherwise \end{cases}$$

and $\mathbb{I}[f]$ is an indicator function which is 1 if f is true and 0 otherwise.

In Chapter 3 and Chapter 4 we propose the NFOIL and KFOIL systems that implement the proposed setting of learning from statistical entailment. In NFOIL, likelihood and conditional likelihood are used as inner and outer scoring functions S_I and S_O. In KFOIL, hinge loss is used as the inner scoring function, and we discuss different outer scoring functions such as kernel target alignment, area under the ROC curve, and root mean squared error.

Note that the formulation of the optimization problem given by Equation (2.8) is rather general, and many existing statistical relational learning approaches can also be cast in this framework. Let us briefly review other instantiations of the framework proposed in the literature. In Bayesian Logic Programs, maximum likelihood is used for both S_O and S_I (Kersting and Raedt, 2007). Regularized weighted pseudo-log-likelihood are employed for S_O and S_I when learning Markov Logic Networks (Kok and Domingos, 2005).

The SAYU system combines the ILP system ALEPH with a (tree augmented) naïve Bayes classifier, using maximum likelihood for S_I and area under the precision-recall curve for S_0 (Davis *et al.*, 2005a). Structural Logistic Regression (Popescul *et al.*, 2003) upgrades logistic regression to the relational setting, and employs a general-to-specific search strategy driven by a regularized maximum likelihood measure for both S_I and S_O. In the RUMBLE margin-based rule learner, f is a linear combination of rules, the inner score S_I is the difference between the empirical mean and variance of the classification margin over training examples; while the outer score S_O is a generalization bound combining the margin minus variance with a capacity term (Rückert and Kramer, 2007). Type Extension Trees (TET) define tree-structured logic formulae, where nodes are conjunctions of literals, and edges are labeled with sets of variables (Frasconi *et al.*, 2008). TET learning has been conducted using pseudo-maximum-likelihood for S_I, and a generic score such as the area under the ROC curve for S_O (Frasconi *et al.*, 2008).

Part I

Efficient SRL for Classification

Outline Part I

The second part of the thesis discusses approaches for integrating ILP with statistical learning according to the framework of learning from statistical entailment (Problem 2.4.1 in Chapter 2). Whereas many approaches in SRL upgrade existing probabilistic learning schemes to deal with relational or logical data (such as Probabilistic Relational Models (Getoor *et al.*, 2001) and Stochastic Logic Programs (Muggleton, 1996)), we start from an inductive logic programming perspective and extend it with statistical techniques. Specifically, we induce a set of first-order clauses as in ILP, but use these as features in a statistical classifier.

The general idea of using clauses in a statistical classifier is not new: it has been explored in a number of propositionalization approaches (see Section 2.3). In traditional propositionalization, the selection of the clause set (relational learning) and the training of the classifier (statistical learning) are two separate steps. This can also be described as a *static* propositionalization approach, in which the propositionalized problem is learned using the statistical classifier. In static propositionalization, the chosen clause set is not optimized for the particular statistical classifier employed. To obtain an accurate final model, most static propositionalization approaches are therefore forced to use large feature sets, for instance, all frequent patterns in the data (Kramer and De Raedt, 2001). However, to preserve interpretability of the final model—one of the most important advantages of relational learning techniques—the final model should be both compact and accurate. A compact and therefore interpretable clause set can only be obtained if the clause set is specifically optimized for use in the statistical classifier.

Prompted by this observation, we propose a *dynamic propositionalization* strategy that tightly integrates statistical and relational learning. In dynamic propositionalization, the search for clauses is driven by the criterion of the statistical classifier, that is, the relational and statistical model are optimized jointly.

We present dynamic propositionalization approaches based on two different statistical learning paradigms: probabilistic graphical models (Chapter 3) and kernels (Chapter 4). According to the setting of learning from statistical entailment, this involves defining an appropriate statistical coverage function, and the inner and outer scoring functions to guide the search process (see Problem 2.4.1). Moreover, we have to specify how the search for a hypothesis with maximum score is carried out. For the sake of simplicity, hypothesis search is implemented using variants of the FOIL algorithm (Quinlan, 1990) for both the graphical model and kernel setting. FOIL is one of the simplest and most well-known ILP algorithms, and implements a greedy hill-climbing search in the space of all clauses structured by generality. However, the presented framework could also accommodate other, more sophisticated ILP search techniques.

Static propositionalization techniques, by taking into account relational structure only via a fixed set of features, avoid the full complexity of statistical relational learning. In the expressivity-efficiency trade-off in SRL they are to be seen as particularly simple and restricted. Dynamic propositionalization approaches extend static propositionalization as the relational part of the model is learned using statistical principles.

A crucial point in dynamic propositionalization is to maintain computational efficiency. To reduce the effort of evaluating candidate clause sets during search, we propose to train the statistical classifier incrementally while the set of first-order clauses is induced. This strategy can significantly reduce complexity, as it enables us to re-use results of previous optimizations in a dynamic programming fashion. Specifically, we discuss incrementally computing naïve Bayes statistics (Section 3.2.3), incrementally learning the structure of tree augmented naïve Bayes (Section 3.3), and incrementally computing kernel matrices, separating max-margin hyperplanes, and kernel target alignments (Section 4.2.2).

The resulting SRL systems are evaluated in several challenging real-world domains, and shown to outperform existing static propositionalization and ILP approaches.

Chapter 3

nFOIL: Integrating Naïve Bayes and FOIL[*]

This chapter presents the nFOIL system as a simple instantiation of the framework of learning from statistical entailment outlined in Section 2.4. We start with the simplest approaches from both domains, the inductive logic programming system FOIL (Quinlan, 1990) and naïve Bayes, and integrate them in the nFOIL system. As the strong independence assumption of naïve Bayes can be problematic in some domains, we furthermore investigate a generalization of naïve Bayes known as tree augmented naïve Bayes in the tFOIL system. Indeed, this methodology could be extended to learning full Bayesian networks; however, the advantage of combining such simple learning schemes is that the resulting probabilistic logical or relational model is easier to understand and to learn.

The chapter is structured as follows. Section 3.1 and Section 3.2 present the nFOIL model and learning algorithm as an instantiation of the framework of learning from statistical entailment (Problem 2.4.1). Section 3.3 presents the extension to tree augmented naïve Bayes. In Section 3.4, the proposed method is evaluated in several real-world domains and compared to standard ILP and static propositionalization approaches.

3.1 Integrating Naïve Bayes and FOIL: Setting

Classical ILP approaches induce a set of clauses that defines a disjunctive hypothesis, as an instance is classified as positive if it satisfies the conditions of one of the rules. On the other hand, a probabilistic model defines a joint probability distribution over a class variable and a set of "attributes" or "features", and the type of model constrains the joint probability distributions that can be represented. A straightforward but powerful

[*]This chapter builds on (Landwehr *et al.*, 2005b, 2007b).

idea to integrate these two approaches is to interpret the clauses or rules as propositional "features" over which a joint probability distribution can be defined. Using naïve Bayes as the probabilistic model, this translates into the statement that "*clauses are independent*".

This idea is not really new. It has been pursued by Pompe and Kononenko (1995) and Davis *et al.* (2004). However, in these traditional approaches for combining ILP and naïve Bayes, one learns the model in two *separate* steps. First, the features or clauses are generated, for example using an existing inductive logic programming system such as ILP-R (Pompe and Kononenko, 1995), and then the probability estimates for the naïve Bayes are determined. This corresponds to a static propositionalization approach, where the propositionalized problem is learned using naïve Bayes. In this chapter, we will instead follow a dynamic propositionalization approach, in which feature construction is tightly coupled with naïve Bayes. Such an approach typically yields more accurate models that still employ small sets of interpretable clauses. Dynamic propositionalization approaches have received increasing attention recently (Landwehr *et al.*, 2005b; Davis *et al.*, 2005a; Landwehr *et al.*, 2007b; Rückert and Kramer, 2007).

We will now introduce a simple statistical relational learner by combining naïve Bayes and FOIL according to the setting of learning from statistical entailment proposed in Section 2.4. For the case of naïve Bayes and FOIL, this will be accomplished by changing the FOIL algorithm to drive the feature search by the criterion of naïve Bayes. The advantage of *dynamic propositionalization* in this setting is that the generated features set will be more appropriate for use in naïve Bayes, compared to rule sets generated by static propositionalization approaches (see Section 3.4).

To implement learning from statistical entailment (Problem 2.4.1), we need to specify 1) the statistical coverage function $\lambda(\theta, H, B)$ and 2) the scoring functions S_O and S_I for hypothesis and parameter learning. Statistical coverage will be implemented by a naïve Bayes model, and as scoring functions we employ conditional likelihood and likelihood respectively.

3.1.1 Statistical Coverage as a Naïve Bayes Model

Let us now define the statistical coverage function $\lambda(\theta, H, B)$ in terms of a naïve Bayes model. The key idea is that we interpret the clauses in H together with the example e as queries or features, as outlined in Section 2.3. More formally, let H contain a set of clauses $\{q_1, ..., q_m\}$ defining the predicate $p(X_1, ..., X_k)$. Then we view each clause q of the form

$$p(X_1, ..., X_k) \leftarrow b_1, \cdots, b_n$$

as a boolean feature or attribute. Such a clause covers an example $\theta \in E$ iff $B \cup \{q\} \models p\theta$. From a statistical perspective, we will say that the random variable $\mathbf{q}$ takes on value $q\theta \in \{true, false\}$ for the example θ. Similarly, we consider the random variable $\mathbf{p}$ denoting the class of an example, which takes on the value $p\theta$ for a particular example θ (note that $p\theta$ can be binary or multi-valued as explained in Section 2.4). In analogy to Section 2.1, we use $P(\mathbf{q})$ to denote the distribution over the random variable $\mathbf{q}$, and use the notation $P(q\theta)$ as a shorthand for $P(\mathbf{q} = q\theta)$ to denote the probability of the value $q\theta$ for the random variable $\mathbf{q}$.

Example 3.1.1. *Reconsider the mutagenicity problem outlined in Example 2.2.2, assuming the representation of molecules is given in the background knowledge. Let the query* q_c *be given by*

$$mutagenic(X) \leftarrow atom(X, A, o, 40, C), bond(X, B, A, 2)$$

For the example $\theta = \{X/d26\}$*, the instantiated query*

$$mutagenic(d26) \leftarrow atom(d26, A, o, 40, C), bond(d26, B, A, 2)$$

succeeds in the background theory, so the boolean random variable $\mathbf{q}$ *takes on value* $q\theta = true$ *for this example. As indicated by the fact* $mutagenic(d26)$*,* $d26$ *is a positive example. Thus, the random variable* $\mathbf{p}$ *takes on the value* $p\theta = true$ *for this example.*

To implement the statistical coverage function λ, we need to define a distribution over possible class labels for a given example θ. This is realized by defining a probabilistic model λ that specifies a joint distribution $P_\lambda(\mathbf{p}, \mathbf{q_1}, ..., \mathbf{q_m})$ over the random variables $\mathbf{p}$ and $\mathbf{q_1}, ..., \mathbf{q_m}$ given $H = \{q_1, ..., q_m\}$. Assume there are l classes $\{c_1, ..., c_l\}$. Define

$$\lambda(\theta, H, B) = \begin{pmatrix} \pi_1 \\ \cdots \\ \pi_l \end{pmatrix}$$

where

$$\pi_i = P_\lambda(\mathbf{p} = c_i \mid \mathbf{q_1} = q_1\theta, ..., \mathbf{q_m} = q_m\theta)$$

and

$$P_\lambda(\mathbf{p}|\mathbf{q}_1, \ldots, \mathbf{q}_m) = \frac{P_\lambda(\mathbf{p}, \mathbf{q}_1, \ldots, \mathbf{q}_m)}{P_\lambda(\mathbf{q}_1, \ldots, \mathbf{q}_m)}.$$

Here, we employ the usual notation for equality of distributions, representing the equality of probabilities for all instantiations of the random variables $\mathbf{p}, \mathbf{q}_1, ..., \mathbf{q}_m$. Now it becomes possible to state the naïve Bayes assumption

$$P_\lambda(\mathbf{q_1}, ..., \mathbf{q_m}|\mathbf{p}) = \prod_i P_\lambda(\mathbf{q_i}|\mathbf{p})$$

and apply it to the statistical coverage function λ:

$$P_\lambda(\mathbf{p} \mid \mathbf{q}_1, ..., \mathbf{q}_m) = \frac{\prod_i P_\lambda(\mathbf{q}_i \mid \mathbf{p}) \cdot P_\lambda(\mathbf{p})}{P_\lambda(\mathbf{q}_1, \cdots, \mathbf{q}_m)} \tag{3.1}$$

where

$$\begin{aligned} P_\lambda(\mathbf{q}_1, \cdots, \mathbf{q}_m) &= \sum_{i=1}^{l} P_\lambda(\mathbf{p} = c_i, \mathbf{q}_1, ..., \mathbf{q}_m) \\ &= \sum_{i=1}^{l} \prod_{j=1}^{m} P_\lambda(\mathbf{q}_j \mid \mathbf{p} = c_i) \cdot P_\lambda(\mathbf{p} = c_i). \end{aligned}$$

Equation (3.1) specifies the parameters of the naïve Bayes model λ, which are the distributions $P_\lambda(\mathbf{q_i}|\mathbf{p})$ and the prior class distribution $P_\lambda(\mathbf{p})$.

Example 3.1.2. *Reconsider the mutagenicity example, and assume that the hypothesis is as sketched before. Then the queries q_1 and q_2 are*

$$\begin{aligned} &mutagenic(X) \leftarrow atom(X, A, o, 40, C), bond(X, B, A, 2) \\ &mutagenic(X) \leftarrow atom(X, A, c, 22, C), atom(X, B, E, 22, 0.02), bond(X, A, B, 7) \end{aligned}$$

and the target predicate p is "mutagenic(X)". Now assume a naïve Bayes model with probability distributions $P_\lambda(\mathbf{q}_i|p), P_\lambda(\mathbf{p})$ given as

$$\begin{array}{lcl} P_\lambda(\mathbf{p} = t) = 0.6, & & \\ P_\lambda(\mathbf{q_1} = t|\mathbf{p} = t) = 0.7 & , & P_\lambda(\mathbf{q_1} = t|\mathbf{p} = f) = 0.4, \\ P_\lambda(\mathbf{q_2} = t|\mathbf{p} = t) = 0.2 & , & P_\lambda(\mathbf{q_2} = t|\mathbf{p} = f) = 0.1. \end{array}$$

Summing out yields

$$\begin{array}{ll} P_\lambda(\mathbf{q_1} = t, \mathbf{q_2} = t) = 0.10, & P_\lambda(\mathbf{q_1} = t, \mathbf{q_2} = f) = 0.48, \\ P_\lambda(\mathbf{q_1} = f, \mathbf{q_2} = t) = 0.06, & P_\lambda(\mathbf{q_1} = f, \mathbf{q_2} = f) = 0.36 \end{array}$$

where t (f) denotes $true$ ($false$). For the positively labeled example $\theta = \{X/d26\}$, q_1 succeeds and q_2 fails: $p\theta = true$, $q_1\theta = true$, $q_2\theta = false$. Thus,

$$P_\lambda(p\theta \mid H, \lambda, B) = \frac{P_\lambda(q_1\theta|p\theta) \cdot P_\lambda(q_2\theta|p\theta) \cdot P_\lambda(p\theta)}{P_\lambda(q_1\theta, q_2\theta)} = \frac{0.7 \cdot 0.8 \cdot 0.6}{0.48} = 0.7.$$

3.1.2 Scoring Functions

To complete the specification of the framework given in Section 2.4, we still need to define the inner and outer scoring functions in nFOIL. We are interested in using a probabilistic scoring function that reflects the probabilistic nature of the overall model. A natural choice is to maximize the likelihood of the observed example labels. The likelihood for a particular example θ is

$$P(p\theta \mid H, \lambda, B) = \lambda(\theta, H, B)^T \mathbf{u}_i \tag{3.2}$$

where $p\theta = c_i \in \{c_1, ..., c_l\}$ is the observed label for example θ, and $\mathbf{u}_i \in \{0,1\}^m$ is a vector whose i-th element is 1 and whose other elements are 0.

As outer scoring function S_O, we can now employ the likelihood of the observed labels given the relational description of the examples. Thus,

$$\begin{aligned} S_O(E, H, \lambda, B) &= P(E \mid H, \lambda, B) \\ &= \prod_{\theta \in E} P_\lambda(p\theta | q_1\theta, ..., q_m\theta) \end{aligned} \tag{3.3}$$

where $H = \{q_1, ..., q_m\}$, $p\theta$ is the observed label of example θ, $P_\lambda(p\theta|q_1\theta, ..., q_m\theta)$ is defined by Equation (3.1), and we assume that examples are independently and identically distributed (i.i.d.).

The inner scoring function S_I is used to choose the probabilistic model λ (that is, the naïve Bayes parameters encoding the distributions $P_\lambda(\mathbf{q_i}|\mathbf{p}), P_\lambda(\mathbf{p})$). Here, we employ the standard likelihood function:

$$S_I(E, H, \lambda, B) = \prod_{\theta \in E} P_\lambda(p\theta, q_1\theta, ..., q_m\theta) \tag{3.4}$$

for which efficient closed-form optimization is possible. The choice of scoring functions is discussed in more detail in Section 3.2.2.

3.2 Integrating Naïve Bayes and FOIL: Learning

The goal of learning is to find a hypothesis H with maximum score. In our case, as in standard ILP (cf. Section 2.2), the hypothesis space to search is given by a language bias $\mathcal{L}$, that defines the set of clauses that are admissible in hypothesis. Specifically, we define $\mathcal{L}$ by means of *rmode* and *type* declarations (see Section 2.2.3).

Learning in nFOIL involves searching for a combined statistical-logical model. Roughly speaking, existing approaches pursued by Pompe and Kononenko (1995) and Davis *et al.* (2004) have solved this task in a two-step process: First, a hypothesis H is found (using a standard ILP system, and thus some deterministic score) and fixed; second, the parameters for the fixed structure are optimized using a probabilistic score (usually, maximum likelihood). In terms of the inner and outer scoring defined in Section 2.4, this correspond to an outer scoring function $S_O^{ILP}(E, H, B)$ for scoring hypotheses H that is independent of the probabilistic model λ used in the final statistical relational model:

```
**for** all $H \in candidates(\mathcal{L})$ **do**
    compute $S_O^{ILP}(E, H, B)$
**end for**
find $H^* = \arg\max_H S_O^{ILP}(E, H, B)$
find $\lambda^* = \arg\max_{\lambda \in \Lambda_{H^*}} S_I(E, H^*, \lambda, B)$
```

The disadvantage of this approach is that the set of clauses is not selected to yield optimal performance of the statistical model. In contrast, Problem 2.4.1 proposes a joint optimization of the logical and statistical model. In the outer loop, a search through the (discrete) hypothesis space is carried out. An individual hypothesis H is scored by augmenting it with a statistical classifier λ_H and then evaluating the performance of the joint model by a scoring function $S_O(E, H, \lambda, B)$. This translates into the following procedure:

```
**for** all $H$ **do**
    find $\lambda_H = \arg\max_{\lambda \in \Lambda_H} S_I(E, H, \lambda, B)$
    compute $S_O(E, H, \lambda_H, B)$
**end for**
find $H^* = \arg\max_{H} S_O(E, H, \lambda_H, B)$
```

Thus, the search for the structure is guided directly by the probabilistic objective function. It still needs to be discussed how the search in the space of candidate hypotheses is carried out, as this space is clearly too large to be searched exhaustively. The rest of this section shows how this can be realized by modifying the original search technique used in FOIL. This will first be presented for learning the basic NFOIL model. An extension of the learning algorithm that accounts for the tree augmented naïve Bayes structure in TFOIL will be discussed in Section 3.3.

3.2.1 Adapting FOIL

The generic FOIL algorithm has already been presented in Section 2.2.3. Like FOIL, NFOIL searches a set of clauses greedily, and a single clause in a general-to-specific manner using a refinement operator. The main difference in the search technique is that FOIL can use a *separate-and-conquer* approach. Because the final model in FOIL is the disjunction of the learned clauses (where every clause covers a certain subset of examples), it holds that

1. Examples that are already covered do not have to be considered when learning additional clauses: $\mathit{update}(E, H) = E \setminus \mathit{covered}(H)$.
2. (Non-recursive) clauses already learned do not need to be considered when scoring additional clauses: $\mathit{score}(E, H \cup \{c'\}, B) = \mathit{score}(E, \{c'\}, B)$.

Here, $\mathit{score}(E, \{c'\}, B)$ is some accuracy-related measure such as information gain or the m-estimate. In NFOIL, such a separate-and-conquer approach is not possible because every clause can affect the likelihood of all examples. Consequently, the NFOIL algorithm can be obtained from FOIL by changing two components in the generic FOIL algorithm (see Algorithm 1):

1. The set of examples is not changed after learning a clause: $\mathit{update}(E, H) = E$.
2. An additional clause c' has to be scored together with the current model, according to the outer scoring functional S_O:
$$\mathit{score}(E, H \cup \{c'\}, B) = S_O(E, H \cup \{c'\}, \lambda^*, B)$$
where $\lambda^* = \arg\max_{\lambda} S_I(E, H \cup \{c'\}, \lambda, B)$ is the optimal statistical coverage function according to S_I.

We also have to modify the stopping criterion. The most basic stopping criterion for FOIL stops when all positive examples are covered. This is replaced in nFOIL by stopping if the change in score when adding a clause falls below a certain threshold. In general, this simple criterion might lead to overfitting. We therefore also investigated post-pruning the learned hypothesis (Section 3.4). In the experiments, however, this basic stopping criterion worked surprisingly well.

3.2.2 Parameter Estimation

To evaluate a set of clauses $H = \{q_1, ..., q_m\}$, one needs to solve the "inner" optimization problem of finding an optimal statistical coverage function λ. In the case of nFOIL, this means estimating parameters for the naïve Bayes model over the random variables $\{\mathbf{p}, \mathbf{q_1}, ..., \mathbf{q_m}\}$. One possibility is to use the proposed outer scoring function (Equation (3.3)) as the optimization criterion:

$$\begin{aligned}
\lambda^* &= \arg\max_{\lambda} P(E \mid H, B) \\
&= \arg\max_{\lambda} \prod_{\theta \in E} P_\lambda(p\theta | q_1\theta, \cdots, q_m\theta) \\
&= \arg\max_{\lambda} \prod_{\theta \in E} \frac{\prod_j P_\lambda(q_j\theta \mid p\theta) \cdot P_\lambda(p\theta)}{P_\lambda(q_1\theta, ..., q_m\theta)}. \qquad (3.5)
\end{aligned}$$

However, these are the parameters maximizing the *conditional likelihood* of the observed class labels given the model. Usually, naïve Bayes (as a generative model) is trained to maximize the *likelihood*

$$\begin{aligned}
\prod_{\theta \in E} P_\lambda(p\theta, q_1\theta, ..., q_m\theta) &= \prod_{\theta \in E} P_\lambda(p\theta | q_1\theta, \cdots, q_m\theta) \cdot P_\lambda(q_1\theta, ..., q_m\theta) \qquad (3.6) \\
&= \prod_{\theta \in E} \prod_j P_\lambda(q_j\theta \mid p\theta) \cdot P_\lambda(p\theta).
\end{aligned}$$

Maximum *likelihood* parameters can be computed easily by

$$P_\lambda(\mathbf{q_i} = q_i | \mathbf{p} = p) = \frac{n(\mathbf{q_i} = q_i, \mathbf{p} = p)}{n(\mathbf{p} = p)}$$

where $n(X)$ are the *counts*, that is, the number of examples for which the query X succeeds. However, there is no closed-form solution for maximum *conditional likelihood* parameters, and relatively slow iterative optimization algorithms have to be used.

Could we use the likelihood as defined by Equation (3.6) as the outer scoring function S_O, instead of using the conditional likelihood defined by Equation (3.5)? The problem is that this term is dominated by $P_\lambda(q_1\theta, ..., q_m\theta)$, the likelihood of the propositional dataset that is obtained by evaluating all features. This likelihood is easily maximized for very non-uniform distributions over the features values. In fact, it can be maximized to 1 if all features are constant (always succeed or always fail). Such features are of course completely uninformative with respect to predicting class labels. In contrast, in Equation (3.5) the likelihood is corrected by the term $P_\lambda(q_1\theta, ..., q_m\theta)$, and in this way informative features are selected. Example 3.2.1 illustrates this situation for a single feature. Note that in the setting of parameter estimation from propositional data this problem does not occur as the distribution over the attribute values is determined by the (fixed) training data and cannot be changed by the learning algorithm. However, similar considerations apply to feature selection problems for (propositional) probabilistic models.

Example 3.2.1. *Consider the following queries q_1 and q_2:*

$$\begin{array}{l} positive(X) \leftarrow true \\ positive(X) \leftarrow \textit{perfect-literal}(X) \end{array}$$

Query q_1 succeeds on all examples. Assume that query q_2 succeeds on all positive and no negative examples, and that half of the examples are positive. Given maximum likelihood parameters, the models H_1/H_2 consisting of only q_1/q_2 actually have the same *likelihood, while conditional likelihood correctly favors H_2:*
For any example θ,

$$\begin{aligned} P_\lambda(p\theta, q_1\theta) &= P_\lambda(q_1\theta|p\theta) \cdot P_\lambda(p\theta) = 1 \cdot 0.5 = 0.5, \\ P_\lambda(p\theta, q_2\theta) &= P_\lambda(q_2\theta|p\theta) \cdot P_\lambda(p\theta) = 1 \cdot 0.5 = 0.5 \end{aligned}$$

as $P_\lambda(q_1\theta|p\theta) = P_\lambda(q_2\theta|p\theta) = 1$. On the other hand,

$$\begin{aligned} P_\lambda(p\theta|q_1\theta) &= \frac{P_\lambda(q_1\theta|p\theta) \cdot P_\lambda(p\theta)}{P_\lambda(q_1\theta)} = \frac{1 \cdot 0.5}{1} = 0.5, \\ P_\lambda(p\theta|q_2\theta) &= \frac{P_\lambda(q_2\theta|p\theta) \cdot P_\lambda(p\theta)}{P_\lambda(q_2\theta)} = \frac{1 \cdot 0.5}{0.5} = 1 \end{aligned}$$

as $P_\lambda(q_1\theta) = 1$ but $P_\lambda(q_2\theta) = 0.5$.

A similar problem has been noted by Grossman and Domingos (2004) when learning the structure of Bayesian networks. There, likelihood maximization leads to over-connected structures, a problem which is also solved by maximizing conditional likelihood. Because finding maximum conditional likelihood parameters is computationally too expensive, Grossman and Domingos propose a "mixed" approach: use conditional likelihood as score, but set parameters to their maximum likelihood values.

For nFOIL, we follow the same approach. Parameters are estimated to maximize the likelihood (Equation (3.6)), while the conditional likelihood, see Equation (3.5), is retained as score for selecting the features. Note that if the naïve Bayes assumption is correct, the maximum likelihood estimates of the parameters will approach the maximum conditional likelihood estimates as the number of training examples goes to infinity. This is because the maximum likelihood estimate $P_\lambda(\mathbf{p}\theta, \mathbf{q_1}\theta, ..., \mathbf{q_m}\theta)$ approaches the true joint probability $P(\mathbf{p}\theta, \mathbf{q_1}\theta, ..., \mathbf{q_m}\theta)$ which determines the true conditional $P(\mathbf{p}\theta|\mathbf{q_1}\theta, ..., \mathbf{q_m}\theta)$ that also maximizes the conditional likelihood (Friedman and Goldszmidt, 1996). In this sense, maximizing likelihood can be seen as an approximation to maximizing conditional likelihood.

3.2.3 Computational Complexity

Compared to the original FOIL algorithm, there is little computational overhead in nFOIL. To see this, let us first analyze the computational complexity of FOIL. When learning a new clause (one iteration of the algorithm), FOIL has to evaluate a number of candidate clauses on the data. This step clearly dominates computational costs. To evaluate a particular candidate clause q, we need to obtain the sets of positive and negative examples it covers. From this information, the actual score (accuracy, information gain, or a related measure) can be computed in linear time. Thus, the main task is to check for every example e whether $H \cup \{q\} \models e$. The computational complexity of this step, which basically involves performing an SLD resolution, strongly depends on the characteristics of the examples and the clause to be evaluated. Moreover, speedup strategies such as caching can be employed. We will not discuss this in detail, and only denote the average time spent on checking whether a clause q covers an example e by $sld(\mathcal{L}, E, B)$. Note that this task needs to be solved for any ILP system including statistical logical learners such as nFOIL, and the same algorithmic strategies can be employed to speed up this task in different approaches.

The costs for scoring an individual clause q in FOIL are thus of the order $O(|\hat{E}| \cdot sld(\mathcal{L}, E, B))$, where $\hat{E}$ is the set of examples currently under consideration. Note that in FOIL, the set of examples $\hat{E}$ is reduced at every iteration, as covered examples are removed. However, the size $|\hat{E}|$ of the current example set is bounded from below by $\frac{N^-}{N^+ + N^-}|E|$, where N^+ (N^-) are the number of examples in E classified as positive (negative) by the final theory H. Assuming there are m clauses in the final model, overall time for learning a model is thus in $O(\frac{N^-}{N^+ + N^-}|E| \cdot m \cdot sld(\mathcal{L}, E, B))$. Note that unless there are vastly more positive than negative examples (an unusual situation in ILP), the factor $\frac{N^-}{N^+ + N^-}$ will not affect runtime significantly.

In nFOIL, a similar evaluation of candidate clauses takes place. In contrast to FOIL, the set of examples under consideration does not change between iterations. Computing the set of covered examples for every candidate clause encountered during the search will thus take time $O(|E| \cdot m \cdot sld(\mathcal{L}, E, B))$ (again, m is the number of clauses in the final model). To compute the likelihood score of the model defined by $H \cup \{q\}$, the corresponding naïve Bayes parameters need to be estimated, which are $P(\mathbf{q}_1 \mid \mathbf{p}), ..., P(\mathbf{q}_a \mid \mathbf{p})$ and $P(\mathbf{p})$ assuming $q = q_a$ (for $a \leq m$) is the candidate clause being added to the model. However, parameters of different clauses are independent due to the naïve Bayes assumption, hence only the distribution $P(\mathbf{q}_a \mid \mathbf{p})$ needs to be estimated, which takes time $O(|E| \cdot l)$ given the set of positive and negative examples covered (where l is the number of classes). Computing the data likelihood defined by Equations (3.1) and (3.3) takes time $O(|E| \cdot m \cdot l)$ for a single candidate clause. Thus, overall time spent on computing the likelihood takes time $O(|E| \cdot m^2 \cdot l)$. The quadratic term m^2 can be removed by incrementally computing the likelihood term as follows. Define

$$\gamma[i][a] = \prod_{j=1}^{a} P_\lambda(\mathbf{q}_j \mid \mathbf{p} = c_i),$$

then

$$\gamma[i][a] = \gamma[i][a-1] P_\lambda(\mathbf{q}_a \mid \mathbf{p} = c_i)$$

can be incrementally computed for all $a \in \{1, ..., m\}$ in overall time $O(l \cdot m)$. The likelihood of example θ can now be computed using

$$P(\mathbf{p} = c_i \mid \mathbf{q}_1, ..., \mathbf{q}_m) = \frac{P(\mathbf{p} = c_i)\gamma[i][m]}{\sum_{j=1}^{l} \gamma[j][m]}$$

in time $O(l)$. Thus, overall time spent on computing the likelihood scores is reduced to $O(|E| \cdot m \cdot l)$ and overall runtime in NFOIL to $O(|E| \cdot l \cdot m \cdot sld(\mathcal{L}, E, B))$, which is essentially the same complexity as in FOIL.

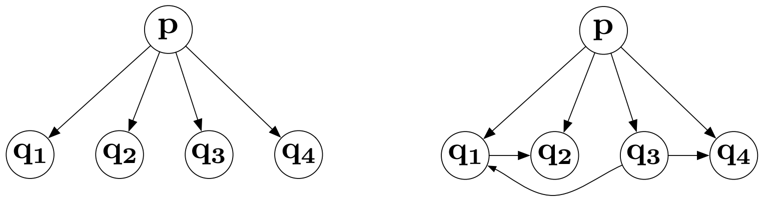
Finally, we note that the actual total runtime of FOIL and NFOIL depends on the path in the search space that is taken by the two algorithms, and the number of clauses learned. This also depends on the language bias and stopping criterion employed. However, we can summarize that NFOIL, compared to FOIL, only incurs a small computational overhead.

3.3 TFOIL: Relaxing the Naïve Bayes Assumption

The naïve Bayes model employed in NFOIL corresponds to the strong assumption that the probability of an example satisfying one query is independent of its probability to satisfy another query, given the class of the example:

$$P_\lambda(\mathbf{q_1}, ..., \mathbf{q_m} | \mathbf{p}) = \prod_i P_\lambda(\mathbf{q_i} | \mathbf{p}). \tag{3.7}$$

Although it has been shown that naïve Bayes can perform well in practice even if this assumption is violated, better models can sometimes be constructed by relaxing this strict independence assumption (Friedman and Goldszmidt, 1996). Tree augmented naïve Bayes (TAN) models generalize naïve Bayes by allowing additional dependencies, but are still significantly more restrictive than full Bayesian networks. The restriction compared to a full Bayesian network is that the additional dependencies form a tree (i.e, every query node has at most one additional parent). This means that the number of parameters of a



(a) Naïve Bayes. (b) Tree augmented naïve Bayes.

Figure 3.1: Graphical model representation of naïve Bayes and tree augmented naïve Bayes model. The random variables $\mathbf{p}$ and $\mathbf{q_1}, ..., \mathbf{q_4}$ denote the class and features.

TAN model is in $O(\#nodes)$ as compared to $O(2^{\#nodes})$ for a full Bayesian network, and learning them is generally easier. Figure 3.1 shows an example for a tree augmented naïve Bayes model in graphical model notation, and compares it to the standard naïve Bayes structure.

In this section, we will discuss the TFOIL algorithm, which extends the NFOIL algorithm presented above to employ tree augmented naïve Bayes. Under the TAN assumption, Equation (3.7) is relaxed to

$$P_\lambda(\mathbf{q_1}, ..., \mathbf{q_m}|\mathbf{p}) = \prod_i P_\lambda(\mathbf{q_i}|\mathbf{p}, \mathbf{q_{pa(i)}})$$

where $\mathbf{q_{pa(i)}}$ is the additional parent of the node $\mathbf{q_i}$ in the TAN model. In analogy to Section 3.1.1, the statistical coverage function can be re-derived as

$$P(\mathbf{p} \mid \mathbf{q}_1, ..., \mathbf{q}_m) = \frac{\prod_i P_\lambda(\mathbf{q}_i \mid \mathbf{p}, \mathbf{q}_{pa(i)}) \cdot P_\lambda(\mathbf{p})}{P_\lambda(\mathbf{q}_1, \cdots, \mathbf{q}_m)} \tag{3.8}$$

Otherwise, the problem specification outlined in Section 3.1.1 directly carries over to TFOIL.

In contrast to NFOIL, learning a TFOIL model involves the additional task of selecting a TAN structure over the random variables representing the logical queries. In general, TFOIL follows the same integrated search strategy as outlined for NFOIL in Section 3.2. However, when evaluating a set of clauses $H_C = \{q_1, ..., q_m\}$ in TFOIL, for every clause $q_i \in H_C$ we have to decide on a $q_j = q_{pa(i)} \in H_C$ for which a dependency $q_{pa(i)} \rightarrow q_i$ is added. Of course, this could be accomplished by running the standard TAN structure learning algorithm, which finds a maximum-likelihood structure in polynomial time (Friedman and Goldszmidt, 1996). However, the incremental way in which a theory H is learned and query nodes are added to the probabilistic model suggests a faster, incremental (though heuristic) way of learning the TAN structure. Rather than re-learning the TAN graph from scratch every time a new candidate clause q_{m+1} is scored, the subgraph over the existing hypothesis $H = \{q_1, ..., q_m\}$ is kept fixed, and all existing clauses $q_j \in H$ are considered as possible parents for the clause q_i. Out of

Algorithm 2 The TFOIL algorithm.

```
Initialize $H := \emptyset$
repeat
    Initialize $c := p(X_1, \cdots, X_k) \leftarrow$
    repeat
        for all $c' \in \rho(c)$ do
            for all $q \in H$ do
                compute $s(c', q) = score(E, H \cup \{c'\}, q \rightarrow c', B)$
            end for
            let $pa(c')$ be the $q \in H$ with maximum $s(c', q)$
        end for
        let $c$ be the $c' \in \rho(c)$ with maximum $s(c', pa(c'))$
    until stopping criterion
    add $c$ with dependency $pa(c') \rightarrow c'$ to $H$
until stopping criterion
output $H$
```

these candidate graph structures, the one maximizing $score(E, H, B)$ is chosen, that is, the maximum-likelihood extension of the existing graph on H.

This approach is outlined in Algorithm 2. The function $score(E, H \cup \{c'\}, q \rightarrow c', B)$ returns the score of the TAN model over $H \cup \{c'\}$, where $\mathbf{q}$ is the additional parent of $\mathbf{c}'$. For every candidate clause c', the best parent $pa(c')$ is identified and the c' with highest score is added to H with the dependency $pa(c') \rightarrow c'$. Comparing Algorithm 2 with the NFOIL algorithm which follows the template of Algorithm parent of a clause that is added to the model. Note that the computational complexity of scoring the $(m + 1)th$ clause q_{m+1} in TFOIL, given we know the set of examples on which the clause succeeds, is $O(|E| \cdot m \cdot l)$, as all m existing clauses are considered as possible parents. Thus, the overall time for computing scores (again excluding coverage computations) in TFOIL is $O(|E| \cdot m^2 \cdot l)$, adding a factor of m compared to the NFOIL algorithm.

3.4 Experimental Evaluation

This section presents an experimental evaluation of the proposed NFOIL and TFOIL systems. More specifically, our intention is to investigate how dynamic propositionalization compares to static propositionalization approaches that use naïve Bayes only to post-process the rule set. The experimental study will address the following questions:

(Q3.1) Is there a gain in predictive accuracy of a dynamic propositionalization approach over its ILP baseline?

(Q3.2) If so, is the gain of dynamic propositionalization over its baseline larger than the gain of static propositionalization approaches?

(Q3.3) Can performance be improved by using a more expressive probabilistic model, such as tree augmented naïve Bayes?

(Q3.4) Is a dynamic propositionalization based on a simple rule learner competitive with advanced ILP approaches?

(Q3.5) Relational naïve Bayes methods such as 1BC2 (Flach and Lachiche, 2004) essentially employ all clauses within a given language bias as features in a probabilistic model and thus perform static propositionalization. Does dynamic propositionalization employ fewer features and perform better than these approaches?

In the following two subsections, we will describe the datasets and algorithms used to experimentally investigate the questions **(Q3.1)**–**(Q3.5)**. Section 3.4.3 then presents and discusses the results.

3.4.1 Datasets

We conducted experiments on eight datasets from four domains. Three domains are binary classification tasks, and one is a multi-class problem. See Table 3.1 for an overview of the different datasets. Most of the domains are concerned with *structure-activity relationship* (or *SAR*) problems. SAR problems are of central importance in many areas of bio- and chemoinformatics: given information about the chemical structure of a substance, predict its activity with regard to a certain property of interest. This property can be to activate or block a receptor in the human body, toxicity, suppression of tumor growth or more generally activity as a pharmacological agent. The search for substances with such properties currently relies on screening trials (so-called *bioassays*), which measure the activity of hundreds or thousands of chemical compounds. This data is an interesting application area for machine learning, and in particular a good showcase for learning from structured data. If good models that relate compound structure to activity can be built, some of the tests currently performed in laboratories could be replaced by automatic classification, greatly accelerating the screening process.

Mutagenesis is a well-known domain for structure-activity relation prediction (Srinivasan *et al.*, 1996). The problem is to classify compounds as mutagenic or not given their chemical structure described in terms of atoms, bonds, atom charge, and information about atom and bond types that have been generated by the molecular modeling package QUANTA. No additional numeric or hand-crafted features of the compounds are used. The dataset is divided into two sets: a regression friendly (**r.f.**) set with 188 entries (125 positives, 63 negatives) and a regression unfriendly (**r.u.**) set with 42 entries (13 positives and 29 negatives).

For **Alzheimer**, the aim is to compare 37 analogues of Tacrine, a drug against Alzheimer's disease, according to four desirable properties: inhibit **amine** re-uptake, low **toxicity**, high **acetyl** cholinesterase inhibition, and good **reversal** of scopolamine-induced memory deficiency (King *et al.*, 1995). For any property, examples consist of pairs $pos(X,Y)/neg(X,Y)$ of two analogues indicating that X is better/worse than Y

Table 3.1: datasets used in nFOIL/tFOIL experiments.

Dataset	**#Classes**	**#Examples**	**Majority Class**	**#Relations**	**#Facts**
Mutagenesis r.f.	2	188	66.5%	4	10324
Mutagenesis r.u.	2	42	69.1%	4	2109
Alzheimer amine	2	686	50.0%	20	3754
Alzheimer toxic	2	886	50.0%	20	3754
Alzheimer acetyl	2	1326	50.0%	20	3754
Alzheimer memory	2	642	50.0%	20	3754
NCTRER	2	232	56.5%	3	9283
Diterpene	23	1530	23.5%	17	33068

w.r.t. the property. The relation is transitive and anti-symmetric but not complete (for some pairs of compounds the result of the comparison could not be determined).

The **NCTRER** dataset has been extracted from the EPA's DSSTox NCTRER Database (Fang *et al.*, 2001). It contains structural information about a diverse set of 232 natural, synthetic and environmental estrogens and classifications with regard to their binding activity for the estrogen receptor. In our experiments, only structural information, that is, atom elements and bonds are used. Additionally, we provided a relation $linked(A_1, A_2, E, BT)$ in the background knowledge that represents that there is a bond of type BT from atom A_1 to atom A_2 and A_2 is of element E. This was done to reduce the lookahead problem for greedy search algorithms.

For **Diterpene**, the task is to identify the skeleton of diterpenoid compounds, given their C-NMR spectra that include the multiplicities and the frequencies of the skeleton atoms (Džeroski *et al.*, 1998). Diterpenes are organic compounds of low molecular weight with a skeleton of 20 carbon atoms. They are of interest because of their use as lead compounds in the search for new pharmaceutical effectors. The dataset contains information on 1530 diterpenes with known structure. There are in total 23 classes. We use the version where both relational and propositional information about the NMR spectra are available.

3.4.2 Algorithms and Methodology

We investigate the following learners:

- **nFOIL**
  An implementation of the nFOIL algorithm as outlined in Section 3.2.1. Instead of a greedy search a beam search with beam size $k = 5$ is performed. During the search for a clause, the algorithm also keeps a set C^* of the k best (partial) clauses found so far. If this set does not change from one refinement level to the next, the search is stopped and the best element $c \in C^*$ is returned. The search for additional clauses is stopped if the change in score between two successive iterations is less

than 0.1%. A hypothesis is limited to contain at most 25 clauses and a clause to contain at most 10 literals. As most other ILP systems, nFOIL can make use of intensionally specified background knowledge to be used in hypothesis. When classifying unseen examples, the class receiving the highest probability is returned (in particular, the default classification threshold of 0.5 is used in the binary case).

- **tFOIL**
  An implementation of the tFOIL algorithm as outlined in Section 3.3. The beam search and stopping criterion are implemented as for nFOIL.

- **mFOIL**
  mFOIL is a variant of FOIL also employing beam search and different search heuristics (Lavrac and Dzeroski, 1994). The beam size is set to $k = 5$, otherwise, default parameters are used. Note that mFOIL, unlike the other systems considered, by default allows negated literals in clauses.

- **Aleph**
  Aleph is an advanced ILP System developed by Ashwin Srinivasan.[1] It is based on the concept of *bottom clauses*, which are maximally specific clauses covering a certain example. The theory is then built from clauses that contain a subset of the literals found in a bottom clause. We used this standard mode of Aleph for the binary domains **Mutagenesis**, **Alzheimer** and **NCTRER**. Additionally, Aleph implements a tree learning algorithm, which we used for the multi-class domain **Diterpene**. The maximum number of literals in a clause was set to 10 instead of the default 4. Otherwise, default settings are used except on **NCTRER** as explained below.

- **1BC2**
  1BC2 is a naïve Bayes classifier for structured data (Flach and Lachiche, 2004). 1BC2 was run with a maximum number of 5 literals per clause, as larger values caused the system to crash. The decision threshold was optimized based on a five fold cross validation.

It would also be interesting to compare against the MACCENT system (Dehaspe, 1997), as maximum entropy models and naïve Bayes are somewhat related. Unfortunately, only an implementation of a propositional version of MACCENT is available, which only handles data in attribute-value (vector) format.[2] This version is not directly applicable to the relational datasets used in our study. We have therefore investigated a static propositionalization approach: frequent clauses were extracted from the relational datasets and then used as features in the propositional MACCENT system. More precisely, we have used a variant of the frequent pattern miner WARMR (Dehaspe *et al.*, 1998), as WARMR patterns have shown to be effective propositionalization techniques on similar benchmarks in

[1]More information on Aleph can be found at `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph\_toc.html`.

[2]Luc Dehaspe, from personal communication.

Table 3.2: Cross-validated predictive accuracy results on all datasets. ●/○ indicates that tFOIL's mean is significantly higher/lower, and ▲/△ that nFOIL's mean is significantly higher/lower (paired sampled t-test, $p = 0.05$). Bold numbers indicate the best result on a dataset. Note that the high variance on **Mutagenesis r.u.** is an artifact of the leave-one-out cross-validation. For **1BC2**, we do not test significance because the results on Mutagenesis and Diterpene are taken from Flach and Lachiche (2004).

Dataset	**tFOIL**	**nFOIL**	**mFOIL**	**Aleph**	**1BC2**
Mutag. r.f.	79.7 ± 13.0	75.4 ± 12.3	76.6 ± 6.7	69.7 ± 11.9	$\mathbf{82.4}$
Mutag. r.u.	83.3 ± 37.7	78.6 ± 41.5	71.4 ± 45.7	$\mathbf{85.7} \pm 35.4$	76.2
Alzh. amine	$\mathbf{87.5} \pm 4.4$	86.3 ± 4.3	74.2 ± 5.9 ● ▲	70.9 ± 5.8 ● ▲	72.3
Alzh. toxic	$\mathbf{92.1} \pm 2.6$	89.2 ± 3.4	81.9 ± 2.9 ● ▲	90.9 ± 1.4	83.4
Alzh. acetyl	$\mathbf{82.8} \pm 3.8$	81.2 ± 5.2	75.2 ± 2.5 ● ▲	73.9 ± 3.4 ● ▲	73.4
Alzh. memo.	$\mathbf{80.4} \pm 5.3$ △	72.9 ± 4.3 ●	60.9 ± 4.6 ● ▲	69.2 ± 5.3 ●	68.8
NCTRER	$\mathbf{78.5} \pm 8.9$	78.0 ± 9.1	70.4 ± 15.4	52.1 ± 11.2 ● ▲	64.7
Diterpene	$\mathbf{90.9} \pm 2.1$	90.8 ± 3.1	–	85.0 ± 3.6 ● ▲	81.9

inductive logic programming (Srinivasan *et al.*, 1999). The variant used was c-ARMR (De Raedt and Ramon, 2004), which removes redundancies amongst the found patterns by focusing on so-called free patterns. c-ARMR was used to generate free frequent patterns in the data with a frequency of at least 20%. However, results obtained using this technique were on average not competitive with those of the other systems, and we decided not to include them.

To compare the different algorithms, we measure both accuracy and area under the ROC curve (see Fawcett, 2003), denoted as AUC. Accuracy is determined by a 10-fold cross-validation on all datasets except the small **Mutagenesis r.u.**, where a leave-one-out cross validation is used instead. ROC curves and AUC are determined from the cross-validation by pooling the rankings obtained on the different test folds. All algorithms were run on the same splits into training/test set for every fold. To test for significant differences in accuracy, a sampled paired t-test is applied to the results of the different folds.

3.4.3 Results

Table 3.2 shows the accuracy results for tFOIL, nFOIL, mFOIL, Aleph and 1BC2 on all datasets. There is no result for mFOIL on Diterpene as it cannot handle multi-class problems. Comparing the results for nFOIL/tFOIL and mFOIL, the experiments clearly show a gain for dynamic propositionalization over the baseline ILP algorithm, giving an affirmative answer to Question **(Q3.1)**.

The comparison between nFOIL and its tree augmented extension tFOIL shows that—as in the propositional case—relaxing the naïve Bayes assumption can yield more accurate models in some cases. tFOIL always gains some predictive accuracy, with gains ranging from very slight (0.1 percentage points) to substantial (7.5 percentage points). Although only one of these gains is significant according to a paired sampled t-test on the folds, a simple sign test on the results of tFOIL and nFOIL ($8/0$) shows that tFOIL significantly outperforms nFOIL ($p < 0.01$). This affirmatively answers Question **(Q3.3)**: performance can be improved by using a more expressive model than naïve Bayes. Furthermore, tFOIL significantly outperforms ALEPH on five datasets, and both a sign test ($7/1$) and the average accuracy seem to favor tFOIL and nFOIL over ALEPH. This shows that relatively simple dynamic propositionalization approaches are competitive with more advanced ILP systems, giving an affirmative answer to Question **(Q3.4)**. The **NCTRER** domain seems to be particularly hard for the ALEPH system. Using standard settings, ALEPH only accepts rules if they cover no negative examples. In this case, for most folds it does not return any rules on the **NCTRER** dataset, and only reaches 43.1% accuracy. The result reported above was obtained by setting the "noise" parameter to 10 (which means that a rule can cover up to 10 negative examples). The *minacc* parameter was left at its default value of 0.

To complement the study of predictive accuracy presented above, we investigate the performance of the probabilistic classifiers by means of ROC analysis. ROC curves evaluate how well the probability estimates produced by classifiers can discriminate between positive and negative examples (that is, *rank* examples) without committing to a particular decision threshold. ROC curves provide more detailed information about performance than accuracy estimates, for example with regard to possible error trade-offs under variable misclassification costs (Provost *et al.*, 1998). As ROC curves are only well-defined for binary classification problems, we do not report results for the multi-class dataset **Diterpene**. For the ILP systems, ROC curves were produced by clause voting as introduced by Davis *et al.* (2004). In clause voting, the threshold that is varied is the number of clauses that have to cover an example before it is classified as positive (the default threshold being one).

AUC results, shown in Table 3.3, generally confirm the results obtained by accuracy analysis: tFOIL outperforms nFOIL across the board, and tFOIL/nFOIL generally produce better rankings than the ILP methods and 1BC2. Note that in some cases dynamic propositionalization achieves higher AUC scores than ILP systems even though it achieves lower accuracy (for instance, compare nFOIL and ALEPH on the **Alzheimer toxic** dataset). To investigate this behavior in more detail, ROC curves for the different methods on all binary datasets are shown. Figure 3.2 shows curves on the relatively large **Alzheimer amine**, **Alzheimer toxic**, **Alzheimer acetyl** and **Alzheimer memory** datasets. Here, ROC curves clearly fall into two groups: for the ILP systems mFOIL and ALEPH, the curve has one sharp angle and is otherwise mostly linear, while the curves for tFOIL, nFOIL and 1BC2 are convex almost across the whole range of the threshold parameter. This means that for the ILP systems there is one (namely, the default) decision threshold which offers a good trade-off between true positives and false posi-

Table 3.3: Area under the ROC curve for all binary datasets. Bold numbers indicate the best result on a dataset. Results for **1BC2** on Mutagenesis are taken from Flach and Lachiche (2004). Rankings are pooled over the different folds of a 10-fold cross-validation, except for Mutagenesis r.u. where a leave-one-out cross-validation is used instead.

Dataset	**tFOIL**	**nFOIL**	**mFOIL**	**Aleph**	**1BC2**
Mutagenesis r.f.	**0.817**	0.809	0.791	0.713	0.816
Mutagenesis r.u.	0.753	0.737	0.645	**0.790**	0.729
Alzheimer amine	**0.945**	0.937	0.747	0.708	0.793
Alzheimer toxic	**0.983**	0.965	0.821	0.912	0.925
Alzheimer acetyl	**0.932**	0.916	0.759	0.752	0.815
Alzheimer memory	**0.913**	0.824	0.608	0.696	0.744
NCTRER	**0.789**	0.760	0.668	0.567	0.636

tives, but ranking below and above this point is relatively poor. At the level of induced clauses, it indicates that clauses induced by the ILP systems are specific in the sense that positive examples are typically covered by only one clause—if the decision threshold in clause voting is set to more than one, the true positive rate drops rapidly. In contrast, ranking performance for the dynamic propositionalization systems and 1BC2 is more stable, meaning that these systems also offer good classification performance under varying misclassification costs (or, equivalently, class skew). This indicates that dynamic propositionalization approaches can make use of more diverse rule sets, which are helpful in ranking examples by providing additional information but would produce too many false-positive classifications if interpreted as a disjunctive hypothesis. We will provide further evidence for this claim below.

Figure 3.3 shows ROC curves on the three smaller datasets included in our study. On **Mutagenesis r.f.** and **NCTRER**, similar observations hold as noted above, although class separation is generally poorer and curves behave less well. On the very small **Mutagenesis r.u.** dataset ranking performance is poor for all methods.

To investigate Question **(Q3.2)**, that is, to compare dynamic and static propositionalization approaches, two additional experiments were performed:

1. Learning a naïve Bayes or tree augmented naïve Bayes model over a set of clauses using a two-step approach: First, a set of rules is learned using mFOIL or Aleph, and afterwards a (tree augmented) naïve Bayes model is built using these rules. This is a static propositionalization approach, where the propositionalized data is used as input for the probabilistic learner. Question **(Q3.2)** is whether these rules are less useful when combined with (tree augmented) naïve Bayes than the rules constructed by a dynamic propositionalization approach (nFOIL/tFOIL). For the training of the TAN model we used the local score based TAN implementation in WEKA 3.4.6 (Witten and Frank, 2000), which implements the learning algorithm

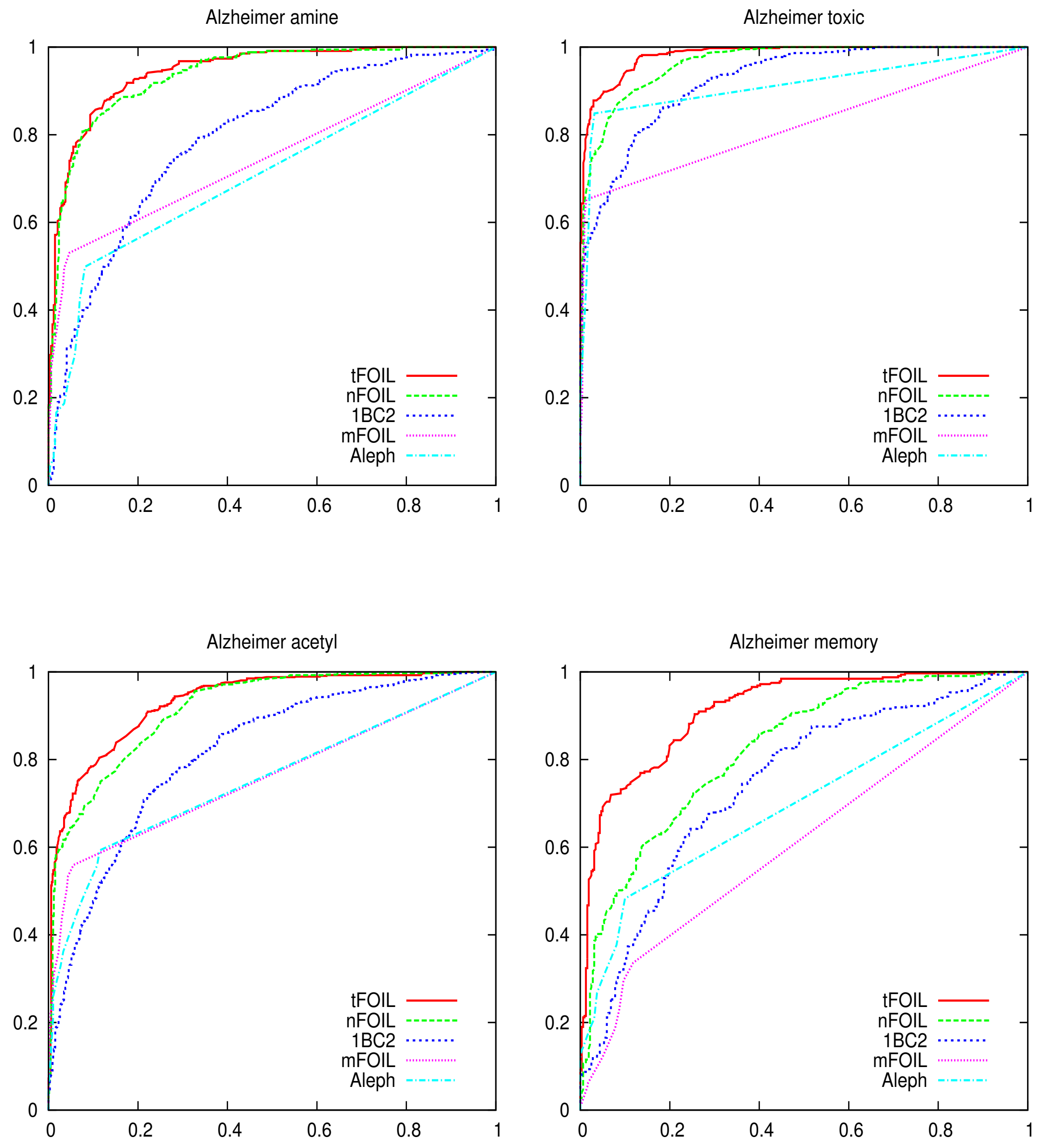


Figure 3.2: ROC curves on the Alzheimer amine, Alzheimer toxic, Alzheimer acetyl and Alzheimer memory datasets for TFOIL, NFOIL, MFOIL, ALEPH and 1BC2. Rankings are pooled over the different folds of a 10-fold cross-validation.

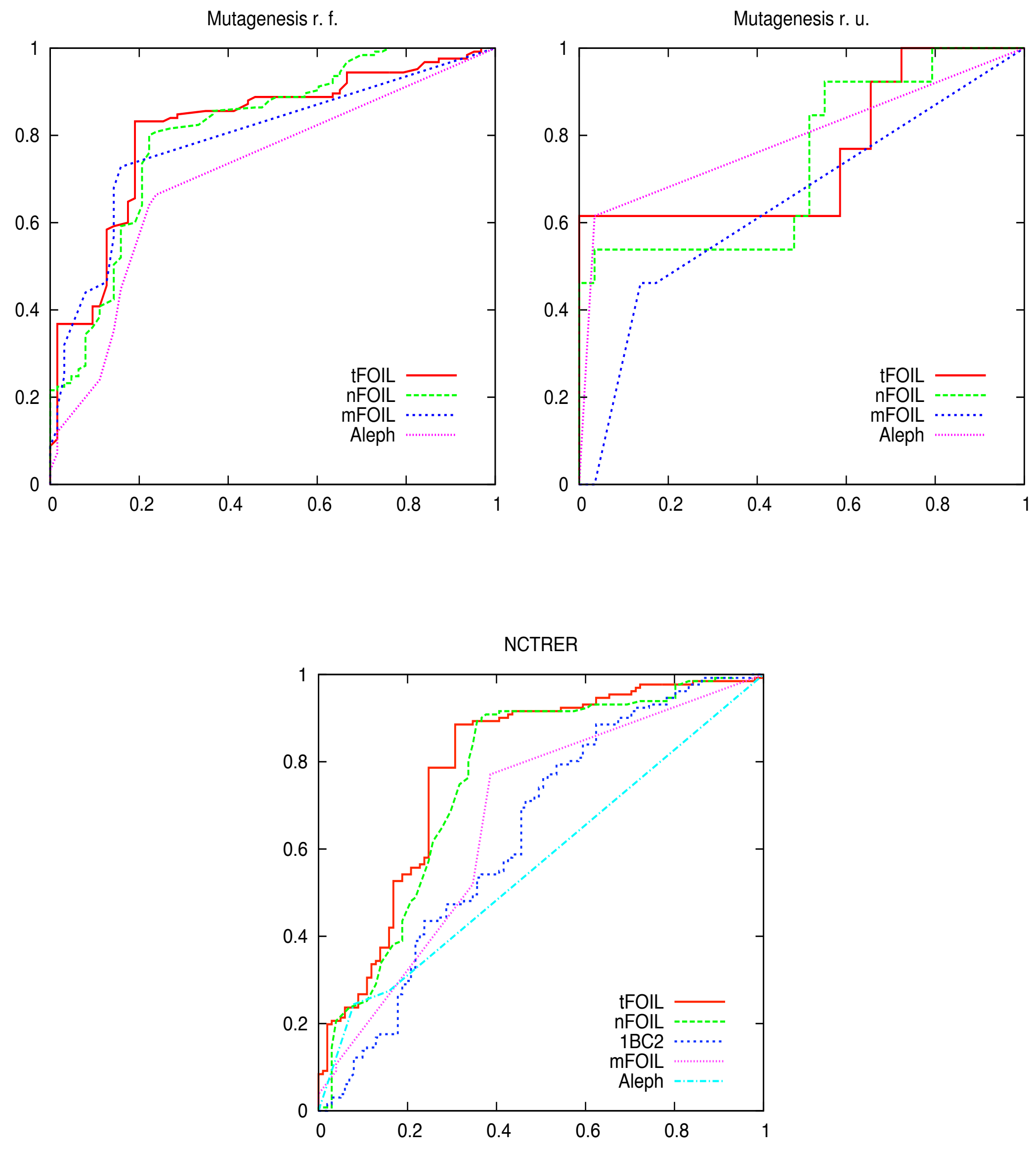


Figure 3.3: ROC curves on the Mutagenesis regression friendly, Mutagenesis regression unfriendly and NCTRER datasets for tFOIL, nFOIL, mFOIL, Aleph and 1BC2. There is no curve for 1BC2 on the Mutagenesis datasets because results are taken from Flach and Lachiche (2004). Rankings are pooled over the different folds of a 10-fold cross-validation, except for Mutagenesis r.u. where a leave-one-out cross-validation is used instead.

Table 3.4: Gain/loss in cross-validated predictive accuracy for the two-step methods mFOIL+NB/TAN, Aleph+NB/TAN and nFOIL/disjunctive over their corresponding baselines mFOIL, Aleph and nFOIL. ●/○ indicates that tFOIL's mean accuracy is significantly higher/lower, and ▲/△ that nFOIL's mean accuracy is significantly higher/lower (paired sampled t-test, $p = 0.05$).

Dataset	**mFOIL**		**Aleph**		**nFOIL/disjunctive**
	+NB	**+TAN**	**+NB**	**+TAN**	
Mutag. r.f.	±0.0 ●	±0.0 ●	±0.0	±0.0	−8.9 ● ▲
Mutag. r.u.	±0.0	+2.4	±0.0	±0.0	−47.6 ● ▲
Alzh. amine	−0.5 ● ▲	±0.0 ● ▲	±0.0 ● ▲	±0.0 ● ▲	−36.3 ● ▲
Alzh. toxic	±0.0 ● ▲	+0.2 ● ▲	±0.0	±0.0	−39.2 ● ▲
Alzh. acetyl	−0.4 ● ▲	±0.0 ● ▲	±0.0 ● ▲	+0.1 ● ▲	−31.2 ● ▲
Alzh. memory	±0.0 ● ▲	±0.0 ● ▲	±0.0 ●	±0.0 ●	−22.9 ● ▲
NCTRER	+2.1	+1.7	+4.3 ● ▲	+4.3 ● ▲	−21.5 ● ▲
Diterpene	–	–	±0.0 ● ▲	−0.3 ● ▲	–

of (Friedman and Goldszmidt, 1996).

2. Using the rules learned by nFOIL as a disjunctive hypothesis Q: For this, every rule q learned by nFOIL is evaluated on the training set. If it covers more positive than negative examples, q is added to Q, otherwise $not(q)$. The rule set Q is then evaluated as a disjunctive hypothesis on the test data. This technique can only be used for binary classification problems.

Table 3.4 shows the result of these experiments. It displays the average gain/loss of the two-step methods mFOIL+NB, Aleph+NB and nFOIL/disjunctive over their corresponding baselines mFOIL, Aleph and nFOIL. Significantly higher/lower mean accuracies of a method as compared to tFOIL/nFOIL are also indicated. On most datasets, applying naïve Bayes as a method for post-processing the learned rule set of mFOIL or Aleph does not yield any improvement, with the exception of small gains on the **NCTRER** and **Mutagenesis r.u.** datasets for mFOIL. Aleph+NB/TAN only improves on the original result of Aleph on **NCTRER** by always predicting the majority class. Furthermore, nFOIL/tFOIL significantly outperform mFOIL+NB and mFOIL+TAN on the same datasets on which they significantly outperform mFOIL. In ROC space, post-processing rules with (tree augmented) Naïve Bayes can increase or decrease the performance depending on the dataset (Table 3.5). However, AUC scores of the static propositionalization approaches are on average much lower than those of dynamic propositionalization approaches.

Using the (possibly negated) rules learned by nFOIL as a disjunctive hypothesis strongly degrades performance. This is because nFOIL can make use of rules which have a very low accuracy individually but still help as additional features in the naïve Bayes. If

Table 3.5: Gain/loss in area under ROC curve for the two-step methods mFOIL+NB/TAN and Aleph+NB/TAN over their corresponding baselines mFOIL and Aleph. Rankings are pooled over the different folds of a 10-fold cross-validation, except for Mutagenesis r.u. where a leave-one-out cross-validation is used instead.

Dataset	**mFOIL**		**Aleph**	
	+NB	**+TAN**	**+NB**	**+TAN**
Mutagenesis r.f.	$+0.045$	$+0.036$	$+0.011$	-0.034
Mutagenesis r.u.	-0.183	-0.183	-0.175	-0.063
Alzheimer amine	$+0.001$	$+0.007$	-0.013	-0.012
Alzheimer toxic	$+0.003$	$+0.011$	$+0.006$	$+0.005$
Alzheimer acetyl	$+0.002$	-0.003	-0.006	-0.010
Alzheimer memory	-0.010	-0.016	-0.010	-0.027
NCTRER	$+0.054$	$+0.048$	-0.095	-0.099

these rules are used for disjunctive classification, they produce many false-positive classifications. In fact, on some domains (e.g, **Alzheimer**) all examples in the test set are always classified as positive. This shows that nFOIL uses significantly different rule sets to ILP approaches. Thus, we can answer **(Q3.2)** as follows:

> A dynamic propositionalization approach that selects rules based on the criterion of the probabilistic model performs better than static propositionalization approaches that post-process a rule set using a probabilistic learner. This is because dynamic propositionalization can make use of different/additional rules that are not considered by traditional ILP systems as they would produce too many false-positive classifications.

It remains to answer Question **(Q3.5)**, that is, to compare dynamic propositionalization approaches to relational naïve Bayes methods such as 1BC2 in terms of accuracy and the number of features they employ. With respect to predictive accuracy and AUC score, sign tests prefer tFOIL over 1BC2 (at 7/1 and 8/0, respectively). Theory complexity is hard to compare because of the different representations. For tFOIL/nFOIL and 1BC2, one complexity measure is the number of probability values attached to clauses. There are $\#classes$ probability values attached to each clause in nFOILand 1BC2, and $2 \cdot \#classes$ in tFOIL ($\#classes$ denotes the number of classes). Additionally, we have to specify the prior distribution over the class variable. The overall number of probability values to be specified is thus of the order of $\mathcal{O}(\#clauses)$, and it is sufficient to compare the number of clauses. In the experiments, 1BC2 uses an order of magnitude more clauses than nFOIL. More precisely, 1BC2 uses more than 400 (in some cases even more than 1000) clauses whereas nFOIL is limited to using 25 clauses. This clearly shows that **(Q3.5)** can be answered affirmatively as well.

We furthermore investigated whether the proposed dynamic propositionalization approach sometimes constructs too many clauses and overfits the training data, as the stop-

Algorithm 3 Algorithm for post-pruning a hypothesis in nFOIL.

```
Initialize $H :=$ nFOIL$(E, B)$
**repeat**
    **for** all $c \in H$ **do**
        $s(c) :=$ *cross-validate-accuracy*$(H \setminus \{c\}, E, B)$
    **end for**
    $c^* := \arg\max_{c \in H} s(c)$
    $H := H \setminus \{c^*\}$
**until** cross-validated accuracy decreases
output $H$
```

Table 3.6: Cross-validated predicative accuracy results and average number of clauses in the final model for nFOIL and nFOIL/pruning. Bold numbers indicate the best result on a dataset. The are no significant differences in mean accuracy between the two methods (paired sampled t-test, $p = 0.05$).

Dataset	**nFOIL**		**nFOIL/pruning**	
	Accuracy	#clauses	Accuracy	#clauses
Mutagenesis r.f.	$\mathbf{75.4} \pm 12.3$	25.0	73.9 ± 12.1	17.7
Mutagenesis r.u.	78.6 ± 41.5	23.1	$\mathbf{85.7} \pm 35.4$	1.4
Alzheimer amine	$\mathbf{86.3} \pm 4.3$	24.7	85.0 ± 4.5	20.1
Alzheimer toxic	$\mathbf{89.2} \pm 3.4$	22.4	87.8 ± 3.6	16.6
Alzheimer acetyl	$\mathbf{81.2} \pm 5.2$	25.0	80.8 ± 4.2	20.5
Alzheimer memory	72.9 ± 4.3	24.9.5	$\mathbf{74.5} \pm 4.3$	20.4
NCTRER	78.0 ± 9.1	15.4	$\mathbf{79.3} \pm 9.7$	4.5
Diterpene	$\mathbf{90.8} \pm 3.1$	25.0	90.7 ± 3.1	23.4

ping criterion is based on the training set score. We therefore tried post-pruning a learned hypothesis. Post-pruning is more easily realized for nFOIL than for tFOIL, as the additional TAN structure in the tFOIL model prevents removal of rules which are parents of other rules. Rule post-pruning was carried out using the greedy algorithm outlined in Algorithm 3. The procedure *cross-validate-accuracy(H,E,B)* cross-validates the naïve Bayes model on the training data for a fixed set H of clauses and returns an accuracy estimate. The algorithm greedily drops clauses from H as long as this does not decrease the cross-validated accuracy estimate. Table 3.6 lists the accuracies of nFOIL and nFOIL/pruning which incorporates this rule post-pruning algorithm. There is some gain in accuracy on the small **Mutagenesis r.u.** and the **NCTRER** domain, although no differences in accuracy are significant at the $p = 0.05$ level. On these two datasets the number of features is also greatly reduced, while few or no features are pruned for the other datasets. To summarize, a clear overfitting behavior can not be observed except possibly on the very small **Mutagenesis r.u.** dataset.

We conclude that our experimental study affirmatively answers Questions **(Q3.1)**–**(Q3.5)**. The dynamic propositionalization approaches nFOIL and tFOIL yield more accurate models than simple ILP rule learning and static propositionalization approaches, and also compare favorably to the first order naïve Bayes system 1BC2 and one of the most advanced ILP systems, namely ALEPH.

3.5 Related and Future Work

The approaches that combine statistical learning with inductive logic programming techniques for addressing classification can be divided into three categories.

A first class of techniques are static propositionalization approaches. They start by generating a set of first order features and then use these features as attributes in a probabilistic model. Probabilistic models of different expressivity have been used, ranging from naïve Bayes (Pompe and Kononenko, 1995; Flach and Lachiche, 2004), to tree augmented naïve Bayes or full Bayesian networks (Davis *et al.*, 2004). The set of features is obtained either by taking all features within a pre-defined language bias, as in the 1BC system (Flach and Lachiche, 2004), or by running a traditional ILP algorithm (Pompe and Kononenko, 1995; Davis *et al.*, 2004). Furthermore, aggregation-based feature construction methods such as RELAGGS (Krogel and Wrobel, 2001) and ACORA (Perlich and Provost, 2006) that search a relational feature space using aggregation operators fall into this group. In this class of techniques the feature construction and the statistical learning steps are performed consecutively and independent of one another, whereas in nFOIL and tFOIL they are tightly integrated. However, an initial step beyond static propositionalization has been taken in the work by Pompe and Kononenko (1997), where rules generated by an ILP system are post-processed by splitting and merging clauses in order to find a rule set that satisfies the naïve Bayes assumption.

A second class of techniques employs a rich probabilistic-relational model, such as Markov Logic Networks Richardson and Domingos (2006), Bayesian Logic Programs (Kersting and De Raedt, 2001) and Probabilistic Relational Models (Getoor *et al.*, 2001). In contrast to the approaches presented here, these formalisms try to combine the full power of statistical and relational learning. While this allows for great representational power, learning these models is computationally much more challenging, in particular if the relational structure needs to be learned from data.

A third class of techniques (Popescul *et al.*, 2003; Dehaspe, 1997) indeed integrates the inductive logic programming step with the statistical learning step. Whereas the dynamic propositionalization methods presented in this chapter employ the simplest possible statistical model, namely naïve Bayes, and are focused on computational efficiency, those approaches use more advanced (and hence computationally more expensive) statistical models such as logistic regression and maximum entropy modeling, which does seem to limit the application potential. For instance, Popescul *et al.* (2003) report that—in their experiments—they had to employ a depth limit of 2 when searching for features. The work on NFOIL and TFOIL is similar in spirit to these two approaches but is much more simple and therefore computationally more efficient.

The most closely related line of research is the work on the SAYU system of Davis *et al.* (2005a), which has been developed in parallel with NFOIL. The basic SAYU system uses a "wrapper" approach where (partial) clauses generated by the refinement search of the ILP system Aleph are proposed as features to a (tree augmented) naïve Bayes, and incorporated if they improve performance. This means that feature learning and naïve Bayes are tightly coupled as in our approach. Differences between SAYU and NFOIL include that NFOIL uses a more simple greedy search algorithm, and likelihood scoring for both clause and parameter search. In SAYU, clauses are scored according to the area under the precision-recall curve, while parameters are optimized according to likelihood. Furthermore, clause selection is based on a separate tuning set. Another difference is that SAYU does not support multi-class classification, while NFOIL and TFOIL handle multi-class problems as naturally as naïve Bayes does. The SAYU system has recently been extended into several directions. One interesting direction is *view learning*, a predicate-invention mechanism that adds new views on the data defined by clauses found during the learning process (Davis *et al.*, 2005b, 2007b). Another direction are regression settings, for instance, to solve quantitative structure-activity relationship problems (Davis *et al.*, 2007a).

Finally, there is also the approach of Craven and Slattery (2001) who combine several naïve Bayes models with FOIL. The decisions of naïve Bayes models are viewed as truth values of literals occurring in clauses. This work can be regarded as the inverse of the approach presented in this chapter in that NFOIL/TFOIL employ naïve Bayes on top of logic, whereas Craven and Slattery employ naïve Bayes as a predicate in the logical definitions.

There are several directions for future work. On the one hand, the underlying ILP algorithm FOIL could be replaced by a more powerful system such as ALEPH. On the other hand, the probabilistic formalisms could be extended, for instance, by taking into account priors on the parameters or the tree augmented naïve Bayes structure. A more general direction for future work is to explore dynamic propositionalization based on other statistical learning frameworks. We will follow this direction by discussing dynamic propositionalization based on kernels in the next chapter.

Chapter 4

kFOIL: Learning Simple Relational Kernels[*]

The previous chapter has introduced a dynamic propositionalization approach based on an integration of probabilistic models (specifically, tree augmented naïve Bayes) and FOIL. In fact, most of the work in statistical relational learning has focused on extensions of various probabilistic models to relational data. In this chapter, we discuss dynamic propositionalization approaches based on an alternative paradigm, namely that of kernel methods. Kernel methods have so far received less attention in the statistical relational learning community (but see Passerini *et al.*, 2006).

Kernel methods are based on learning linear functions in a suitable reproducing kernel Hilbert space (RKHS), mostly by means of convex optimization procedures. This framework offers remarkable advantages, including the simplicity due to the uniqueness of the solution, the efficiency of well understood optimization procedures, the ability to control overfitting by means of regularization, and the flexibility of abstracting away the type of the data points via implicit feature mappings. One of the limitations of the kernel-based framework is that the kernel function needs to be carefully designed for the problem at hand. Within a statistical relational learning context, significant efforts have been devoted to devising effective kernels for structured and relational data types (Gärtner, 2003; Gärtner *et al.*, 2004; Leslie *et al.*, 2002; Lodhi *et al.*, 2002). Designing the right kernel, however, requires a sufficient understanding of the application domain so that one can figure out the relevant features for the problem at hand. While it is possible to define "universal" kernels defining arbitrarily complex hypothesis spaces (Micchelli *et al.*, 2006; Caponnetto *et al.*, 2008), and such kernels often give good results on a wide range of application problems, kernels that are customized to a particular application domain typically outperform universal kernels. Ideally, the kernel function, or the set of relevant

[*]This chapter builds on (Landwehr *et al.*, 2006b).

features, should be learned from data, rather than designed by hand. Research on kernel learning, however, has mainly focused on methods for combining existing kernel matrices with appropriate weights (Argyriou *et al.*, 2007), rather than on learning the kernel function from data.

Learning the kernel function arguably stands at a higher level of abstraction, as compared to function learning in a given RKHS. In a statistical relational learning context, the relationship between kernel learning and function learning shares similarities to the relationship between structure and parameter learning in SRL approaches based on probabilistic models. In fact, this chapter shows how relational kernel learning can be naturally performed in the setting of learning from statistical entailment presented in Section 2.4 if kernel methods are used to represent the statistical part of the model. In this setting, learning the relational model structure corresponds to inferring a suitable (relational) kernel function for the problem at hand, while parameter learning corresponds to function learning in the RKHS.

Specifically, we present the KFOIL system, which follows a similar methodology as the NFOIL system discussed in the previous chapter, but employs kernel methods instead of probabilistic models. As in the propositional case, kernel methods can produce more expressive and more accurate classifiers than simple models such as naïve Bayes. On the other hand, kernel machines are typically computationally more expensive, leading to new computational challenges in KFOIL.

This chapter is organized as follows. We start from the setting of learning from statistical entailment introduced in Section 2.4, and show how kernel-based techniques can be employed in this framework. In particular, we discuss different scoring functions that can be used with kernel machines. One of the major challenges is to find principled scoring functions that can be evaluated efficiently. Moreover, a straightforward "wrapper" approach where each candidate clause is evaluated separately will prove to be computationally infeasible; instead, incremental optimization procedures are needed that exploit the similarity of repeated optimization problems encountered during clause search. We also discuss how the KFOIL methodology can be extended to naturally deal with multi-task learning problems by learning a joint kernel function for all tasks. Finally, experimental results show that KFOIL yields competitive accuracies compared to NFOIL, ILP, and static propositionalization approaches.

4.1 Learning Relational Kernels

This section discusses how kernel machines can be incorporated into the setting of learning from statistical entailment introduced in Section 2.4. As in the previous chapter, this involves specifying the statistical coverage function $\lambda(\theta, H, B)$, and the inner and outer scoring functions S_I and S_O.

4.1.1 Statistical Learning and Kernels

Statistical learning with kernels has received widespread attention, and has proven to be one of the most competitive approaches for solving challenging real-world problems. A kernel is a positive semi-definite symmetric function[1] that generalizes the notion of inner product to arbitrary domains. An inner product defines a concept of distance between points in a space. Inner product spaces are the basis of principled and powerful classification algorithms such as the support vector machine (Cortes and Vapnik, 1995).

Kernels generalize inner products by means of a *reproducing kernel Hilbert space* (RKHS): the kernel function $K(x, x')$ between two examples $x, x' \in \mathcal{X}$ corresponds to taking an inner product in some (typically high-dimensional) feature space into which instances from X are embedded. As long as a classifier only needs access to inner products in all calculations, this feature mapping never has to be computed explicitly, but is implicit in the kernel function (so-called "kernel trick").

Different kernel-based classifiers exist. In this thesis, we will mostly be interested in *support vector machines*. A support vector machine model f is a hyperplane in the RKHS. Generally speaking, model estimation for SVMs involves solving the following generic Tikhonov regularized problem:

$$f^* = \arg\min_{f \in \mathcal{F}} C \sum_{i=1}^{N} \ell(y_i, f(x_i)) + \|f\|_K^2 \tag{4.1}$$

where $x_1, ..., x_N$ are training examples with labels $y_1, ..., y_N$ (see Section 2.1), $\mathcal{F}$ the space of hyperplanes under consideration, $\ell(y, f(x))$ is a positive function measuring the loss incurred in predicting $f(x)$ when the target is y, C is a positive regularization constant, and $\|\cdot\|_K$ is the norm in the RKHS of K. The solution of the above optimization problem can be expressed as a linear combination of kernel computations between individual training examples x_i and x using the representer theorem:

$$f(x) = \sum_{i=1}^{N} c_i K(x, x_i), \tag{4.2}$$

where $f(x)$ is proportional to the (signed) distance of x to the hyperplane represented by f, and is also called the "margin". More specifically, for classification problems the margin function f can be expressed as

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b \tag{4.3}$$

where we generalize Equation (4.2) to include a bias term b. Similarly, for regression

[1]A symmetric function $K : \mathcal{X} \times \mathcal{X} \mapsto I\!R$ is called a *positive semi-definite kernel* iff $\forall m \in I\!N, \forall x_1, \ldots, x_m \in \mathcal{X}, \forall a_1, \ldots, a_m \in I\!R$, $\sum_{i,j=1}^{m} a_i a_j K(x_i, x_j) \geq 0$.

problems one obtains

$$f(x) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) K(x, x_i) + b \tag{4.4}$$

by means of an ϵ-tube regression (see Vapnik, 1995, for details). Note that from Equation (4.3), we can obtain a simple classification decision by

$$class(x) = \begin{cases} \text{positive: } f(x) > 0 \\ \text{negative: } f(x) \leq 0 \end{cases}, \tag{4.5}$$

or a probability for $class(x)$ being positive by fitting a sigmoid function to a previously trained support vector machine:

$$P(class(x) = positive \mid x) = \sigma(af(x) + b) \tag{4.6}$$

where

$$\sigma(z) = \frac{e^z}{1 + e^z}$$

is the sigmoid function and the parameters a, b are fit to minimize cross-entropy (see Platt, 1999, for details). To avoid explicit computation of the feature map to the RKHS, the problem of Equation (4.1) is typically solved in a dual (Lagrangian) formulation. This optimization problem will be discussed in more detail in the context of scoring functions in Section 4.1.4.

4.1.2 Logical Kernel Machines

Our goal is to use kernel machines in the setting of learning from statistical entailment introduced in Section 2.4. Equation (4.2) suggests a natural way to combine kernels and statistical logical learning by means of a logical kernel function $K(\theta, \theta', H, B)$ where θ, θ' are relational examples: following the representer theorem, a margin function f for relational examples will be expressed as

$$f(\theta, H, B) = \sum_{\theta' \in E} c_{\theta'} K(\theta, \theta', H, B). \tag{4.7}$$

The only prerequisite at this point is a kernel function $K(\cdot, \cdot, H, B) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ for a relational example space $\mathcal{X}$. How such a kernel function can be defined in terms of a set of clauses H will be discussed in the next section.

Given f, a statistical coverage function λ as introduced in Section 2.4 can be obtained for (probabilistic) classification problems as given by Equation (4.5) and Equation (4.6), and regression problems can be directly modeled using Equation (4.4). For instance, in a two-class classification problem, define

$$\lambda(\theta, H, B) = \begin{pmatrix} \sigma(af(\theta, H, B) + b) \\ 1 - \sigma(af(\theta, H, B) + b) \end{pmatrix}. \tag{4.8}$$

It is also possible to tackle multi-class classification problems in this setting, either using standard techniques for multi-class SVMs (Crammer *et al.*, 2001), or by treating multi-class problems in a multi-task framework. This will be discussed in more detail in Section 4.1.5.

We will continue with a discussion of how kernel functions can be defined by logical theories in the next section. Section 4.1.4 then explores the important issue of scoring functions to complete the specification of statistical logical learning with kernels.

4.1.3 Kernel Functions Based on Definite Clause Sets

The simplest way to introduce kernels $K(\theta_1, \theta_2, H, B)$ based on a set H of definite clauses is to propositionalize the examples θ_1 and θ_2 using H and B and employ existing kernels on the resulting vectors. We will thus map each example θ onto a vector $\varphi_{H,B}(\theta)$ over $\{0,1\}^m$ with $H = \{q_1, ..., q_m\}$, with i-th component

$$\varphi_{H,B}(\theta)_i = \begin{cases} 1 : B \cup \{q_i\} \models p\theta \\ 0 : otherwise. \end{cases}$$

In other words, we have defined an m-dimensional feature map from the original relational example space to the space $\{0,1\}^m$ of binary vectors. Note that $m = |H|$ is the number of clauses in the logical theory H. Thus, the dimension of the defined feature map will eventually be learned from data.

Example 4.1.1. *Reconsider Example 2.3.1 in Chapter 2. Assume the theory H consists of the clauses in the relational feature set $\mathcal{F}$ given in Example 2.3.1, and molecules e_1, e_2 are as given in that example. For e_1 and e_2 the corresponding mapping to $\mathbb{R}^3$ is*

$$\varphi_{H,B}(e_1) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad \varphi_{H,B}(e_2) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Let us now look at the effect of defining kernels on the propositionalized representation. A simple linear kernel K_L would give the following results:

$$\begin{aligned} K_L(e_1, e_2, H, B) &= \langle \varphi_{H,B}(e_1), \varphi_{H,B}(e_2) \rangle = 1 \\ K_L(e_1, e_1, H, B) &= \langle \varphi_{H,B}(e_1), \varphi_{H,B}(e_1) \rangle = 2 \\ K_L(e_2, e_2, H, B) &= \langle \varphi_{H,B}(e_2), \varphi_{H,B}(e_2) \rangle = 2 \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product. This kernel can be interpreted as the number of clauses in H that succeed on both examples.

The linear kernel introduced in the above example can be formalized in terms of logical entailment:

$$K_L(\theta_1, \theta_2, H, B) = \#ent_{H,B}(p\theta_1 \wedge p\theta_2) \tag{4.9}$$

where, for a first-order formula r, $\#ent_{H,B}(r) = |\{q \in H | B \wedge \{q\} \models r\}|$ denotes the number of clauses in H that together with B logically entail r.

To derive Equation (4.9), note that the feature representation of an example e is a binary vector $\varphi_{H,B}(\theta)$. Let $\varphi_{H,B}(\theta)_i$ denote the i-th component of this vector. Now $\varphi_{H,B}(\theta)_i = 1$ if and only if the i-th clause in H matches the example θ, that is, $B \wedge \{q_i\} \models p\theta$ where $H = \{q_1, ..., q_m\}$. Moreover, for the linear kernel defined on the binary vectors it holds that

$$\begin{aligned}\langle \varphi_{H,B}(\theta_1), \varphi_{H,B}(\theta_2) \rangle &= |\{i \mid \varphi_{H,B}(\theta_1)_i = 1, \varphi_{H,B}(\theta_2)_i = 1\}| \\ &= |\{i \mid B \wedge \{q_i\} \models p\theta_1, B \wedge \{q_i\} \models p\theta_2\}| \\ &= |\{i \mid B \wedge \{q_i\} \models p\theta_1 \wedge p\theta_2\}| \\ &= |\{q \mid B \wedge \{q\} \models p\theta_1 \wedge p\theta_2\}|\end{aligned}$$

proving Equation (4.9). Intuitively, this implies that two examples are similar if they share many structural features. The structural features to look at when computing similarities are encoded in the hypothesis H.

If a linear kernel is used as in the example above, the corresponding feature map is m-dimensional and directly given by $\varphi_{H,B}$. Note that this feature map is explicit as the binary vectors are obtained directly by evaluating how clauses match on relational examples. However, the formalism can also be generalized to standard polynomial (K_P) and Gaussian (K_G) kernels. This is achieved by using the corresponding propositional kernels on the binary vectors resulting from the explicit feature map $\varphi_{H,B}$. For example, a polynomial kernel can be defined as

$$\begin{aligned}K_P(\theta_1, \theta_2, H, B) &= (K_L(\theta_1, \theta_2, H, B) + 1)^d \\ &= (\langle \varphi_{H,B}(\theta_1), \varphi_{H,B}(\theta_2) \rangle + 1)^d \\ &= (\#ent_{H,B}(p\theta_1 \wedge p\theta_2) + 1)^d.\end{aligned}$$

Note that for the polynomial kernel, the resulting feature map can be decomposed into two steps: relational examples are first mapped to an m-dimensional space of binary features (by $\varphi_{H,B}$), and the polynomial kernel then maps these vectors to an higher-dimensional feature space in which all products of up to d of these binary features are considered. Further note that while the first part of the feature map is explicit, the second part induced by the polynomial kernel does not have to be computed explicitly. Logically speaking, a polynomial kernel amounts to considering conjunctions of up to d clauses which logically entail the two examples, this can easily be shown by a similar derivation as for Equation (4.9).

As for the polynomial case, we can also use a propositional Gaussian kernel on top of the binary vectors resulting from $\varphi_{H,B}$. This turns out to implement the relational kernel function

$$K_G(\theta_1, \theta_2, H, B) = exp\left(-\frac{\#ent_{H,B}((p\theta_1 \vee p\theta_2) \wedge \neg(p\theta_1 \wedge p\theta_2))}{2\sigma^2}\right),$$

which can again be derived in a similar fashion as Equation (4.9) by starting from $\varphi_{H,B}$ and the propositional Gaussian kernel . Logically speaking, the argument of $ent_{H,B}$ can be interpreted as a kind of symmetric difference between the two examples, as

$$ent_{H,B}((p\theta_1 \vee p\theta_2) \wedge \neg(p\theta_1 \wedge p\theta_2))$$

counts the number of features that match on one of the examples but not on the other. As for the standard Gaussian kernel, the exponential function then computes a kernel/similarity value which is high if this difference is low. Moreover, again as for the standard Gaussian kernel, the parameter σ controls how quickly the similarity drops as the difference between examples increases.

4.1.4 Scoring Functions

We still need to specify the inner and outer scoring functions S_I and S_O. As for the nFOIL system presented in Chaper 3, the choice of inner and outer scoring function involves a certain trade-off between the optimization objective and computational convenience. On the one hand, jointly optimizing the statistical classifier λ and the hypothesis H according to one scoring function, that is, setting $S_I = S_O$, is appealing. On the other hand, the two optimization problems are rather different (discrete clause search vs. smooth numeric parameter spaces), and a scoring function that is computationally convenient for one problem might not be appropriate for the other.

We will first review scoring functions and the optimization problem in the propositional case, that is, for standard support vector machines, and derive S_I from these considerations. To simplify the exposition, a binary classification problem is assumed. Extensions of support vector machines to multi-class classification and regression are well-known (see Crammer *et al.*, 2001; Vapnik, 1995) but will not be discussed in detail here. Afterwards, different options for S_O will be discussed.

The basic principle of SVM optimization is to fit a maximum margin separating hyperplane in the RKHS, which is given by an (implicit) feature map $\phi : \mathcal{X} \rightarrow \phi(\mathcal{X})$ where $\mathcal{X}$ is the original space of examples and $\phi(\mathcal{X})$ the RKHS. The max-margin solution is obtained by minimizing

$$\frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to } \; y_i f(x_i) \geq 1, \quad i = 1, ..., N$$

where

$$f(x) = \langle \mathbf{w}, \phi(x) \rangle + b$$

($\mathbf{w}$ is the normal vector of the decision hyperplane and b its bias), and y_i are the class labels of examples x_i. However, data are not always separable, and even if they are, a classifier that tries to fit all training instances perfectly is prone to overfitting. Thus, the problem is typically relaxed by means of *slack variables* $\xi_1, ..., \xi_N$ that allow an example x_i to lie within the margin, or even on the wrong side of the decision boundary. The relaxed optimization problem is to minimize

$$C\sum_{i=1}^{N}\xi_i + \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to } y_i f(x_i) \geq 1-\xi_i,\ \xi_i \geq 0, \quad i=1,...,N$$

where C is a constant that controls the trade-off between regularization and fit of the training data. In a binary classification setting, this optimization problem can be cast into the generic Tikhonov regularization problem defined by Equation (4.1): it is equivalent to minimizing

$$C'\sum_{i=1}^{N}[1-y_i f(x_i)]_+ + \|f\|^2 \tag{4.10}$$

where $[\cdot]_+ = max(\cdot, 0)$, $\|f\| = \|w\|$ is the norm of the normal vector of the hyperplane defining f, and $[1-y_i f(x_i)]_+$ is called the *hinge loss* error function.

To avoid explicit computation of the feature map (and thus be able to work with any feature embedding implicitly defined by a kernel function), this so-called primal optimization problem is typically replace by a dual (Lagrangian) formulation of the form: maximize

$$\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i, x_j) \tag{4.11}$$

$$\text{subject to } 0 \geq \alpha_i \geq C \text{ and } \sum_{i=1}^{N}\alpha_i y_i = 0.$$

This is a convex optimization problem, which can be solved exactly and relatively efficiently using special-purpose quadratic programming techniques. Popular algorithms include chunking strategies (Kaufman, 1999) or sequential minimal optimization (Platt, 1998). The fact that model fitting can be solved efficiently and without local optima problems is considered to be one of the most important advantages of the support vector machine framework.

In order to retain these advantages of convex optimization and make use of existing optimized algorithms and software packages, we will use the loss function defined by Equation (4.10) as the (negative) inner scoring function S_I to train the support vector machine. Thus, we define for a given hypothesis H

$$S_I(E, H, \lambda, B) = -\left(C\sum_{i=1}^{N}[1-y_i f(x_i)]_+ + \|f\|^2\right) \tag{4.12}$$

where f is the linear function in the RKHS that implements the statistical coverage function λ according to Equation (4.8).

The choice of the outer scoring function S_O is less obvious. By optimizing Equation (2.8), we jointly learn a relational kernel function and solve the corresponding support vector machine problem. In a purely statistical learning setting, joint learning of kernels and function parameters has been addressed by learning combinations of simpler kernel functions, whose coefficients are learned by gradient descent (Chapelle *et al.*, 2002), semidefinite programming (Lanckriet *et al.*, 2004; Ong *et al.*, 2005) or regularization (Micchelli and Pontil, 2005). In a hybrid statistical-logical setting like the one we propose here, a constructive approach has to be pursued instead, trading optimality for expressiveness and interpretability of the learned relational rules.

Note that using the score defined by Equation (4.12) as outer scoring function S_O is not an option. It turned out in preliminary experimentation that hinge loss, while an effective scoring function for S_I, does not lead to stable search in the hypothesis space. This is because different candidate hypotheses H give rise to different kernel functions on the training examples. The hinge loss function is appropriate for determining an optimal separating hyperplane (in the max-margin sense), but it is not meant for comparing different kernel functions which lead to different feature embeddings and thus different propositional optimization problems. Moreover, the computational advantages of the hinge loss for fitting an SVM model (convexity, efficiency) do not carry over to the discrete clause search in the hypothesis space in any case. A straightforward solution is thus to employ as S_O directly the loss function we are ultimately interested in. For binary classification problems, the most common choices are 0-1 loss (accuracy) or area under the ROC curve (AUC). In our setting, 0-1 loss is defined as follows:

Definition 4.1.1 (Accuracy). *The 0-1 loss, or accuracy, of a statistical-logical model given by H and λ on a set E of examples is*

$$S_O^{Acc}(E, H, \lambda, B) = \sum_{\theta \in E} \mathbb{I}[prediction(\lambda(\theta, H, B)) = p\theta]$$

where

$$prediction(\left(\begin{array}{c} \lambda_1 \\ \lambda_2 \end{array} \right)) = \left\{ \begin{array}{l} true : \lambda_1 \geq \lambda_2 \\ false : otherwise \end{array} \right.$$

and $\mathbb{I}[f]$ is an indicator function which is 1 if f is true and 0 otherwise.

AUC can be defined in terms of ranks as follows, as it essentially counts the number of pairs of examples that are ranked correctly:

Definition 4.1.2 (Area Under the ROC Curve). *Let E denote a set of N examples, let $r_i \in \{1, ..., N\}$ denote the rank assigned to the examples in E by $\lambda(\cdot, H, B)$, and N^+ the number of positive and N^- the number of negative examples. Let furthermore r_j^+, $j = 1, ..., N^+$, denote the ranks of the positive examples, and r_j^-, $j = 1, ..., N^-$, the ranks of the negatives. Then the area under the ROC curve of a statistical-logical model given by H and λ is*

$$S_O^{AUC}(E, H, \lambda, B) = \frac{1}{N^+ N^-} \sum_{i=1}^{N^+} \sum_{j=1}^{N^-} \mathbb{I}[r_i^+ > r_j^-]$$

(note that this quantity can be computed in time $O(N \log N)$ by sorting examples according to ranks).

For regression problems, the most common scoring function is the *root mean squared error*:

Definition 4.1.3 (Root Mean Squared Error)**.** *The root mean squared error of a statistical-logical model given by H and λ is*

$$S_O^{RMSE}(E, H, \lambda, B) = \sqrt{\frac{1}{N} \sum_{\theta \in E} (\lambda(\theta, H, B) - p\theta)^2}.$$

AUC and Hinge Loss Optimization

For classification problems, choosing AUC as the outer scoring function S_O is particularly appealing. First, there are general arguments preferring AUC over classification accuracy, as AUC is less sensitive to skewed class distributions (Provost *et al.*, 1998). Second, there are theoretical arguments (discussed below) that AUC and the (negative) regularized hinge loss used as S_I are closely related. From a theoretical perspective, maximizing one joint score in both the hypothesis and function learning problem is appealing. Unfortunately, training the support vector machine to maximize AUC (rather than minimizing the regularized hinge loss of Equation (4.10)) is computationally too demanding, and therefore only approximate methods are available (Brefeld and Scheffer, 2005). However, the close relationship between hinge loss and AUC means that hinge loss minimization can be seen as an approximation to maximizing AUC also in the SVM training.

It has been repeatedly observed that standard SVMs that minimize hinge loss achieve very good AUC values, indicating that the two optimization criteria are closely related (Rakotomamonjy, 2004). More recently, this has been supported also by a theoretical analysis. Steck (2007) introduced a ranking version of the standard hinge loss function called the *hinge rank loss*. The hinge rank loss is defined as

$$L_\nu = \sum_{i=1}^{N} \left[\frac{1}{2} - y_i(r_i - \nu) \right]_+ \tag{4.13}$$

where ν is the ranking threshold (the rank from which on examples would be classified as positive). Steck (2007) has shown that the following proposition holds:

Proposition 4.1.1. *The AUC is related to the hinge rank loss and the number of false negatives as follows:*

$$AUC = 1 - \frac{L_{\bar{\nu}} - \text{const}_{D,\bar{\nu}} - N_{\bar{\nu}}^{fn}}{N^+ N^-} \quad \textit{and} \quad AUC \geq 1 - \frac{L_{\bar{\nu}} - \text{const}_{D,\bar{\nu}}}{N^+ N^-} \tag{4.14}$$

where $\bar{\nu} = \nu - \frac{1}{2}$*;* $N_{\bar{\nu}}^{fn} = \sum_{j=1}^{N^+} \mathbb{I}[r_j^+ \leq \bar{\nu}]$ *is the number of false negatives; and*

$$\text{const}_{D,\bar{\nu}} = \begin{cases} \binom{N^- - \bar{\nu} + 1}{2} : N^- \geq \bar{\nu} \\ \binom{\bar{\nu} - N^-}{2} : \textit{otherwise}; \end{cases}$$

that is, $\text{const}_{D,\bar{\nu}}$ *is a constant given the data and the ranking threshold* $\bar{\nu}$*.*

Furthermore, Steck (2007) proved that:

Proposition 4.1.2. *The lower bound in Equation* (4.14) *is tight in the asymptotic limit* $N \to \infty$ *under the assumption that* $N^+/N \to \text{const}_D^+$*, where* $0 < \text{const}_D^+ < 1$ *is a constant.*

Thus, minimizing hinge rank loss coincides with maximizing AUC in the limit. In the experimental evaluation presented in (Steck, 2007), it is shown that for most datasets, optimizing standard hinge loss instead of hinge rank loss (or AUC directly) provides near-optimal solutions in terms of AUC performance.

Note that the situation for KFOIL is very similar to the situation discussed for NFOIL in Section 3.2.2. In NFOIL, standard likelihood is used as the inner scoring function because it is closed-form computable given simple "counts" of feature and class values on the training data. However, likelihood needs to be replaced by conditional likelihood in the outer scoring function. As it is computationally too demanding to also optimize parameters according to conditional likelihood, we employed standard likelihood as an approximation to conditional likelihood in parameter learning.

Similarly, hinge loss is computationally efficient as inner scoring in KFOIL, but not effective as an outer scoring function. AUC is a natural outer scoring function but too expensive to optimize as the inner scoring function. However, with the arguments given above, optimization of the standard hinge loss as inner score can be seen as a computationally efficient approximation to maximizing AUC.

Direct Optimization of the Kernel Function

Following the definition of learning from statistical entailment (Problem 2.4.1), we have so far assumed that computing the score of a candidate hypothesis H involves estimating the statistical model λ. In KFOIL, this step involves training a support vector machine, and clearly constitutes the computational bottleneck during learning.

We now consider an alternative approach which avoids solving the SVM optimization problem while scoring a candidate hypothesis. In KFOIL, a given hypothesis H directly defines a kernel function. This function already contains valuable information concerning the potential performance of a learning machine using it, and can be employed to assess the quality of H. More specifically, *kernel target alignment* (KTA) provides a principled way of assessing the quality of a given kernel function with regard to the target concept represented by the training labels (Lanckriet *et al.*, 2004; Cristianini *et al.*, 2002). In a binary classification setting, KTA is defined as the normalized Frobenius product between the kernel matrix and the matrix representing pairwise target products:

Definition 4.1.4 (Kernel Target Alignment)**.** *Let K be an $N \times N$ kernel matrix for examples $e_1, ..., e_N$, and let $\mathbf{y} \in \{-1, 1\}^N$ denote the example labels. The kernel target alignment of K with $\mathbf{y}$ is*

$$KTA(K, \mathbf{y}) = \frac{\langle K, \mathbf{y}\mathbf{y}^T \rangle_F}{\sqrt{\langle K, K \rangle_F \langle \mathbf{y}\mathbf{y}^T, \mathbf{y}\mathbf{y}^T \rangle_F}} \tag{4.15}$$

where

$$\langle M, N \rangle_F = \sum_{ij} M_{ij} N_{ij}$$

is the Frobenius product between M and N.

The alignment of a kernel matrix is also related to bounds on the maximum performance of a classifier using this kernel (in terms of generalization error) (Cristianini *et al.*, 2002). It is thus reasonable to treat the alignment score of a kernel function as an indication of the actual performance of a support vector machine using this kernel, and use this criterion to drive the search for features. This yields the alternative outer scoring function

$$S_O^{KTA}(E, H, \lambda, B) = KTA(K, \mathbf{y}) \tag{4.16}$$

where K is the kernel matrix defined by the hypothesis H as in Equation (4.9) and $\mathbf{y} \in \{-1, 1\}^N$ is a vector representing the class labels $p\theta$ for $\theta \in E$ (note that this score is actually independent of λ). A naïve computation of the kernel target alignment as given in Equation (4.15) has complexity $O(N^2)$ where N is the number of examples. However, in Section 4.2.2 incremental approaches to computing KTA scores for all candidate clauses encountered during clause search will be discussed, which yield significant computational savings. Section 4.3 will show empirically that in this setting learning in KFOIL can be performed in linear time in the number of examples.

4.1.5 Multi-task Learning

Multi-task learning is a technique for solving multiple, related tasks in a given domain by learning a joint model for all tasks (Caruana, 1997). The rationale behind multi-task learning is the following. Given limited training data for each individual task, there are typically many (single-task) models that fit the data equally well, and the learner has to rely on its built-in bias to choose between them. By learning a joint model for all tasks, an additional bias is introduced, as the model now has to fit the observed data from all tasks simultaneously. This can significantly alleviate problems associated with sparse training data such as overfitting and unstable search. In fact, it has been shown that multi-task learning often leads to significantly improved generalization on unseen examples (Caruana, 1997).

Multi-task learning is traditionally addressed by learning a common representation of the example for different tasks. For instance, in feedforward neural networks this can be achieved by sharing the hidden layers across tasks. In KFOIL, we explicitly learn a feature representation, and thus a kernel function, by means of the induced hypothesis H. A natural approach for multi-task learning in the spirit of Caruana (1997) is therefore to share this representation (or, equivalently, the kernel function) across tasks. This can be achieved by an appropriate multi-task scoring functional, which is obtained as a combination of single-task scoring functionals on the different individual tasks. Assume that $S_O(E, H, \lambda, B)$ is an (outer) scoring functional as introduced in Section 4.1.4, and that $E_1, ..., E_T$ are the available training data for T tasks $T_1, ..., T_T$. A simple but effective multi-task scoring functional is obtained by averaging single-task scores, that is, by replacing Equation (2.8) with

$$\max_{H \in \mathcal{H}} \frac{1}{T} \sum_{t=1}^{T} S_O(E, H, \arg\max_{\lambda \in \Lambda} S_I(E_t, H, \lambda, B), E_t, B). \tag{4.17}$$

Experimental results presented in Section 4.3 show three advantages of multi-task learning in the KFOIL setting. First, they show a consistently improved generalization performance, confirming the advantages of multi-task learning observed in the literature (Caruana, 1997). Second, in our setting learning a shared clause set for multiple tasks leads to a more compact representation of the learned concept, as the multi-task clause set is significantly smaller than the union of the task-specific clause sets. In terms of the similarity/kernel function learned, this yields a generic definition of similarity that is shared between tasks and should be easier to interpret than a number of task-specific similarity functions. Third, learning a model for multiple tasks simultaneously also results in significant computational savings, as discussed in more detail in Section 4.3.8.

Finally, traditional approaches to multi-class learning with support vector machines can be considered from a multi-task perspective. Traditionally, multi-class classification in SVMs is performed by solving several binary classification tasks, either in a one-vs-one or one-vs-rest setup, using a shared representation for the examples which is fixed in advance. Test examples are then classified into the class with the highest margin. In such a setting, we propose to pursue a multi-task approach, by jointly learning the shared representation of examples which best fits all binary tasks. In this case, the average of single-task scoring functions (Eq. (4.17)) can also be replaced by multi-class accuracy. Multi-class accuracy can thus be seen as an additional multi-task scoring function in this setting. Section 4.3 shows that also for multi-class classification, multi-task approaches can improve over standard single-task (i.e., one-vs-all) approaches.

4.2 The KFOIL Algorithm

We now present KFOIL as a simple learning algorithm within the proposed framework of learning from statistical entailment with kernels. As the NFOIL algorithm presented in the previous chapter, KFOIL employs an adaptation of the well-known FOIL algorithm (Quinlan, 1990) to learn the hypothesis H and thus the kernel function. FOIL, presented in Section 2.2.3, essentially implements a straightforward *separate-and-conquer* rule learning algorithm in a relational setting, and is one of the most basic and widely used ILP algorithms

4.2.1 From FOIL to KFOIL

Search in FOIL is based on the notion of logical coverage: the goal is essentially to cover all positive and no negative examples. In the setting of learning from statistical entailment considered here, coverage is replaced by a kernel machine λ. As in the previous chapter, this implies that Algorithm 1 needs to be adapted to account for the joint statistical-logical model. First, the scoring function $score(E, H \cup \{c'\}, B)$ is replaced by the outer scoring function S_O given in Problem 2.4.1:

$$score(E, H \cup \{c'\}, B) = S_O(E, H, \arg\max_{\lambda \in \Lambda_H} S_I(E, H, \lambda, B), B). \tag{4.18}$$

For classification problems, S_O can be 0-1 loss (Definition 4.1.1), the area under the ROC curve (Definition 4.1.2), or kernel target alignment (Definition 4.1.4); for regression problems, S_O is root mean squared error (Definition (4.1.4)).

Second, KFOIL cannot use a separate-and-conquer approach. Because the final model in FOIL is the logical disjunction of the learned clauses, (positive) examples that are already covered by a learned clause can be removed from the training data (in the $update(E, H)$ function in Algorithm 1, Chapter 2). In KFOIL, this notion of coverage is lost, and the training set is not changed between iterations. Therefore, $update(E, H)$ returns E. Finally, FOIL stops when it fails to find a clause that covers additional positive examples. As an equally simple stopping criterion, learning in KFOIL is stopped when there is no improvement in score between two successive iterations.

Adapting Algorithm 1 in this way yields a "wrapper" approach in which candidate clauses are successively evaluated according to the scoring function S_O. This is similar in spirit to hypothesis search in the NFOIL system presented in the previous chapter. By replacing a generative statistical model with a consistent discriminative one, KFOIL improves over its predecessor NFOIL, as shown in Section 4.3.3, while retaining most of the interpretability inherent in selecting a small set of relevant features. However, a disadvantage of KFOIL in comparison to NFOIL is the increased computational complexity of the search procedure. If the outer scoring function is 0-1 loss or area under the ROC curve, evaluating S_O involves training a support vector machine. If the outer score is kernel target alignment, Equation (4.15) (in Definition 4.1.4) needs to be solved instead. In both cases, in a naïve implementation of the algorithm outlined so far, scoring candidate clauses will have a computational complexity at least quadratic in the number of training examples. In contrast, the naïve Bayes statistics needed for scoring in NFOIL can be updated in linear time (see Section 3.2.3).

Computational complexity can be reduced significantly if the wrapper approach is replaced by an *incremental* learning procedure. The key idea is to exploit similarities in the optimization problems that need to be solved during search. Rather than solving an optimization problem from scratch for any new candidate clause encountered, solutions or intermediate results of previous optimizations can be reused to speed up the current optimization. In fact, Section 4.3 will show empirically that for kernel target alignment with incremental optimization, learning in KFOIL can be performed in linear time in the number of examples.

4.2.2 Computational Complexity and Incremental Optimization

The computational bottleneck in KFOIL learning is the successive evaluation of candidate clauses. Assume the current theory is H with $|H| = n$. Candidate clauses are obtained as refinements of the currently best clause c (see Algorithm 1). Thus, assume the goal is to evaluate a candidate clause $c' \in \rho(c)$, and let $H' = H \cup \{c'\}$ denote the hypothesis including c'. Evaluation consists of the following three steps:

1. $\forall x \in E$, compute the feature space representation $\varphi_{H',B}(x)$ of x.
2. Compute the kernel matrix M: $\forall \theta, \theta' \in E$ compute $K(\theta, \theta', H', B)$.
3. Compute $score(E, H', B)$. Here, two cases have to be distinguished.

   3 a) S is a scoring functional that requires training a support vector machine, such as 0-1 loss or AUC. In this case, a (soft margin) SVM optimization needs to be performed. In binary classification, for instance, this amounts to solving the maximization problem given by Equation (4.11).

   3 b) S is defined by kernel target alignment. In this case, we have to compute the alignment $A(K, y)$ between the kernel $K = K(\cdot, \cdot, H', B)$ and the examples as defined by Definition 4.1.4.

In the following, we will show how significant computational savings can be obtained in all of the outlined steps by incrementally updating the relevant pieces of information.

Incrementally Computing the Feature Space Representation

In order to compute the feature space representation $\varphi_{H',B}(e)$ for every example $e \in E$, KFOIL has to first retrieve the set of examples $cov(c')$ covered by the clause c', a task that typically has to be carried out in any ILP system. This task can be solved more efficiently in an incremental way. First, we compute $cov(c)$ for the current best clause c before evaluating coverage of all refinements $c' \in \rho(c)$. As c' is a refinement of c,

$cov(c') \subseteq cov(c)$. Thus, coverage of c' only has to be checked on examples $e \in cov(c)$. Second, coverage calculations in KFOIL are sped up additionally by remembering for every free variable V in c and every example $e \in cov(c)$ the set of constant bindings

$$const(V, e) = \{a \in \mathcal{A} \mid c\theta \text{ covers } e, \theta = \{V/a\}\}$$

making c cover the example e, where $\mathcal{A}$ is the set of all (type-conform) constants that can be bound to V. Assume that c' is obtained from c by adding literal l, that is, $c' = c, l$, and $V_1, ..., V_r$ are the shared variables between l and c. Now, c' can only cover e if there is a tuple $(a_1, ..., a_r) \in const(V_1, e) \times ... \times const(V_r, e)$ such that $l\theta$ covers e for substitution $\theta = \{V_1/a_1, ..., V_r/a_r\}$. As shown empirically in Section 4.3, this can yield significant computational savings. Similar strategies have been used in other ILP systems, for instance, the original FOIL algorithm also keeps track of tuple assignments satisfying clauses (Quinlan, 1990).

A further major speedup can be obtained from the way the kernel function in KFOIL is defined in terms of a set of clauses. Note that the complex relational example set E is mapped to the much simpler (feature) space $\varphi_{H',B}(E) \subseteq \{0, 1\}^m$ in which the kernel function is computed. Two examples $e, e' \in E$ of the same class are indistinguishable in the feature space representation if $\varphi_{H,B}(e) = \varphi_{H,B}(e')$, that is, they exhibit the same structural features. Thus, a hypothesis H partitions E into clusters of examples that share the same feature space representation. For the subsequently employed kernel method, the examples in one cluster can be merged to one example with a weight corresponding to the cluster size.

We will refer to the number $\bar{m}$ of clusters as the *effective* number of examples, because this number determines the complexity of the kernel method (including the size of the kernel matrix). A detailed empirical analysis of the computational savings achievable in this way is presented in Section 4.3.8.

Incrementally Computing the Kernel Matrix

A lower-triangular representation of the kernel matrix M is kept in memory, and incrementally updated as clauses are added to the current hypothesis H. Note that the size of the kernel matrix is quadratic only in the number of *effective* examples $\bar{m}$.

Initially, $H = \emptyset$, $\bar{m} = 1$ and $K(e, e', H, B) = 0$ for all $e, e' \in E$. For scoring a candidate clause c', its contribution to the kernel function is computed and a number of cells in M need to be incremented. If the clause does not cause any cluster in E to be split, that is, it does not change $\bar{m}$, this can be done in time $O(|\varphi(cov(c))|^2)$, where $|\varphi(cov(c))|$ is the effective number of examples covered by c. If c splits a cluster in E, the dimensionality $\bar{m}$ of the kernel matrix increases, and some of the cells need to be split. Splitting an effective example i implies: adding a row of length $\bar{m} + 1$ to the lower triangular representation of the kernel matrix; copying $\bar{m} + 1$ kernel values into it, setting $M_{\bar{m}j} \leftarrow M_{ij}$ for $j \in [0, \bar{m} - 1]$ and $M_{\bar{m}\bar{m}} \leftarrow M_{ii}$; updating the number of effective examples $\bar{m} \leftarrow \bar{m} + 1$; incrementing the row entries corresponding to covered effective

Clause/Example Coverage Matrix

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
c_1	1	1	1	1	1	0	0	0
c_2	1	1	1	0	0	0	0	0
c_3	0	0	0	0	0	1	1	1

$$\underbrace{e_1\; e_2\; e_3}_{\hat{e}_1}\quad \underbrace{e_4\; e_5}_{\hat{e}_2}\quad \underbrace{e_6\; e_7\; e_8}_{\hat{e}_3}$$

$w(\hat{e}_1) = 3$

$w(\hat{e}_2) = 2$

$w(\hat{e}_3) = 3$

Kernel Matrix (Effective Examples)

	$\hat{e}_1$	$\hat{e}_2$	$\hat{e}_3$
$\hat{e}_1$	2	1	0
$\hat{e}_2$		1	0
$\hat{e}_3$			1

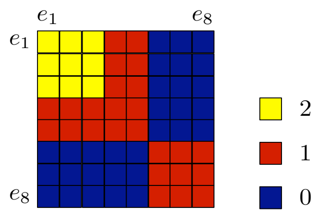


Figure 4.1: Effective examples and kernel matrix for three clauses.

Clause/Example Coverage Matrix

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
c_1	1	1	1	1	1	0	0	0
c_2	1	1	1	0	0	0	0	0
c_3	0	0	0	0	0	1	1	1
c_4	0	0	0	1	1	1	1	0

$$\underbrace{e_1\; e_2\; e_3}_{\hat{e}_1}\quad \underbrace{e_4\; e_5}_{\hat{e}_2}\quad \underbrace{e_6\; e_7}_{\hat{e}_3}\quad \underbrace{e_8}_{\hat{e}_4}$$

$w(\hat{e}_1) = 3$

$w(\hat{e}_2) = 2$

$w(\hat{e}_3) = 2$

$w(\hat{e}_4) = 1$

Kernel Matrix (Effective Examples)

	$\hat{e}_1$	$\hat{e}_2$	$\hat{e}_3$	$\hat{e}_4$
$\hat{e}_1$	2	1	0	0
$\hat{e}_2$		2	1	0
$\hat{e}_3$			2	1
$\hat{e}_4$				1

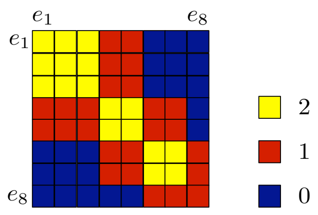


Figure 4.2: Effective examples and kernel matrix for four clauses.

examples by one. A single split can thus be done in $O(\bar{m})$ time, and a full update in time $O(|\varphi(cov(c))|\bar{m})$, where $\bar{m}$ is the number of effective examples after all splits. Note that this number will typically be much smaller than the overall number of examples, and Section 4.3.8 empirically shows that it indeed grows with its square root in a real world domain.

Example 4.2.1. *Consider a dataset containing examples $e_1, ..., e_8$, and a current hypothesis consisting of the three clauses c_1, c_2, c_3 as visualized in Figure 4.1. Assume further* *that the coverage of clauses is as given in the "Clause/Example Coverage Matrix" in Figure 4.1. For instance, example e_1 is covered by clauses c_1 and c_2. This yields three effective examples, or clusters, $\hat{e}_1, \hat{e}_2, \hat{e}_3$, with weights $w(\hat{e}_1) = 3, w(\hat{e}_2) = 2, w(\hat{e}_3) = 3$. The resulting kernel matrix is a 3×3 lower-diagonal matrix, also shown in Figure 4.1. At the level of original examples $e_1, ..., e_8$, this yields a relatively coarse kernel function, as kernel values are identical for all examples falling into one cluster.*

Figure 4.2 visualizes the situation after adding a (candidate) clause c_4. Note that the cluster represented by $\hat{e}_3$ in Figure 4.1 has been split into two new clusters $\hat{e}_3$ and $\hat{e}_4$. Accordingly, the resulting kernel matrix has one more row and one more column. At the level of the original examples $e_1, ..., e_8$, the kernel function has been refined and now represents a more detailed structure of similarity.

Incremental SVM optimization

If S is a scoring functional that requires training a support vector machine during clause evaluation, efficiently solving the optimization problem given by Equation (4.11) is crucial. Well-known algorithms for this problem include sequential minimal optimization (Platt, 1998) or other chunking techniques (Joachims, 1999), and more recently introduced online optimization approaches such as LaSVM (Bordes *et al.*, 2005) or stochastic gradient descent (Shalev-Shwartz *et al.*, 2007). Optimization in the KFOIL implementation is carried out using the SVMlight package, which is based on the technique described by Joachims (1999) (see Section 4.3). Assume we have already solved the optimization problem for the currently optimal clause c, and are evaluating a refinement $c' \in \rho(c)$. That is, we have maximized Equation (4.11) for the feature space representation resulting from the hypothesis $H \cup \{c\}$. Let α^0 denote an optimal solution. Typically, the optimization problem resulting from the hypothesis $H \cup \{c'\}$ will be very similar: in the feature space representation of the data, only one of the attributes has been changed, and only on a subset of the examples (as only one feature has been refined). One can thus expect that also the optimal solution α^* for the new problem will be close to the old solution α^0. A straightforward but effective approach to incremental optimization is thus to restart the optimization at the old maximum α^0, and continue optimizing until the new optimality criterion (Karush-Kuhn-Tucker conditions) is met. Intuitively, this corresponds to starting from the old separating hyperplane and slightly adapting it until it defines the max-margin solution of the new optimization problem. Section 4.3 will show significant benefits of this technique compared to re-starting the optimization procedure from scratch.

Incrementally Computing Kernel Target Alignment

Finally, also the kernel target alignment score (Definition 4.1.4) can be computed incrementally. The key observation is that as a clause c is added to H, it produces an additional, incremental contribution to the kernel matrix M. This contribution can be propagated to the three Frobenius norms from which KTA is computed (see Equation (4.15) in Definition 4.1.4). For all examples covered by the candidate clause c, their contribution to the previously computed norms should be first removed, and then replaced with the contribution due to their updated kernel values.

4.3 Experimental Evaluation

This section presents an experimental evaluation of the proposed KFOIL algorithm in several domains. The goal of the experimental study is two-fold. In a first part, KFOIL is compared to several related systems that employ inductive logic programming and propositionalization approaches. In a second part, we present a detailed analysis of the performance of KFOIL in different learning settings and with different scoring functions. More specifically, we compare multi-task and single-task learning, explore the relative merits of different scoring functions with regard to accuracy of inferred models and computational cost, and present experiments in a multi-class domain using different approaches to multi-class classification. Finally, the computational complexity and scaling behavior of KFOIL will be investigated, and the clause sets returned by the algorithm will be inspected.

4.3.1 Experimental Domains and Datasets

Table 4.1 gives an overview of the different datasets used in the experimental evaluation. As in the previous chapter, we mostly focus on *structure-activity relationship* (or *SAR*) problems, which are of central importance in many areas of bio- and chemoinformatics (see Section 3.4.1). Large quantities of experimental SAR data have been generated, which are stored in centralized, easily accessible public databases such as PubChem[2] (Wheeler *et al.*, 2008). Note that SAR problems are not only an interesting application for relational learning in general, but also a natural application area for multi-task learning: often substances have been tested for several related properties, and thus come with several class labels. Jointly building a model for all properties can yield increased predictive accuracy, as will be shown below.

The individual datasets will now be described in more detail, and their use in the study will be motivated. The **Mutagenesis** datasets are concerned with predicting the mutagenicity of small molecules based on their chemical structure (Srinivasan *et al.*, 1996). For **Alzheimer**, the aim is to compare analogues of Tacrine, a drug against Alzheimer's disease, according to four desirable properties: inhibit **amine** re-uptake, low **toxicity**, high **acetyl** cholinesterase inhibition, and good **reversal** of scopolamine-induced memory deficiency (King *et al.*, 1995). The **NCTRER** dataset contains structural information (atoms and bonds) for a diverse set of natural, synthetic and environmental estrogens, and classifications with regard to their binding activity for the estrogen receptor (Fang *et al.*, 2001). For more details on Mutagenesis, Alzheimer, and NCTRER, see Section 3.4.1.

In the **Biodegradability** domain the task is to predict the biodegradability of chemical compounds based on their molecular structure and global molecular measurements(Blockeel

[2]`http://pubchem.ncbi.nlm.nih.gov/sources/`

Table 4.1: Overview of all datasets used in experiments, including the number of classes, number of available examples, accuracy of majority class predictor, number of relations that are used in rules, and the number of facts in the Prolog database.

Dataset	**#Classes**	**#Examples**	**Maj. Class**	**#Rel.**	**#Facts**
Mutagenesis r.f.	2	188	66.5%	4	10324
Mutagenesis r.u.	2	42	69.1%	4	2109
Alzheimer amine	2	686	50.0%	20	3754
Alzheimer toxic	2	886	50.0%	20	3754
Alzheimer acetyl	2	1326	50.0%	20	3754
Alzheimer memory	2	642	50.0%	20	3754
NCTRER	2	232	56.5%	3	9283
BioDeg – classification	2	328	56.4%	36	27236
BioDeg – regression	n.a.	328	n.a.	36	27236
NCGC BJ (AID 421)	2	1285	96.2%	29	89929
NCGC Jurkat (AID 426)	2	1242	89.1%	29	89929
NCGC Hek293 (AID 427)	2	1250	94.1%	29	89929
NCGC HepG2 (AID 433)	2	1282	96.1%	29	89929
NCGC MRC5 (AID 434)	2	1289	96.2%	29	89929
NCGC SK-N-SH (AID 435)	2	1281	93.3%	29	89929
MTDP E.coli (AID 365)	2	206	51.2%	23	23734
MTDP Human (AID 366)	2	206	79.5%	23	23734
MTDP HIV-2 (AID 367)	2	206	73.7%	23	23734
NCI BT_549	2	2778	50.4%	30	283612
NCI HCC_2998	2	3177	56.8%	30	283612
NCI HS_578T	2	2870	54.0%	30	283612
NCI SR	2	3006	62.2%	30	283612
NCI T_47D	2	2909	53.3%	30	283612
WebKB	6	1089	51.2%	6	86392

et al., 2004). The original (numeric) target variable is the half-life for aerobic aquenous biodegradation of the particular chemical compound. Alternatively, the problem can be treated as a classification task by thresholding this target variable. The Mutagenesis, Alzheimer, NCTRER and Biodegradability domains were included in the study because they are well-known benchmark datasets for ILP and more generally relational learning methods. Furthermore, Alzheimer can be cast as a multi-task domain, with tasks corresponding to the four properties of interest.

The following additional SAR problem domains were chosen because they are natural test-cases for multi-task learning. The **NCGC** datasets contain results of high throughput screening assays to determine in vitro cytotoxicity of small molecules. Experiments have been performed multiple times with cell lines derived from different tissue types: BJ cell line (human foreskin fibroblasts), Jurkat cell line (human T cell leukemia), Hek293 cell line (human embryonic kidney cells), HepG2 cell line (hepatocellular carcinoma), MRC5 cell line (human lung fibroblasts) and SK-N-SH cell line (human neuroblastoma). Test results for different cell lines will typically be different but related, thus it is natural to treat them as different prediction tasks in a multi-task learning setting. The **MTDP** datasets contain results of enzymatic assays for inhibition of *ribonuclease H* activity. Individual datasets represent assay results for ribonuclease H enzymes from different organisms: E. coli ribonuclease H, human ribonuclease H1, and HIV-2 ribonuclease H. Again, these results can be treated as different but related prediction tasks. Both the NCGC and MTDP datasets have been extracted from the PubChem database. Compound descriptions and class labels, as well as more details about the experimental protocol, are available from this database by looking up the bioassay ID (denoted *AID XXX* in Table 4.1). Finally, the **NCI** datasets provide screening results for the ability of compounds to suppress or inhibit the growth of tumor cells (Swamidass *et al.*, 2005). Screening results are available for 60 different cell lines; however, not all compounds have been tested against all cell lines. We selected five cell lines (BT_549, HCC_2998, HS_578T, SR, T_47D) that result in relatively small datasets but together contain test results for almost all compounds used in the study. As for the NCGC domain, test results for different cell lines correspond to different prediction tasks. The NCI datasets are also significantly larger than the other domains considered (both in terms of number of examples and number of facts, that is, size of the logical database), and thus serve as a good benchmark to test computational efficiency and scaling behavior of κFOIL. Note that the number of facts in the logical database mainly affect the complexity of the logical part of learning (the effort of matching clauses on examples), and, as in standard ILP, is only limited by the performance of the underlying logic programming system. In contrast, the number of individual examples will mainly affect the complexity of the statistical part of learning.

WebKB is a multi-class domain which has a somewhat different background and will be discussed in more detail in Section 4.3.6.

4.3.2 κFOIL Implementation and Experimental Setup

The κFOIL algorithm, with computational improvements as discussed in Section 4.2.2 and support for multi-task learning as described in Section 4.1.5 has been implemented based on YAP Prolog[3] and the support vector machine package SVMlight[4].

A polynomial kernel of degree $d = 2$ is used in all experiments, and a beam size of 5 is used unless noted otherwise. A model in κFOIL is refined as long as the training score improves (even by a small margin), see Section 4.2.1. To avoid overfitting, we perform model selection to choose the regularization parameter C by (repeatedly) splitting the available training data into a training and a validation set. Evaluation of algorithms is performed by cross-validation or more generally multiple splits into train/test data. As

[3]`http://www.dcc.fc.up.pt/~vsc/Yap/`

[4]`http://svmlight.joachims.org/`

KFOIL can produce a ranking (via example margins in the SVM model), area under the ROC curve (AUC) can be computed on the test data. For binary classification problems, performance is mostly evaluated by AUC, except when directly comparing against accuracy results from the literature. The motivation for using AUC instead of accuracy is that some of the datasets used in the study are very unbalanced, and there are in general arguments for preferring AUC over accuracy as an evaluation measure (Provost *et al.*, 1998). For regression problems, performance is evaluated by root mean squared error (RMSE).

The following sections present a comprehensive evaluation of the proposed KFOIL algorithm, addressing different aspects such as effectiveness, efficiency, and interpretability of the learned model. Specifically, we set up experiments to answer the following questions:

(Q4.1) Does KFOIL improve over its predecessor NFOIL employing naïve Bayes?

(Q4.2) Does multi-task learning provide advantages over single-task learning in the KFOIL setting?

(Q4.3) Is the approach of solving multi-class problems in a multi-task formulation (see Section 4.1.4) effective and efficient?

(Q4.4) Are the first-order kernel functions learned by KFOIL interpretable for human experts?

(Q4.5) Does learning in KFOIL scale up to large datasets when using kernel target alignment scoring and incremental learning algorithms?

4.3.3 Comparison to ILP and Propositionalization Approaches

We begin by comparing KFOIL to other relational learning approaches. More specifically, we considered the following systems. The NFOIL system, presented in Chapter 3, combines naïve Bayes and FOIL in a similar spirit as KFOIL combines kernels and FOIL. ALEPH is a state-of-the-art ILP system developed by Ashwin Srinivasan.[5] It is based on the concept of *bottom clauses*, which are maximally specific clauses covering a certain example. Furthermore, we consider a static propositionalization baseline. More specifically, a variant of the relational frequent query miner WARMR (Dehaspe *et al.*, 1998) was used for static propositionalization as WARMR patterns have shown to be effective propositionalization techniques on similar benchmarks in inductive logic programming (Srinivasan *et al.*, 1999). The variant used was c-ARMR (De Raedt and Ramon, 2004), which removes redundancies amongst the found patterns by focusing on so-called free patterns. c-ARMR was used to generate all free frequent patterns in the data sets where the frequency threshold was set to 20%. We used at most 5000 of the generated patterns as features to generate (binary) propositional representations of the datasets.

[5] `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph_toc.html`.

Table 4.2: Average predictive accuracy on Mutagenesis, Alzheimer and NCTRER for kFOIL with accuracy scoring, nFOIL, Aleph and static propositionalization. A polynomial kernel of degree $d = 2$ was used for kFOIL in all experiments. On Mutagenesis r.u. a leave-one-out cross-validation was used (which, combined with the small size of the dataset, explains the high variance of the results), on all other datasets a 10 fold cross-validation. • indicates that the result for kFOIL is significantly better than for other method (paired two-sided t-test, p = 0.05).

Dataset	**kFOIL**	**nFOIL**	**Aleph**	**c-ARMR**
Muta. r.f.	77.0 ± 14.5	75.4 ± 12.3	73.4 ± 11.8	73.9 ± 11.2
Muta. r.u.	85.7 ± 35.4	78.6 ± 41.5	85.7 ± 35.4	76.2 ± 43.1
Alz. amine	89.8 ± 5.7	86.3 ± 4.3	70.2 ± 7.3•	81.2 ± 4.5•
Alz. toxic	90.0 ± 3.85	89.2 ± 3.4	90.9 ± 3.5	71.6 ± 1.9•
Alz. acetyl	90.6 ± 3.4	81.2 ± 5.2•	73.5 ± 4.3•	72.4 ± 3.6•
Alz. mem.	80.5 ± 6.2	72.9 ± 4.3•	69.3 ± 3.9•	68.7 ± 3.0•
NCTRER	78.5 ± 9.3	78.0 ± 9.1	50.9 ± 5.9•	65.1 ± 13.2•

On the propositionalized datasets, a cross-validation of a support vector machine was then performed[6]. Table 4.2 presents cross-validated accuracy results for kFOIL, nFOIL, Aleph and c-ARMR. To facilitate comparison to previously published results, performance is evaluated according to classification accuracy, and kFOIL is run with accuracy scoring. Results clearly show that kFOIL outperforms Aleph and static propositionalization. Moreover, it consistently improves upon its predecessor nFOIL, indicating that in a hybrid statistical-logical setting kernel methods can improve over a simple naïve Bayes model. This affirmatively answers Question **(Q4.1)**.

Table 4.3 compares kFOIL to the relational tree induction algorithms Tilde and S-CART on the Biodegradability classification and regression datasets. kFOIL has been run with accuracy (classification problem) and RMSE (regression problem) scoring. As in the experiments reported by Blockeel *et al.* (2004), results are averaged over five runs of a ten-fold cross-validation, and the same splits into training and test sets have been used. kFOIL clearly outperforms the other two approaches in the regression setting, while yielding comparable accuracy in the classification setting.

4.3.4 Evaluation of Different Scoring Functions

As discussed in Section 4.1.4, different scoring functions can be considered for classification problems. From a computational perspective, the main difference is between 0-1 loss or AUC on the one hand and kernel target alignment on the other hand. The former scores require to solve the SVM optimization problem (Equation (4.11)), while kernel

[6]Note that this methodology puts this approach at a slight advantage and might yield over-optimistic results.

Table 4.3: Average cross-validated accuracy (*Classification*) and root mean squared error (*Regression*) on the Biodegradability dataset. The results for Tilde and S-CART have been taken from Blockeel *et al.* (2004). 5 runs of 10 fold cross-validation have been performed, on the same splits into training and test set as used by Blockeel *et al.* (2004). For classification, average accuracy is reported, for regression, root mean squared error. ● indicates that the result for KFOIL is significantly better than for other method (unpaired two-sided t-test, p = 0.05).

Dataset	**KFOIL**	**TILDE**	**S-CART**
Classification			
BioDeg GR	73.4 ± 1.63	73.6 ± 1.1	72.6 ± 1.1
BioDeg GP1P2R	73.5 ± 0.95	72.9 ± 1.1	71.3 ± 2.3
Regression			
BioDeg GR	1.139 ± 0.036	1.265 ± 0.033●	1.290 ± 0.038●
BioDeg GP1P2R	1.182 ± 0.038	1.335 ± 0.036●	1.301 ± 0.049●

target alignment can be directly computed given the kernel matrix (see Definition 4.1.4), which is typically much more efficient.

Table 4.4 compares KTA scoring to accuracy scoring on the first nine datasets listed in Table 4.1. Results show that if the standard beam size of 5 is used, KTA is substantially less accurate. There are two possible explanations for this result: either kernel target alignment is generally not a good scoring function in our context, or it does not work well together with the greedy search strategy employed in KFOIL. In the latter case, the clause set truly maximizing KTA would result in a good joint model, but only a (strongly) suboptimal clause set is found during search. To test this hypothesis, the system was run with increased beam sizes (10, 20, and 30). This improves relative performance, indicating that the weak results for beam size 5 are at least partly due to local search issues. For the rest of the experimental study, we thus use a beam size of 20 rather than the standard 5 for KFOIL with KTA scoring. Even with the increased beam size KTA scoring is typically more efficient than accuracy or AUC scoring (detailed results are presented in Section 4.3.8).

4.3.5 Single-task vs. Multi-task Learning

For the four multi-task domains (Alzheimer, NCGC, MTDP and NCI), the benefits of simultaneously learning a joint model for all tasks were explored. In these domains, there are between three and six different target labels available, although not all labels are necessarily known for all examples (cf. Table 4.1). We compare a multi-task (MT) approach against a single-task (ST) approach using the following methodology. All examples available in a given domain are first split into a training set (80%) and test set (20%). In the MT approach, a joint model is built from the training set using the technique described in

Table 4.4: Average cross-validated prediction accuracy of KFOIL with accuracy scoring and standard beam size 5 (Acc[5]) and KFOIL with KTA scoring and beam sizes 5, 10, 20 and 30 (KTA[5], KTA[10], KTA[20], KTA[30]).

Dataset	**Acc[5]**	**KTA[5]**	**KTA[10]**	**KTA[20]**	**KTA[30]**
Mutag. r.f.	77.0 ± 13.7	74.9 ± 12.1	74.4 ± 12.0	77.7 ± 11.7	78.8 ± 6.6
Mutag. r.u.	85.7 ± 35.0	83.3 ± 37.3	83.3 ± 37.3	85.7 ± 35.0	85.7 ± 35.0
Alzh. amine	89.8 ± 5.4	72.7 ± 6.3	75.5 ± 5.8	82.5 ± 6.1	83.1 ± 5.2
Alzh. toxic	90.0 ± 3.7	80.9 ± 4.3	88.9 ± 4.1	92.3 ± 1.6	93.7 ± 1.1
Alzh. acetyl	90.6 ± 3.2	74.7 ± 3.6	75.0 ± 3.0	81.7 ± 4.3	83.2 ± 3.5
Alzh. memory	80.5 ± 5.9	66.8 ± 8.0	63.9 ± 7.6	75.1 ± 4.2	77.4 ± 3.9
NCTRER	78.5 ± 8.8	78.5 ± 8.8	78.5 ± 8.8	78.0 ± 9.3	77.6 ± 8.9
BioDeg GR	73.4 ± 1.6	63.5 ± 1.7	64.8 ± 0.8	65.1 ± 1.9	69.1 ± 1.6
BioDeg GP1P2R	73.5 ± 0.9	68.1 ± 2.1	68.5 ± 1.1	68.6 ± 2.1	68.6 ± 2.0

Section 4.1.5. In the ST approach, one model is built for each task using those examples for which label information for that task is available. For every example in the test set, both approaches return a predicted class label for every task. This prediction is compared to the true label for that task if it is known, and resulting area under the ROC curve (AUC) is computed. To obtain reliable estimates, results are averaged over 50 randomly chosen splits into training and test set.

Table 4.5 shows average AUC results on Alzheimer, NCGC, MTDP and NCI for KFOIL in the single-task and multi-task setting, using KTA and AUC scoring. For the NCI datasets, only 25% of the available training data was used to infer a model to reduce the total computational cost of experiments. Results indicate that multi-task learning provides small but consistent improvements in test set AUC over single-task learning. This is particularly evident on the NCGC, MTDP and NCI datasets, while the result for the Alzheimer datasets is less clear. A likely explanation is that the different tasks in Alzheimer—predicting a compounds amine re-uptake, toxicity, acetyl cholinesterase inhibition, and reversal of memory deficiency—are not as strongly related as for the other domains. Moreover, results confirm the earlier observation that evaluating clause sets by the performance of the corresponding support vector machine (in this case, by its AUC) yields slightly better results than evaluating them by kernel target alignment.

We furthermore investigate how the gains from multi-task learning depend on the amount of training data available. Figure 4.3 shows average AUC of KFOIL in the single-task and multi-task learning setting for different numbers of training examples on NCI, averaged over the five available tasks. The system has been run with KTA scoring and beam size 5 to achieve maximum computational efficiency for these larger datasets. To obtain stable AUC estimates, the following methodology was used. A 5-fold cross-validation is performed. However, for each fold, 10 models are built on bootstrapped samples of

Table 4.5: Average AUC ± standard deviation on Alzheimer, NCGC, MTDP and NCI for KFOIL with KTA and AUC scoring, using a single-task (ST) or multi-task (MT) learning setting. Results are averaged over 50 random 80%/20% train/test splits of the data. For NCI, models are learned from only 25% of the available training data in every split. Bold font indicates whether single-task or multi-task learning yielded better results.

Dataset	**KFOIL**			
	KTA Scoring		AUC Scoring	
	ST	MT	ST	MT
Alzheimer amine	**91.1**± 2.4	90.1±2.8	93.9± 3.6	**95.6**±2.2
Alzheimer toxic	98.1± 0.8	**98.3**±0.7	95.9± 3.4	**96.6**±2.0
Alzheimer acetyl	89.8± 2.2	**90.0**±2.6	92.1± 4.6	**93.8**±1.9
Alzheimer memory	82.5± 5.0	**86.3**±2.7	**86.5**± 5.8	85.1±4.8
NCGC BJ	68.0± 9.3	**69.9**±9.9	**72.4**± 10.0	72.2±8.7
NCGC Jurkat	64.7± 5.3	**66.2**±5.6	69.9± 6.0	**71.7**±5.6
NCGC Hek293	67.2± 7.6	**68.7**±7.2	71.0± 7.7	**72.0**±6.8
NCGC HepG2	71.5± 8.7	**74.6**±8.8	74.2± 8.8	**77.6**±8.2
NCGC MRC5	65.3± 10.0	**68.8**±9.4	67.0± 9.6	**69.8**±9.3
NCGC SK-N-SH	63.6± 7.1	**66.0**±7.8	66.4± 7.4	**67.2**±7.0
MTDP E.coli	61.5± 9.3	**62.6**±8.0	61.8± 8.0	**64.5**±7.0
MTDP Human	56.8± 9.7	**61.2**±10.5	62.5± 11.7	**66.7**±10.7
MTDP HIV-2	58.8± 8.8	**61.7**±8.8	61.0± 10.1	**63.5**±10.1
NCI BT_549 25%	69.7± 2.3	**71.1**±2.2	70.5± 2.6	**71.8**±2.5
NCI HCC_2998 25%	65.3± 2.5	**65.8**±2.1	64.1± 2.5	**67.5**±2.6
NCI HS_578T 25%	70.5± 2.2	**71.1**±2.2	70.3± 2.5	**71.9**±2.6
NCI SR 25%	70.3± 3.1	**71.2**±2.4	69.9± 2.9	**71.3**±2.4
NCI T_47D 25%	71.1± 2.3	**72.2**±2.1	70.9± 3.0	**72.5**±2.5

the fold's training set, and their results on the fold's test set are averaged. As in standard cross-validation, this yields one datapoint (AUC result) per fold. Figure 4.3 also reports one-standard-deviation error bars, and for every fraction of training data the significance of the difference between single- and multi-task learning according to a paired two-sided t-test on the fold results. Results again indicate an advantage of multi-task (MT) over single-task (ST) learning. More specifically, MT learning from 50% of the available training data reaches about the same accuracy as ST learning from the whole dataset. The difference between MT and ST learning is significant or borderline significant for small training set sizes, but becomes less pronounced if more training data is available. Overall, this indicates a positive answer to Question **(Q4.2)**.

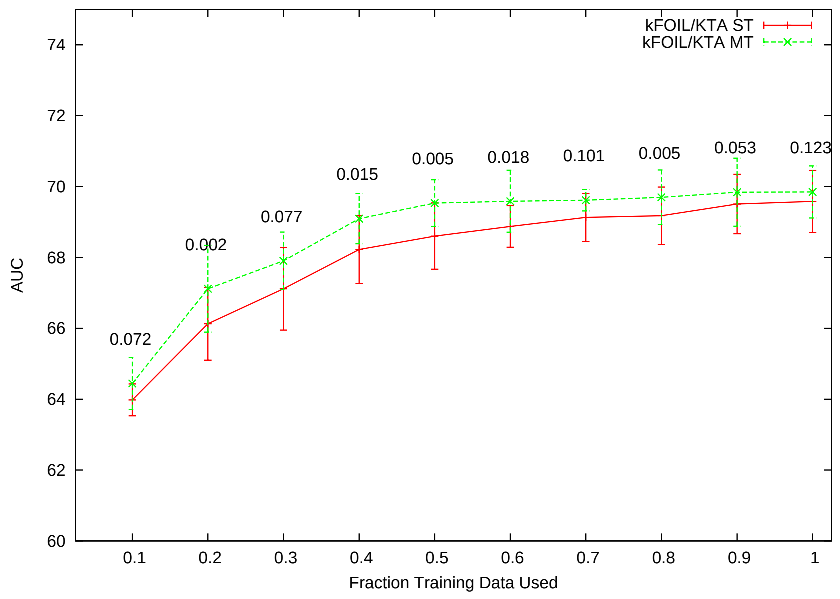


Figure 4.3: Learning curve for KFOIL with KTA scoring and beam size 5 in a single-task and multi-task learning setting on the NCI datasets (test set AUC as a function of training set size). Results are averaged over a 5-fold cross-validation, and over individual tasks. For each fold, 10 models are built on bootstrapped samples of the fold's training set and their results on the test set averaged (see text).

4.3.6 Multi-class Problems

As a relational multi-class problem, we consider the "University Computer Science Department", or **WebKB**, dataset. This dataset consists of web pages collected from four computer science departments: Cornell University, University of Texas, University of Washington and University of Wisconsin. Web pages are classified into six categories: `course`, `department`, `faculty`, `research project`, `staff` and `student`. Pages not belonging to any of these classes are assigned to a default `other` class. Table 4.6 reports class statistics for each of the four Universities. We relied on a relational version of the dataset which includes hyperlink and anchor word information (Slattery and Craven, 1998). Table 4.7 reports the predicates we employed in our experimental evaluation[7]. In order to decrease class skew and to obtain simpler and more readable

[7]All predicates where taken from Slattery and Craven (1998) apart for `dirs_after_tilde_in_url/2` which was our addition.

Table 4.6: Class statistics for the WebKB dataset.

Class	Cornell	Texas	Washington	Wisconsin
`course`	44	38	77	85
`department`	1	1	1	1
`faculty`	34	46	31	42
`research project`	20	18	21	25
`staff`	21	3	10	12
`student`	128	148	126	156
`other`	619	573	940	946

Table 4.7: Description of the relational predicates employed for the "University Computer Science Department" dataset.

Predicate	Description
`has_word(page,word)`	`word` is contained in the `page` text
`has_anchor_word(anchor,word)`	`word` is contained in the `anchor` text
`all_words_capitalized(anchor)`	all words in the `anchor` text are capitalized
`has_alphanumeric_word(anchor)`	the `anchor` text contains an alphanumeric word
`linktopage(page1,page2,anchor)`	`anchor` identifies a link from `page1` to `page2`
`dirs_after_tilde_in_url(page,num)`	The URL of `page` contains a tilde followed by `num` directories

models, we removed the `other` class in all experiments. In a preprocessing phase, we furthermore selected for each training set the first 50 words and 50 anchor words with highest information gain, and restricted clauses to only contain these informative words.

As outlined in Section 4.1.5, multi-class problems can be cast in the multi-task setting by identifying classes with tasks. We compare a single-task approach (that is, a standard one-vs-rest setup) against multi-task approaches using average single-task scores or multi-class accuracy. Table 4.8 reports experimental results for these three settings and different scoring functions, in a leave-one-university-out cross-validation (that is, web pages of one university are classified using a model trained on the remaining three universities). The evaluation measure on the test set is multi-class accuracy. Results confirm the advantage of multi-task learning with respect to single-task, as multi-task scoring functions achieve better results in all cases. It is interesting to note that while multi-task KTA achieves the

Table 4.8: Leave-one-university out results for the ”University Computer Science Department” dataset. Evaluation measure is multi-class accuracy, while scoring measures are single- and multi-task accuracy, AUC and KTA, and multi-class accuracy.

Test University	Accuracy		AUC		KTA		Multi-Class
	ST	MT	ST	MT	ST	MT	Accuracy
Cornell	69.8	75.4	77.4	77.4	31.9	75.8	72.6
Texas	74.8	75.2	80.7	81.9	34.6	83.1	81.1
Washington	75.2	74.4	68.8	74.4	75.2	70.3	70.7
Wisconsin	78.2	82.9	78.8	84.1	76.3	80.4	78.8
Micro Average	74.7	77.3	76.5	79.7	56.2	77.5	75.9

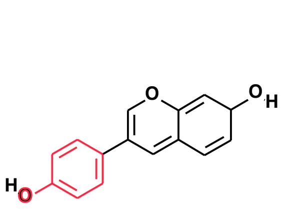
overall best results, its single-task counterpart performs very badly for the two Universities with the least number of examples, Cornell and Texas. This performance degradation is due to a very low recall on the `student` class, which is mostly predicted as `course` and `faculty` in the Cornell and Texas cases. Note also that directly using multi-class accuracy as a scoring function does not improve over average accuracy.

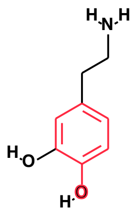
To answer Question **(Q4.3)**, we note that solving the WebKB multi-class problem in a multi-task formulation indeed yields an effective classification method. Note furthermore that the multi-task formulation will typically be more efficient than training independent models in a one-vs-rest setup, as only one clause set is learned and thus significant computational cost of computing clause coverage is saved (confer the analysis of multi-task efficiency in Section 4.3.8).

4.3.7 Interpreting Learned Models

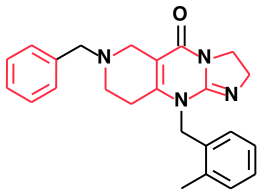
A major advantage of the proposed methodology for learning relational kernels compared to other relational kernel-based approaches (such as pre-defined kernels for structured data, cf. Gärtner, 2003) is that it retains some of the interpretability of its underlying inductive logic programming approach. After training, KFOIL returns a relatively small set of first-order logical clauses that define a similarity measure between examples in the given domain. These clauses are typically easy to read by human experts, especially as they can build on human-supplied background knowledge such as known functional groups for chemical compounds. Figure 4.4 shows example clauses learned in the NCTRER, NCI and WebKB domains, and visualizes how these clauses match on examples.

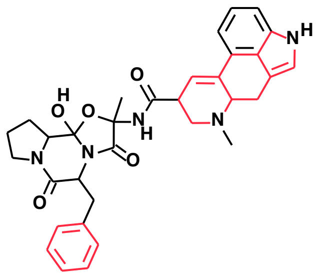
The key part in understanding a final learned model is to understand the similarity function defined by the kernel. The kernel function $k(e_1, e_2)$ is defined by the number of clauses that match both e_1 and e_2, or some non-linear transformation thereof (cf. Section 4.1.3). That is, the kernel counts how many structural features are shared by the two examples. Examples that share a large number of features will be considered similar, and are thus likely to receive the same classification. If clauses are understandable to the hu-



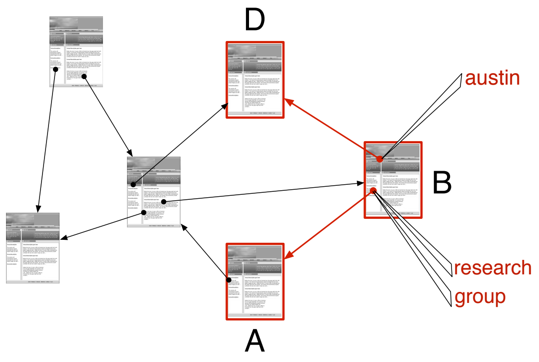


$$\leftarrow atm(B,o), bd\_atm(B,C,c,-), bd\_atm(C,D,c,=), bd\_atm(C,E,c,-),$$
$$bd\_atm(E,F,c,=), bd\_atm(G,D,c,-), bd\_atm(F,H,I,-)$$



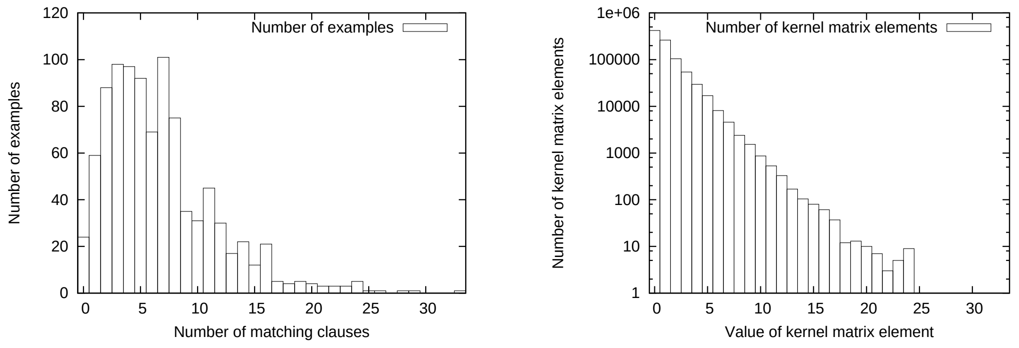


$$\leftarrow three\_linear\_rings(A,B), benzene(A,C,D)$$



$$\leftarrow linktopage(B,A,C), anchor\_word(C,research), anchor\_word(C,group),$$
$$linktopage(B,D,E), anchor\_word(E,austin))$$

Figure 4.4: Examples for bodies of clauses learned by κFOIL on the NCTRER (upper), NCI (middle) and WebKB (lower) data sets. Additionally, examples on which the clauses match are shown, with the sub-structure defined by the clauses highlighted in red. Note that for NCTRER, only low-level atom/bond structure is given, and the algorithm automatically infers that aromatic rings with a phenol group are relevant for the classification problem at hand. For NCI a library of high-level chemical structures was supplied as background knowledge, such that small clauses can encode relatively complex substructures.



(a) Number of clauses matching example (b) Kernel matrix value

Figure 4.5: Histogram representation of the average number of clauses matching an example, and the average number of features shared between a pair of examples, that is, value of the corresponding kernel matrix element. Note the logarithmic scale in the second plot. Statistics are collected from NCI-BT_549 (25%), building a model on all of the training data, AUC scoring.

man experts, the resulting similarity function will be understandable as well as long as the number of features shared between two examples is not overwhelmingly large. Figure 4.5 shows a histogram representation of the average number of clauses matching an example, and the average number of features shared by a pair of examples. It can be observed that for most pairs of examples the number of shared features is very small, thus it will be easy to manually inspect their similarity. Figure 4.6 visualizes a learned kernel function on the WebKB dataset, for the single-task (left) and multi-task (right) case respectively. In the single-task case, `student` is the positive class, as it leads to the most balanced task. Lighter colors correspond to larger kernel values. Examples are grouped according to classes, and classes ordered so to maximize similarity between neighboring ones. Within each class, examples are sorted according to their principal component, in order to cluster together similar examples. In the single-task case, it can be observed that positive examples (bottom left) are roughly grouped together into block-like clusters by the structural features they exhibit. Negative examples have lower kernel values in general, meaning that the predictor tends to rather model the positive class, a common behavior when negative examples come from many possible sources. Anyhow, a slight tendency for negative examples to cluster together on a per-class basis can still be recognized. In

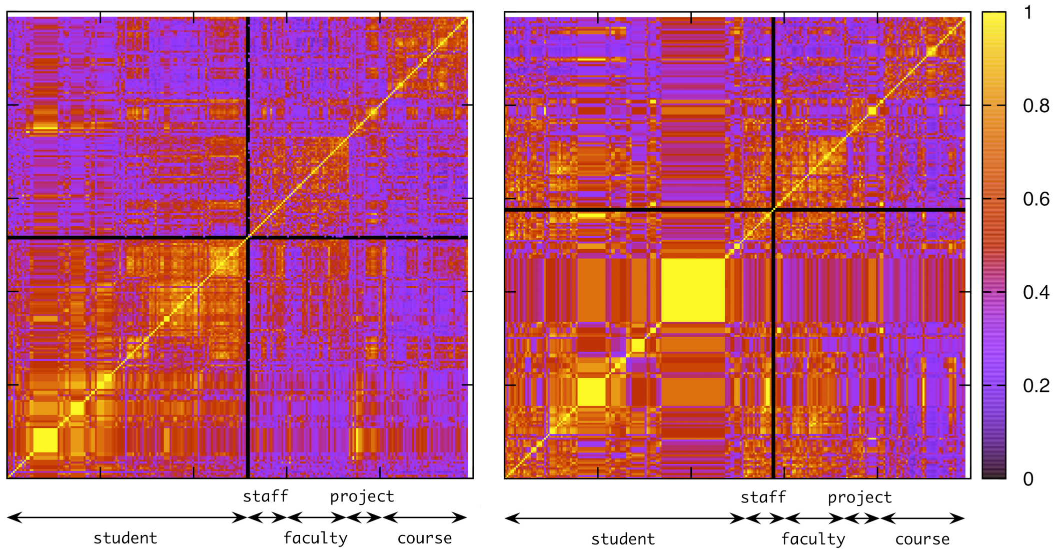


Figure 4.6: Visualization of learned kernel functions on the WebKB dataset, in a single-task (left) and multi-task (right) learning setting. Kernel values are shown for web pages at Cornell University for a model trained on the remaining three universities (Washington, Wisconsin, and Texas). In the single-task case, `student` is the positive class as it leads to the most balanced task. Examples are grouped by class, classes are sorted so to maximize similarity between pairwise ones. Examples within each class are sorted by their principal component, in order to cluster together similar ones.

the multi-task case, on the other hand, two main differences can be observed: first, the matrix exhibits a coarser grain structure, with larger clusters on average, as smaller clause sets are learned in the multi-task setting. Second, even if the `student` class is still much more represented, being by far the majority one (see Table 4.6), other classes are modeled as well, and their clusters are more evident than in the single-task case. However, note that there is a tendency for `staff` and `faculty` classes to share a common representation, and to share features with some of the `student` examples. This is quite intuitive considered that all such classes represent personal homepages.

Table 4.9 shows the number of clauses obtained on the Alzheimer, EPA, MTDP, NCI, and WebKB datasets for single-task and multi-task learning. Results show that multi-task learning yields a more compact representation than single-task learning: the set of clauses obtained is significantly smaller than the union of the task-specific clause sets. This indicates that it is possible to infer a clause set (and thus, similarity measure) that generalizes over the individual tasks. Interpreting such a generalized representation will typically be easier than looking at all task-specific clause sets individually.

To summarize, the results show that 1) KFOIL learns a relatively small number of

Table 4.9: Number of clauses obtained for single-task and multi-task learning on the Alzheimer, NCGC, MTDP, NCI, and WebKB datasets (AUC scoring). The clause set for single-task learning is the union of the clause set obtained on the individual tasks (that is, duplicate clauses have been removed). For Alzheimer, NCGC, MTDP, and NCI, results are averaged over a 50 train-test splits as in Table 4.5. For WebKB, results are averaged over a leave-one-university-out cross-validation as in Table 4.8.

Dataset	Number of Clauses	
	ST	MT
Alzheimer	79.5	34.7
NCGC	195.7	76.0
MTDP	84.0	59.7
NCI	414.5	175.3
WebKB	127.5	37.8

first-order clauses and 2) relatively few clauses are typically shared between any pair of examples. Moreover, multi-task learning provides more compact models than single-task learning. This indicates an affirmative answer to questions **(Q4.2)** and **(Q4.4)**.

4.3.8 Computational Complexity

Computational complexity in KFOIL is dominated by the evaluation of candidate clauses within the greedy top-down refinement search (cf. Algorithm 1). More specifically, for every candidate clause under consideration it has to be determined (1) which examples are covered by the clause and (2) how adding the clause to the current model affects the score. Task (1) consists of running a Prolog query against the current database that holds the description of the examples and the background knowledge. This is a standard task that has to be carried out in ILP systems, and thus constitutes a "computational baseline" in the sense that it is the minimum effort any system has to perform. Task (2) is the additional effort required to score the hybrid statistical-logical model defined by KFOIL, and thus constitutes the computational "overhead" compared to a purely logical approach. Note that the complexity of the second task will strongly depend on the particular scoring function used.

Figure 4.7 shows the scaling behavior of KFOIL with AUC scoring (beam size 5) and KTA scoring (beam sizes 5 and 20). Complexity is broken down into time spent on Task (1) and Task (2). Results clearly show that KTA scoring scales better than AUC scoring, even taking into account the larger beam size. For KTA scoring coverage calculations clearly dominate overall runtime, and computing kernel target alignment for a candidate clause only constitutes a small overhead. In contrast, for AUC scoring the actual score update strongly dominates the total computational effort. Overall, KFOIL with KTA scoring scales roughly linear in the number of examples, while AUC scoring exhibits

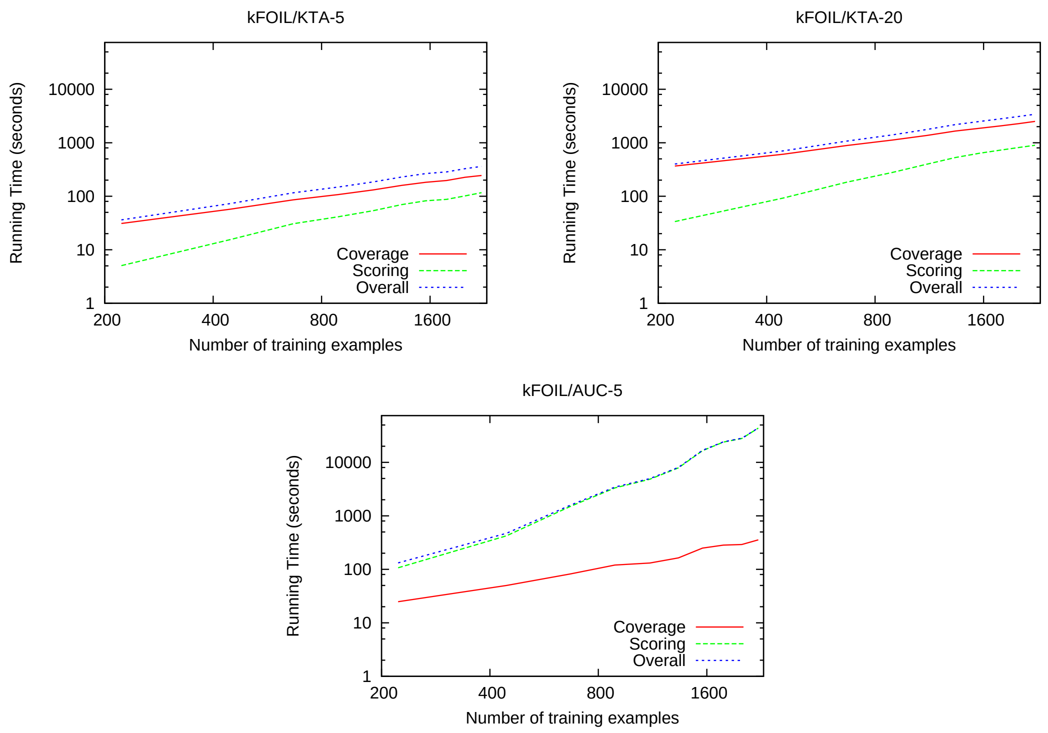


Figure 4.7: Runtime in seconds for building a single model on NCI-BT_549 as a function of the number of training instances for different scoring functions and beam sizes. Graphs show (1) time spent on determining the coverage of a clause ("Coverage"), (2) time spent to determine the score of the combined model given the coverage ("Scoring"), and (3) overall time spent in the evaluation of the clause ("Overall"). Results are averaged over 50 random samples for each dataset size.

clearly non-linear scaling. Note that scoring by classification accuracy has essentially the same complexity as scoring by AUC, as in both cases the full SVM model has to be built.

Additional computational savings are obtained in a multi-task setting: coverage computations only have to be carried out once (as only one clause set is learned). For KTA scoring, where coverage calculations dominate computational cost, multi-task learning thus yields significant computational savings compared to building an individual model for every task.

The linear scaling behavior of κFOIL with KTA scoring is surprising, as even the incremental algorithm for computing the alignment of a clause set involves operations which are non-linear in the number of examples. However, note that the relevant factor is the number of *effective* examples, that is, that are mapped to different propositional vectors by the clause set (see Section 4.2.2). Figure 4.8, left plot, shows the number of

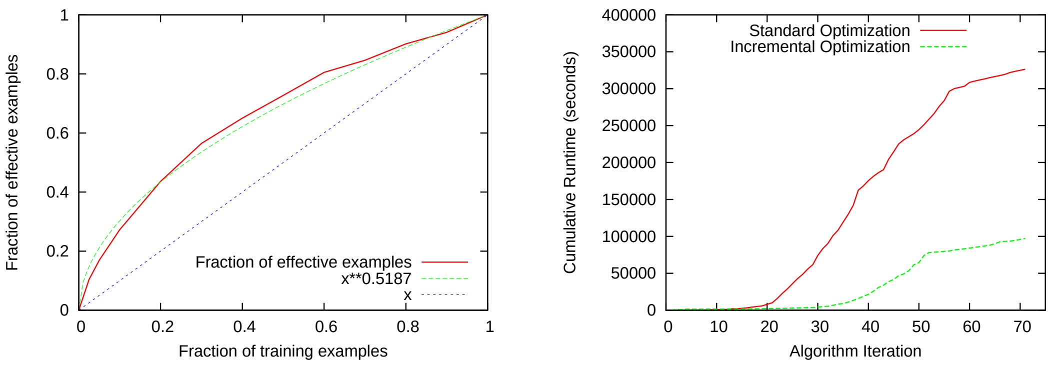


Figure 4.8: Left plot: number of effective examples as a function of the overall number of examples for KFOIL with KTA scoring and beam size 5. Both axis are normalized to the interval $[0, 1]$, that is, values are divided by their corresponding maximum. A least-squares curve fit of the function $f(x) = x^a$ yields a constant $a \approx 0.5187$. Results are averaged over all tasks in NCI and 50 random splits into training (80%) and test (20%) data. Right plot: Cumulative runtime of KFOIL training as a function of the algorithm iteration for AUC scoring, with and without incremental optimization of the support vector machine.

effective examples as a function of the overall number of examples. A least-squares curve fit of the function $f(x) = x^a$ to the (normalized) curve yields $a \approx 0.5187$. Thus, the number of effective examples grows approximately with the square root of the total number of examples, explaining the overall linear scaling behavior observed in Figure 4.7. These results indicate that KFOIL using kernel target alignment scoring and incremental optimization scales well to reasonably large datasets, giving an affirmative answer to question **(Q4.5)**.

4.3.9 Summary of Experimental Results

KFOIL has been shown to compare favorably against well-known ILP systems and static propositionalization approaches.

With respect to the comparison between AUC and accuracy scoring on the one hand and alignment scoring on the other hand, experimental results indicate a classical accuracy-efficiency trade-off. Alignment scoring yields slightly lower accuracy, but offers a much better scaling behavior, making the proposed algorithm practical for large-scale relational learning problems. According to our experiments, linear scaling can be expected—a surprising result. The explanation is that the number of *effective* examples that are fed to the support vector machine only grows with the square root of the original number of examples.

Multi-task learning has been shown to offer three key benefits compared to single-task learning. First, our experiments confirm the observation that multi-task learning can offer advantages in terms of generalization performance (Caruana, 1997). Second, the resulting representation of the learned model in terms of a clause set is more compact, as the joint clause set is smaller than the union over the task-specific clause sets induced. Finally, together with alignment scoring multi-task learning offers additional computational benefits, as coverage calculations are shared between tasks.

4.4 Related and Future Work

KFOIL is directly related to several other statistical relational learning systems. Moreover, the proposed approach also touches upon work in other areas, most notably kernel learning and multi-task learning.

4.4.1 Statistical Relational Learning based on Kernels

The use of kernel functions for relational learning is not new; several approaches have been proposed in the literature. A popular method are static propositionalization approaches as discussed in Section 3.1: a set of features is first generated using an ILP system (as in Muggleton *et al.*, 2005) or a pattern miner (as in Kramer and De Raedt, 2001), and an SVM model is built on this feature set. The approach is particularly appealing for feature sets generated from pattern mining, as SVMs can typically cope with very large feature sets easily. However, as outlined above, it has the disadvantage that the feature set is not optimized with respect to the particular SVM model employed. Moreover, the interpretability of large feature sets (as returned by a pattern miner) is very limited.

In contrast, we propose a *dynamic propositionalization* approach, in which the feature set defining the kernel and the statistical classifier are optimized jointly. The approach is related to other dynamic propositionalization systems such as SAYU (Davis *et al.*, 2005a, 2007b), the NFOIL system presented in Chapter 3, and Structural Logistic Regression (Popescul *et al.*, 2003), but relies on kernel methods instead of probabilistic models. This enables us to tackle different learning tasks such as classification or regression in a uniform framework. Also, the resulting KFOIL system has been shown to improve upon NFOIL in terms of predictive accuracy in our experimental study. As an alternative direction for solving regression problems, Davis *et al.* (2007a) propose to combine the SAYU system with a (multi-instance) linear regression model. Comparing these two approaches experimentally is an interesting direction for future work.

Trading interpretability and efficiency for effectiveness, the RUMBLE margin-based rule learner uses a large set of features encoded by mathematical expressions, frequent substructures, definite clauses and their combinations, adding them one at a time, and keeps the subset achieving the best results, removing features with low weight (Rückert and Kramer, 2007). On the other hand, KFOIL aims at keeping a simple and parsimonious approach, that can be easily applied to other ILP systems.

Finally, dynamic approaches to propositionalization have also been addressed in the graph mining community, where the goal is to learn a set of subgraphs that propositionalize graph data. In this context, Saigo *et al.* (2009) consider a boosting approach to collect a set of informative patterns, which can be seen as a dynamic propositionalization approach combining graph patterns and kernel machines.

Another way of employing kernels on relational data is to pre-define a kernel function without any additional relational learning phase. For instance, many kernel functions for graph data are based on the idea of (approximately) counting the number of co-occuring substructures between two graphs (Gärtner, 2003). From our perspective, this can be seen as implicitly constructing a very large (even infinite) feature space, where one feature represents a substructure that could be shared between two graphs. Counting co-occuring structures then means applying a standard propositional kernel in this space. A number of other logical and relational kernels have been developed recently. Cumby and Roth (2003) described a family of relational kernels defined using a simple description logic. Gärtner *et al.* (2004) defined kernels over complex individuals using higher order logic. Passerini *et al.* (2006) introduced the notion of *visiting* predicates that are used to explore relational objects given background knowledge, and they defined kernel between objects as similarities between proof trees obtained for the visitor predicates. Declarative kernels rely on topological and parthood relations to define object similarity in terms of parts and connections between parts (Frasconi *et al.*, 2005). Recently, Wachman and Khardon (2007) proposed a kernel on relational data by upgrading walk-based graph kernels to hypergraphs. Complementary research has focused on defining distances in a relational setting, see (Ramon and Bruynooghe, 1998; Kirsten *et al.*, 2001) or (Ramon, 2002) for an extensive treatment.

While retaining much of the expressivity granted by logic, these approaches need to pre-define an appropriate kernel function or distance measure for the domain at hand. In contrast, the kernel we introduced here defines similarity in terms of a small set of clauses that are learned from data by leveraging ILP-style search techniques.

4.4.2 Kernel Learning

The technique we propose can also be seen as a kernel learning approach, in that it constructs the kernel based on the available data. Joint learning of kernels and function parameters has been addressed in a purely statistical setting by learning combinations of simpler kernel functions, whose coefficients are learned by gradient descent (Chapelle *et al.*, 2002), semidefinite programming (Lanckriet *et al.*, 2004; Ong *et al.*, 2005) or regularization (Micchelli and Pontil, 2005). Ong *et al.* (2005) propose a general framework where kernels are chosen from a hyper-RKHS induced by hyperkernels based on an appropriately regularized scoring functional. These approaches are typically more principled than our approach (as they learn the kernel by solving well-posed optimization problems). However, the formulation by which the kernel is obtained as a convex combination of other kernel functions would be difficult or impossible to apply in the context of dynamic feature construction in a fully-fledged relational setting. Furthermore, to the best of the authors' knowledge, KFOIL is the first system that can learn kernels defined by small sets of interpretable first-order rules.

4.4.3 Multi-task Learning

Multi-task learning is traditionally addressed by learning a common representation of an example for different tasks. For instance, in feedforward neural networks this can be achieved by sharing the hidden layers across tasks. Multi-task learning with kernel machines was first introduced by Evgeniou *et al.* (2005) by including in the regularization term a matrix which encodes the relation between tasks. This work showed how to convert the resulting problem into an equivalent single task problem via an appropriate multi-task kernel function. However, this method requires to explicitly encode task relationships, which have to be known in advance. Multi-task kernel learning was later introduced using a generalization of single-task 1-norm regularization (Obozinski *et al.*, 2006; Argyriou *et al.*, 2007), which minimizes the number of non-zero features across tasks, or relying on maximum entropy discrimination (Jebara, 2004).

The setting considered in this chapter is different, as we are learning a hybrid statistical-logical model. We propose to exploit multi-task information for learning the relational model structure, by constructing a relational kernel function that is shared across tasks. As shown in Section 4.3, this leads not only to better generalization but also yields a more compact representation of the learned concept. To the best of the authors' knowledge, multi-task kernel learning has never been addressed in a statistical relational learning context before. Indeed, multi-task structure learning itself has received little attention in the statistical relational learning setting, a notable exception being a recent work by Deshpande *et al.* (2007) on learning multi-task probabilistic relational rule sets.

Multi-task learning has also received attention in the ILP community, in the form of learning several related concepts simultaneously. Different approaches have been pursued. Related to our approach is the work by Reid (2004), where the assessment of candidate clauses on the primary task is augmented with the performance of similar rules on a secondary task. Furthermore, a scenario resembling multi-task learning has been studied by Datta and Kibler (1993), where (sub)structures of concepts already learned are used as building blocks when learning a new concept. A further related scenario is that of *repeat learning* and *multiple predicate learning* (Khan *et al.*, 1998; De Raedt *et al.*, 1993), where an ILP learner has to discover a series of related concepts drawn from some (initially unknown) distribution. Moreover, *predictive clustering trees* have been used in an ILP setting. These trees can be used in a multi-task setting, where predictions for several tasks are made at every leaf (Blockeel *et al.*, 1998, 2004).

4.4.4 Future Work

kFOIL can be extended in a number of directions. Greedy search, for instance, produces a small but suboptimal set of features, and more complex strategies such as bottom-up search (Muggleton and Feng, 1990) could be employed. Such search techniques can sometimes avoid the problems with local optima and premature convergence that are associated with greedy search strategies.

Concerning multi-task learning, while kFOIL currently learns a feature space representation which is common across tasks, it is possible to extend it with task-specific components accounting for the specificity of each task, as is done with hierarchical Bayesian models (see, for example, Deshpande *et al.*, 2007).

With respect to the multi-class learning strategy proposed in Section 4.1.5, it would be interesting to verify if the advantage we observed addressing multi-class classification as a multi-task problem is confirmed with different multi-task learning algorithms.

Conclusions Part I

Part I of the thesis has introduced dynamic propositionalization techniques based on probabilistic models and kernels. In contrast to static propositionalization, relational and statistical learning in dynamic propositionalization are tightly coupled. This results in more accurate models using a smaller number of interpretable first-order rules.

The nFOIL system presented in Chapter 3 combines the simplest approaches from statistical and relational learning, namely naïve Bayes and FOIL. This yields a simple but very efficient SRL systems with little computational overhead compared to its relational baseline algorithm FOIL. Nevertheless, nFOIL was shown to be competitive in terms of accuracy with static propositionalization and ILP systems on a number of benchmark datasets. We also discussed extending the simple naïve Bayes model used in nFOIL to tree augmented naïve Bayes (TAN), and introduced an incremental algorithm for learning the TAN structure.

As an alternative framework, Chapter 4 has discussed dynamic propositionalization approaches based on kernels. Advantages of kernels methods in propositional learning include state-of-the-art accuracy in many domains, the efficiency and effectiveness of convex optimization, and the ability to handle classification and regression problems in a uniform framework. One of the challenges of using kernel methods—especially with complex and structured data—is that an appropriate kernel function needs to be predefined by the user. Using kernel methods in dynamic propositionalization naturally leads to a framework for learning a relational kernel function from data. The resulting kFOIL system was shown to be more accurate than its predecessor nFOIL, and can handle both classification and regression problems. On the negative side, the use of kernels in dynamic propositionalization poses new computational challenges. To reduce computational complexity, we proposed to directly score the kernel defined by a clause set using kernel target alignment. Moreover, incremental computation of the kernel matrix, max-margin separating hyperplane, and kernel target alignment can significantly reduce computational complexity. Finally, we investigated multi-task learning in the kFOIL framework. An extensive experimental evaluation has proven the advantage of the multi-task learning approach over its single-task counterpart, both in efficiency and effectiveness.

The dynamic propositionalization approaches presented in this part of the thesis improve over static propositionalization while retaining computational efficiency. They are thus a step towards simple but principled statistical relational learning, while avoiding the complexity of more powerful and general approaches proposed in the literature. Despite their simplicity, the resulting systems have been shown to be surprisingly accurate. However, the greedy search strategy implemented in the underlying FOIL algorithm is only one simple alternative. Note that dynamic propositionalization can easily be combined with more advanced ILP methods and search strategies. This is an interesting direction for future work.

Part II

Efficient SRL for Sequential Data

Outline Part II

This part of the thesis introduces simple statistical relational models for sequential data. Statistical sequence models occupy an important position within the field of machine learning. They are not only theoretically appealing but have also proven to provide effective learning algorithms across many application areas, ranging from natural language processing to bioinformatics and robotics. In traditional sequence models, sequences are strings $s = w_1 \dots w_T$ over a (finite) alphabet Σ. Typically, the goal is to estimate the joint distribution $P(S)$ over all possible sequences. Given $P(S)$, several tasks can be solved, including sequence classification, sampling, or predicting the next state in a sequence given a history of preceding states (cf. Section 2.1.2).

Recent technological advances and progress in artificial intelligence have led to the generation of structured data sequences. One example is that of the smart phone, where communication events of many phone users have been logged during extended periods of time (Raento *et al.*, 2006). Other ones are concerned with logging the activities and locations of persons (Liao *et al.*, 2005b), or visits to websites (Anderson *et al.*, 2002). Finally, in bioinformatics, sequences often contain structural information (Durbin *et al.*, 1998). These developments, together with the interest in statistical relational learning, have motivated the development of probabilistic models for relational sequences. For instance, the Relational Markov Model (RMM) approach by Anderson *et al.* (2002) extends traditional Markov models to deal with relational sequences, while Logical Hidden Markov Models (LOHMMs) by Kersting *et al.* (2006) upgrade traditional HMMs towards relational sequences and also allow for logical variables and unification.

The setting that has received most attention in the SRL literature is that where sequence elements are individual atoms. It can be formally defined as follows:

Definition 5.0.1 (Learning from Sequences of Atoms)**.** *In the setting of learning from sequences of atoms, examples are sequences $s = p_1, ..., p_m$ where $p_i \in hb(\Sigma)$ are ground facts over a first-order alphabet Σ (cf. Chapter 2).*

Example 5.0.1. *The following are example sequences of atoms from different domains:*

$$outcall(\mathbf{11651}, fail), outcall(\mathbf{11651}, fail), incall(\mathbf{11651}, succ), outtxt(\mathbf{21344}), \dots$$

$mkdir(\mathbf{rgrams}), ls(\mathbf{rgrams}), emacs(\mathbf{rgrams.tex}), latex(\mathbf{rgrams.tex}), \ldots$

$strand(\mathbf{sa}, plus, short), helix(right, alpha, medium), strand(\mathbf{blb}, plus, short),$
$helix(right, f3to10, short), strand(\mathbf{sa}, minus, long), strand(\mathbf{blb}, plus, short) \ldots$

The first domain describes incoming and outgoing calls and text messages for a mobile phone user. For instance, $outcall(\mathbf{11651}, fail)$ *represents that a user has tried to reach phone number* $\mathbf{11651}$ *but failed. In the second domain, Unix shell commands executed by a user are described in terms of command names and arguments. In the third domain, helices and strands as protein secondary structure elements are defined in terms of their orientation, type, and length.*

The setting of learning from sequences of atoms is appropriate whenever sequence elements can be represented as atomic states. However, in some cases it is useful to consider a more general setting. For instance, one of the current challenges in artificial intelligence is the modeling of dynamic environments that change due to actions and activities people or other agents take. As one example, consider a model of the activities of a cognitively impaired person (Pollack, 2005). Such a model could be used to assist persons, using common patterns to generate reminders or detect potentially dangerous situations, and thus help to improve living conditions. As another example and one to which we will return in Chapter 6, consider a model of the environment in a *massively multiplayer online game* (MMOG). These are computer games that support thousands of players in complex, persistent, and dynamic virtual worlds. They form an ideal and realistic testbed for developing and evaluating artificial intelligence techniques and are also interesting in their own right (cf. also Laird and van Lent, 2000).

In such domains, states are characterized by a variable number of objects and relations amongst them. Thus, individual states—and therefore, sequence elements—are most easily represented as logical interpretations:

Definition 5.0.2 (Learning from Sequences of Interpretations)**.** *In the setting of learning from sequences of interpretations, examples consist of sequences* $s = I_1, ..., I_m$ *where* $I_i \subseteq hb(\Sigma)$ *are logical interpretations over a first-order alphabet* Σ *(cf. Chapter 2).*

Example 5.0.2. *As a simple example for a sequence of interpretations, consider the following sequence of observed world states and actions in the* blocks world *domain (see, for example, Slaney and Thiébaux, 2001):*

$$\boxed{\begin{array}{l} on(a, floor). \\ on(b, a). \\ on(c, floor). \\ move(b, c). \\ \end{array}}\ ,\ \boxed{\begin{array}{l} on(a, floor). \\ on(b, c). \\ on(c, floor). \\ move(a, b). \\ \end{array}}\ ,\ \boxed{\begin{array}{l} on(a, b). \\ on(b, c). \\ on(c, floor). \\ \\ \end{array}}\ ,\ \cdots$$

In the blocks world, blocks such as a or b can be on top of another block, indicated by $on(a, b)$, or on the $floor$. Moreover, a block can be moved on top of another block by an action such as $move(a, b)$. The state sequence denotes a sequence of actions (and resulting world states) in which blocks are re-stacked such that block a eventually is on top of block b.

As a more complex example for the setting of learning from sequences of interpretations, consider the discussion of the *Travian* MMOG domain in Section 6.3.2.

In the following, we introduce approaches for learning from sequences of atoms in Chapter 5 and sequences of interpretations in Chapter 6. In both settings, we start from the simplest—and also computationally most efficient—propositional sequence model, namely a Markov process (Definition 2.1.5, see Chapter 2). Choosing such a simple framework will allow us to obtain tractable models also in a relational setting.

The material in this part of the thesis is organized as follows. In Chapter 5, a simple model for learning from sequences of atoms is derived by upgrading n*-gram* models to the relational case. n-grams are amongst the most widely used sequence models for large-scale application domains, for instance, in natural language processing. In an n-gram model, Markov processes of different order are mixed to obtain smoothed distribution estimates. This provides more expressivity than a standard first-order Markov process, but at the same time maintains computational efficiency and avoids overfitting and data sparseness problems. In the relational r*-gram* model, smoothed distributions can be obtained by decreasing the order of the Markov process as well as by relational generalization of the r-gram. The relational grams are applied to user data collected in a Unix and Smartphone environment, and to a protein classification problem.

In Chapter 6, a Markov model for sequences of complete relational state descriptions, that is, complete logical interpretations, is introduced. Transition probabilities are defined using CP-logic, an expressive causal probabilistic logic recently introduced by Vennekens *et al.* (2006). However, by restricting CP-logic to the sequential case, employing a Markov assumption, and requiring fully observable data, inference and learning in CPT-L are significantly easier than in full CP-logic. The resulting CPT-L model thus occupies an intermediate position in the expressivity-efficiency trade-off inherent in statistical relational learning: it is able to model sequences of complex state descriptions, but employs a more restricted setting than more general approaches such as full CP-logic. We will present efficient inference and learning algorithms based on binary decision diagrams that are specifically tailored to the more restricted setting employed in CPT-L. The resulting system scales well to relatively large datasets. The efficiency and effectiveness of the system are evaluated in a massively multiplayer online game domain.

Chapter 5

Markov Chains for Sequences of Atoms*

This chapter introduces r-grams, which implement the setting of learning from sequences of atoms. The chapter is organized as follows. We start by reviewing n-grams as a simple but effective model for propositional sequences in Section 5.1. Afterwards, we derive the r-gram model as a relational extension of n-grams in Section 5.2. Finally, Section 5.3 reports on experiments using the proposed technique.

5.1 n-grams: Smoothed Markov Chains

n-grams are a simple but widely used probabilistic model for sequential data. They define a distribution over sequences $w_1...w_m$ of length m by an order $n-1$ Markov assumption:

$$P(w_1...w_m) = \prod_{i=1}^{m} P(w_i \mid w_{i-n+1}...w_{i-1}), \tag{5.1}$$

where $w_{i-n+1}...w_{i-1}$ should be interpreted as a shorthand for $w_{max(1,i-n+1)}...w_{i-1}$ for $i < n$. The name n-gram derives from the fact that all statistics needed to employ n-gram models can be computed from so-called "gram" counts $C(w_1...w_k)$, where the gram $w_1, ..., w_k$ is a sequence of states $w_i \in \Sigma$ and $C(w_1...w_k)$ is the number of times $w_1, ..., w_k$ appears as a subsequence in any of the training sequences $E = \{s_1, ..., s_N\}$ given to the learner. In the most basic case, the conditional probabilities are estimated from a set E of training sequences as relative frequencies:

$$P_n(w_i \mid w_{i-n+1} \dots w_{i-1}) = \frac{C(w_{i-n+1} \dots w_i)}{C(w_{i-n+1} \dots w_{i-1})}. \tag{5.2}$$

*This chapter builds on (Landwehr and De Raedt, 2007).

This is the estimate maximizing the likelihood

$$\begin{aligned} P(E) &= \prod_{i=1}^{N} P(s_i) \\ &= \prod_{i=1}^{N} \prod_{j=1}^{m_i} P(w_{i,j} \mid w_{i,j-n+1} ... w_{i,j-1}) \end{aligned}$$

where $s_i = w_{i,1}, ..., w_{i,m_i}$ and $P(s)$ is defined by Equation (5.1).

The gram order n defines the trade-off between reliability of probability estimates and discriminatory power of the model. Rather than selecting a fixed order n, performance can be increased by combining models of different order which are discriminative but can still be estimated reliably (Manning and Schütze, 1999). The two most popular approaches are back-off and interpolation estimates. In this chapter, we focus on the latter approach, which defines conditional distributions as linear combinations of models of different order:

$$P(w_i \mid w_{i-n+1} \ldots w_{i-1}) = \sum_{k=1}^{n} \alpha_k P_k(w_i \mid w_{i-k+1} \ldots w_{i-1}) \tag{5.3}$$

where the $\alpha_1, ..., \alpha_n$ are suitable weights with $\sum_{k=1}^{n} \alpha_k = 1$, and the lower-order distributions $P_k(w_i \mid w_{i-k+1} \ldots w_{i-1})$ are estimated according to maximum likelihood (Equation (5.2)). Several more advanced smoothing techniques have been proposed (cf. Manning and Schütze, 1999), but are beyond the scope of this chapter.

5.2 r-grams: Smoothed Relational Markov Chains

Let us now discuss how to extend n-gram models to sequences of logical atoms. Before defining the actual r-gram model, we have to discuss a special characteristic of such sequences, namely the occurrence of *object identifiers*. These are constants that identify a particular object in the domain. Object identifiers will be treated differently from other constants, as it is typically not appropriate to define a generative distribution over them. Accordingly, r-grams are generative models at the level of variablized sequences with local object identity constraints. These sequences define equivalence classes of ground sequences, in which elements are identical up to local identifier renaming.

5.2.1 Relational Sequences and Object Identifiers

An *atom* $p(t_1, \ldots, t_k)$ is a relation p of arity k that is followed by k terms t_i (cf. Chapter 2). In this chapter, we will work with three kinds of terms: constants (in $italic\ font$, such as $fail$), identifiers (in $\mathbf{bold\ roman\ font}$, such as $\mathbf{rgrams.tex}$) and variables (starting with an upper case character, such as X). That is, structured terms that are built up from functors will be excluded. Furthermore, we will assume that relations are *typed*: for each argument position i of the relation p, one can either have constants or identifiers. Note that the distinction between constants and identifiers will need to be made by the user, and the corresponding information be supplied to the system as part of its language bias. Variables may appear at any position. We shall also distinguish constant-variables from identifier-variables; the former can be instantiated to constants, the latter to identifiers. They will be written in $italic$ or $\mathbf{bold\ roman}$ font respectively. For instance, the relation $outcall$ has identifiers at the first position, and constants at the second one. Therefore, $outcall(\mathbf{11651}, fail)$ is type-conform. The types, constants, identifiers, variables and relations together specify the first-order alphabet Σ, that is, the set of type-conform atoms. $\bar{\Sigma} \subseteq \Sigma$ is the set of atoms that do not contain identifiers. A *sequence of atoms* is then a string $s = w_1, \ldots, w_m$ in Σ^*. Example sequences will typically be ground, cf. Example 5.0.1.

Notice that constants typically serve as attributes describing properties of the relations, and identifiers identify particular objects in the domain. Identifiers have no special meaning and they are only used for sharing object identity between events. Moreover, there is no fixed vocabulary for identifiers which is known a priori, rather, new identifiers will appear when applying a model to unseen data. Therefore, it is desirable to distinguish ground sequences only up to identifier renaming, which motivates the following definition.

Definition 5.2.1 (Sequence Congruence)**.** *Two sequences of atoms $p_1 \ldots p_m$, $q_1 \ldots q_m$ are $\mathbf{n}$-**congruent** if for all $i \in \{1, \ldots, m-n\}$ the subsequences $p_i \ldots p_{i+n-1}$, $r_i \ldots r_{i+n-1}$ are identical up to identifier renaming. Two sequences s_1, s_2 are identical up to identifier renaming if there exist a one-to-one mapping φ of the identifiers in s_1 to those in s_2 such that s_1 equals s_2 after replacing the identifiers according to φ.*

Example 5.2.1. *The sequences $r(\mathbf{x})p(a, \mathbf{y})r(\mathbf{y})p(b, \mathbf{x})$ and $r(\mathbf{z})p(a, \mathbf{w})r(\mathbf{w})p(b, \mathbf{u})$ are 3-congruent (but not 4-congruent).*

m-congruent sequences are identical up to identifier renaming. For $n < m$, the definition takes into account a limited history: two sequences are congruent if their object identity patterns are locally identical. Finally we note that n-congruence defines an equivalence relation, that is, it is reflexive, symmetric and transitive.

Let us now define a generative model for sequences of atoms. It should not sample actual identifier values, but rather equivalence classes of congruent sequences. This yields the following definition:

Definition 5.2.2 (Generative Model for Sequences of Atoms)**.** *Let Σ be a relational alphabet. Let $S(\Sigma)$ be the set of all ground sequences of length m over Σ, and let $\mathbb{S}_n(\Sigma)$ be the set of equivalence classes induced on $S(\Sigma)$ by n-congruence. Then a **generative model** of order n over Σ defines a distribution over $\mathbb{S}_n(\Sigma)$.*

Such a generative model can be learned from a set of ground sequences and the alphabet Σ. In its simplest form, the learning problem can be stated as maximum likelihood estimation:

Problem 5.2.1 (Maximum-Likelihood Learning from Sequences of Atoms)**.**

Given

- *a relational alphabet* Σ
- *a set* E *of ground sequences over* Σ
- $n \in \mathbb{N}$
- *a family* Λ *of generative models of order* n *over* Σ*,*

Find *a model* $\lambda \in \Lambda$ *that maximizes*

$$P(E \mid \lambda) = \prod_{s \in E} P([s] \mid \lambda)$$

where $[s]$ denotes the equivalence class of s with regard to n-congruence. A simple instance of such a family of generative models will be introduced next.

5.2.2 The r-gram Model

Consider ground sequences of atoms of the form $g_1 \dots g_m \in S(\Sigma)$. The key idea behind r-grams is a Markov assumption

$$P(g_1 ... g_m) = \prod_{i=1}^{m} P(g_i \mid g_{i-n+1} ... g_{i-1}). \tag{5.4}$$

However, defining conditional probabilities at the level of ground grams does not make sense in the presence of object identifiers. Thus, ground grams will be replaced by generalized ones. Generality is defined by a notion of subsumption:

Definition 5.2.3 (Subsumption)**.** *A sequence of atoms* $l_1, \dots, l_k$ *subsumes another sequence* $k_1, \dots, k_n$ *with substitution* θ*, notation* $l_1, \dots, l_k \preceq_\theta k_1, \dots, k_n$*, if and only if* $k \leq n$ *and* $\forall i, 1 \leq i \leq k : l_i\theta = k_i$*. A substitution is a set* $\{V_1/t_1, \dots, V_l/t_l\}$ *where the* V_i *are different variables and the* t_i *are terms, and we require that no identifier or identifier variable occurs twice in* $\{t_1, ..., t_l\}$*.*

The restriction on the allowed substitutions implements the object identity subsumption of Semeraro *et al.* (1995). The notion of sequence subsumption is due to Lee and De Raedt (2003). It can be tested in linear time.

Example 5.2.2. *Let* $\mathbf{Y}, \mathbf{Z}, \mathbf{U}, \mathbf{V}$ *be identifier variables.*

$$r(\mathbf{X})p(a, \mathbf{Y})r(\mathbf{Y}) \preceq_{\theta_1} r(\mathbf{w})p(a, \mathbf{u})r(\mathbf{u})$$
$$r(\mathbf{X})p(a, \mathbf{Y})r(\mathbf{Z}) \preceq_{\theta_2} r(\mathbf{W})p(a, \mathbf{U})r(\mathbf{V})$$

but

$$r(\mathbf{X})p(a,\mathbf{Y})r(\mathbf{Z}) \not\preceq r(\mathbf{W})p(a,\mathbf{U})r(\mathbf{U}).$$

with $\theta_1 = \{\mathbf{X}/\mathbf{w}, \mathbf{Y}/\mathbf{u}\}$ *and* $\theta_2 = \{\mathbf{X}/\mathbf{W}, \mathbf{Y}/\mathbf{U}, \mathbf{Z}/\mathbf{V}\}$.

We can now refine Equation (5.4) to take into account generalized sequences. This will be realized by defining

$$P(g_1...g_m) = \prod_{i=1}^{m} P(l_i \mid l_{i-n+1}...l_{i-1})$$

where $l_{i-n+1}...l_i \preceq_\theta g_{i-n+1}...g_i$. This generalization abstracts from identifier values and at the same time yields smoothed probability estimates, with the degree and characteristic of the smoothing depending on the particular choice of $l_{i-n+1}...l_i$. This is formalized in the following definition.

Definition 5.2.4 (r-gram Model)**.** *An* r***-gram model*** R *of order* n *over an alphabet* Σ *is a set of relational grams*

$$l_n^1 \vee ... \vee l_n^d \leftarrow l_1...l_{n-1}$$

where

1. $\forall i : l_1...l_{n-1}l_n^i \in \bar{\Sigma}^*$;
2. $\forall i : l_n^i$ *contains no constant-variables;*
3. $\forall i : l_n^i$ *is annotated with the probability values* $P_r(l_n^i \mid l_1...l_{n-1})$ *such that* $\sum_{i=1}^{d} P_r(l_n^i \mid l_1...l_{n-1}) = 1$
4. $\forall i \neq j : l_1...l_{n-1}l_n^i \not\preceq l_1...l_{n-1}l_n^j$; *that is, the heads are mutually exclusive;*
5. *there are no two grams in* R *with identical bodies.*

Example 5.2.3. *The following is an example of an order* 2 *relational gram in the mobile phone domain (see Example 5.0.1).*

$$\left.\begin{array}{rl} 0.3 & outtxt(\mathbf{X}) \\ 0.05 & outtxt(\mathbf{Y}) \\ 0.2 & outcall(\mathbf{X}, fail) \\ ... & \\ 0.05 & intxt(\mathbf{Y}) \end{array}\right\} \leftarrow outcall(\mathbf{X}, fail)$$

It states that after not reaching a person a user is more likely to write a text message to this person than to somebody else.

We still need to show that an r-gram model R defines a distribution over sequences of atoms. We first discuss a basic model by analogy to an unsmoothed n-gram, before extending it to a smoothed one in analogy to Equation (5.3).

A Basic Model

In the basic r-gram model, for any ground sequence $g_1...g_{n-1}$ there is exactly one gram $l_n^1 \vee ... \vee l_n^d \leftarrow l_1...l_{n-1}$ with $l_1...l_{n-1} \preceq_\theta g_1...g_{n-1}$. Its body $l_1...l_{n-1}$ is the most specific sequence in $\bar{\Sigma}^*$ subsuming $g_1...g_{n-1}$. Note that this implies that the gram can only contain identifier variables, but no constant variables. According to Equation (5.4), we start by defining a probability $P_R(g \mid g_1...g_{n-1})$ for any ground atom g given a sequence $g_1...g_{n-1}$ of ground literals. Let g be a ground literal and consider the above gram r subsuming $g_1...g_{n-1}$. If there is an $i \in \{1,...,d\}$ such that $l_1...l_{n-1}l_n^i \preceq_\theta g_1...g_{n-1}g$ it is unique and we define

$$\begin{aligned} P_R(g \mid g_1...g_{n-1}) &= P_r(g \mid g_1...g_{n-1}) \\ &= P_r(l_n^i \mid l_1...l_{n-1}). \end{aligned}$$

Otherwise, $P_R(g \mid g_1...g_{n-1}) = 0$. From $P_R(g \mid g_1...g_{n-1})$, a probability value $P_R(g_1...g_m)$ can be derived according to Equation (5.4). Note that this is *not* a distribution over all ground sequences of length m, as the model does not distinguish between n-congruent sequences. Instead, the following holds:

Lemma 5.2.1. *Let R be an order n r-gram over Σ, and $s, s' \in S(\Sigma)$ be sequences of atoms with s n-congruent to s'. Then $P_R(s) = P_R(s')$.*

Proof. Let $s = g_1,...,g_m$ and $s' = g'_1,...,g'_m$. Because

$$P_R(g_1...g_m) = \prod_{i=1}^{m} P_R(g_i \mid g_{i-n+1}...g_{i-1}),$$

$$P_R(g'_1...g'_m) = \prod_{i=1}^{m} P_R(g'_i \mid g'_{i-n+1}...g'_{i-1})$$

it suffices to show that for $i \in \{1,...,m\}$ $P_R(g_i \mid g_{i-n+1}...g_{i-1}) = P_R(g'_i \mid g'_{i-n+1}...g'_{i-1})$. Because s and s' are n-congruent, there is a mapping φ from identifiers in s to those in s' such that s equals s' after replacing identifiers according to φ (see Definition 5.2.1). This means that the gram $r = l_n^1 \vee ... \vee l_n^d \leftarrow l_1...l_{n-1}$ subsuming $g_{i-n+1}...g_{i-1}$ also subsumes $g'_{i-n+1}...g'_{i-1}$, as $l_1...l_{n-1}$ does not contain any ground identifiers (because of $l_i...l_{n-1} \in \bar{\Sigma}^*$), and therefore matches irrespective of any identifier renaming. Similarly, if there is a head literal l_n^k with $l_1...l_{n-1}l_n^k \preceq_\theta g_{i-n+1}...g_{i-1}g$ it also holds that $l_1...l_{n-1}l_n^k \preceq_\theta g'_{i-n+1}...g'_{i-1}g'$, and if there is no such literal no l_n^j fulfills $l_1...l_{n-1}l_n^j \preceq_\theta g'_{i-n+1}...g'_{i-1}g$. Thus, if a matching l_n^k exists,

$$\begin{aligned} P_R(g_i \mid g_{i-n+1}...g_{i-1}) &= P_r(l_n^k \mid l_{n+1}...l_1) \\ &= P_R(g'_i \mid g'_{i-n+1}...g'_{i-1}), \end{aligned}$$

and if not, $P_R(g_i \mid g_{i-n+1}...g_{i-1}) = P_R(g'_i \mid g'_{i-n+1}...g'_{i-1}) = 0.$ □

Let us define $P_R([s]) := P_R(s)$ for any $[s] \in \mathbb{S}_n(\Sigma)$, which is well-defined according to Lemma 5.2.1. It is now also easy to see that $\sum_{[s]\in\mathbb{S}_n(\Sigma)} P_R([s]) = 1$ in direct analogy to the propositional case. Therefore, we have

Theorem 5.2.1. *An order n r-gram over Σ is a generative model over Σ.*

Proof. Let $m \in \mathbb{N}$ be the length of the sequences under consideration. Let furthermore Σ be a relational alphabet, and R be an order n r-gram as in Definition 5.2.4. To prove that $\sum_{[s]\in\mathbb{S}_n(\Sigma)} P_R([s]) = 1$, we define a ground r-gram model R_g such that sequences that can be sampled from R_g correspond to equivalence classes $[s] \in \mathbb{S}_n(\Sigma)$ with $P_R([s]) > 0$. Because a ground r-gram model is a (propositional) n-gram, it is clear that the probabilities over sequences of length m sampled from R_g sum to one.

Let $var(r)$ denote the set of all variables occurring in a gram $r \in R$, and $K = max\{|var(r)| \mid r \in R\}$. We now introduce a set of K constants $\mathcal{C} = \{c_1, ..., c_K\}$. Let $ground(r) = \{r\theta \mid \theta : var(r) \rightarrow \mathcal{C}\}$ be all groundings of r using variables in $\mathcal{C}$, where substitutions θ respect object identity. $ground(r)$ is not an r-gram model, as there can be grams $h \leftarrow b, h' \leftarrow b$ in $ground(R)$ with the same body but different heads. Let R_g denote the r-gram model in which such ambiguities are removed by arbitrarily choosing one $h \leftarrow b$ for every b. R_g is a finite, propositional n-gram, and therefore defines a distribution over sequences of length m. Now it holds that 1) for any ground sequence s_g over constants in $\mathcal{C}$ we have $P_{R_g}(s_g) = P_R([s_g])$; 2) for s_g, s'_g sampled from R_g with $s_g \neq s'_g$ we have $[s_g] \neq [s'_g]$; and 3) for all $[s] \in \mathbb{S}_n(\Sigma)$ with $P_R([s]) > 0$ a ground sequence s_g can be sampled from R_g with $[s] = [s_g]$. This proves the theorem.

$\square$

Example 5.2.4. *Consider the r-gram model R with grams*

$$\begin{aligned} p(a, \mathbf{X}) \vee p(b, \mathbf{X}) &\leftarrow r(\mathbf{X}) \\ r(\mathbf{X}) &\leftarrow p(b, \mathbf{X}) \\ r(\mathbf{X}) \vee r(\mathbf{Y}) &\leftarrow p(a, \mathbf{X}) \\ r(\mathbf{X}) &\leftarrow \epsilon \end{aligned}$$

and uniform distributions over head literals. Here, ϵ is an artificial start symbol that only matches at the beginning of the sequence. The ground sequence $g_1...g_5 = r(\mathbf{u})p(a, \mathbf{u})r(\mathbf{v})p(b, \mathbf{v})r(\mathbf{v})$ has probability $P_R(g_1...g_5) = 1 \cdot 0.5 \cdot 0.5 \cdot 0.5 \cdot 1 = 0.125$.

Smoothing r-grams

In the basic model, there was exactly one gram $r \in R$ subsuming a ground subsequence $g_1 \cdots g_{n-1}$, namely the most specific one. As for n-grams, the problem with this approach is that there is a large number of such grams and the amount of training data needed to reliably estimate all of their frequencies is prohibitive unless n is very small. For n-grams, grams are therefore generalized by shortening their bodies, that is, smoothing with k-gram estimates for $k < n$ (Equation (5.3)).

The basic idea behind smoothing in r-grams is to generalize grams logically, and mix the resulting distributions:

$$P_R(g \mid g_1...g_{n-1}) = \sum_{r \in \hat{R}} \frac{\alpha_r}{\alpha} P_r(g \mid g_1...g_{n-1})$$

where $P_r(g \mid g_1...g_{n-1})$ is the probability defined by r as explained above, $\hat{R}$ is the subset of grams in R subsuming $g_1...g_{n-1}$, and α is a normalization constant with $\alpha = \sum_{r \in \hat{R}} \alpha_r$. The more general r, the more smooth the probability estimate $P_r(g \mid g_1...g_{n-1})$ will be. The actual degree and characteristic of the smoothing is defined by the set of matching r-grams together with their relative weights α_r.

By analogy with n-grams, additional smoothing can be obtained by also considering relational grams $r \in R$ with shorter bodies $l_1...l_{k-1}$, $k < n$. However, there is a subtle problem with this approach: Relational grams of order $k < n$ define a probability distribution over $\mathbb{S}_k(\Sigma)$ rather than $\mathbb{S}_n(\Sigma)$. Thus, the sequences are partitioned into a smaller number of equivalence classes. However, this can be taken care of by a straightforward normalization, which distributes the probability mass assigned to an equivalence class modulo k equally among all subclasses modulo n.

Example 5.2.5. *Consider the r-gram model R with grams r,q given by*

$$\begin{array}{ll} r: & r(\mathbf{X}) \vee r(\mathbf{Y}) \leftarrow r(\mathbf{X}) \\ q: & r(\mathbf{X}) \leftarrow \end{array}$$

uniform distribution over head literals and $\alpha_r = \alpha_q = 0.5$. We expect $P_R(r(\mathbf{u}) \mid r(\mathbf{v})) + P_R(r(\mathbf{v}) \mid r(\mathbf{v})) = 1$. However, when directly mixing distributions

$$P_R(r(\mathbf{u}) \mid r(\mathbf{v})) = \alpha_r P_r(r(\mathbf{Y}) \mid r(\mathbf{X})) + \alpha_q P_q(r(\mathbf{X})) = 0.75$$

$$P_R(r(\mathbf{v}) \mid r(\mathbf{v})) = \alpha_r P_r(r(\mathbf{X}) \mid r(\mathbf{X})) + \alpha_q P_q(r(\mathbf{X})) = 0.75,$$

as the r-gram q does not distinguish between the sequences $r(\mathbf{x})r(\mathbf{x})$ and $r(\mathbf{x})r(\mathbf{y})$. Instead, we mix by

$$P_R(r(\mathbf{x}) \mid r(\mathbf{y})) = \alpha_r P_r(r(\mathbf{X}) \mid r(\mathbf{X})) + \frac{1}{\gamma} \alpha_q P_q(r(\mathbf{X}))$$

where $\gamma = 2$ is the number of subclasses modulo 2-congruence of the class [r(X)].

The ultimate level of smoothing can be obtained by a relational gram r of the form $l_n^1 \vee ... \vee l_n^d \leftarrow$ where the l_n^i are fully variablized (also for non-identifier arguments). For this gram

$$P_r(g \mid g_1...g_{n-1}) = P_r(l_n^i) \prod_{j=1}^{a} P(X_j = x_j)$$

where $X_1, ..., X_a$ are the non-identifier arguments of l_n^i and $x_1, ..., x_a$ their instantiations in g. This case corresponds to an out-of-vocabulary event (observing an event that was not part of the training vocabulary) for n-grams.

Algorithm 4 Algorithm for building r-grams from data.

```
 1: procedure $r$-GRAMS((input: sequences $E$; alphabet: $\Sigma$; parameters: $\gamma, n$))
 2:     $B := \{l_1 \dots l_{k-1} \in \bar{\Sigma}^* | C(l_1 \dots l_{k-1}) > 0$ and $k \leq n\}$
 3:     for all $l_1 \dots l_{k-1} \in B$ do
 4:         let $\{l_k^1 \dots l_k^d\}$ contain all maximally specific literals $l_k^i \in \bar{\Sigma}$ such that
                $C(l_1 \dots l_{k-1} l_k^i) > 0$
 5:         add $r = l_k^1 \vee \cdots \vee l_k^d \leftarrow l_1 \dots l_{k-1}$ to $R$ with $P_r(l_k^i | l_1 \dots l_{k-1}) = \frac{C(l_1 \dots l_{k-1} l_k^i)}{C(l_1 \dots l_{k-1})}$
 6:         $L(r) := \prod_{g_1 \dots g_{n-1} g \in S(r)} P_r(g \mid g_1 \dots g_{n-1})$.
 7:         $t := |S(r)|$
 8:         $\alpha_r := L(r)^{\frac{\gamma}{t}}$
 9:     end for
10:     return $R$
11: end procedure
```

5.2.3 Building r-grams From Data

To learn an r-gram from a given set E of training sequences, we need to 1) choose the set R of relational grams; 2) estimate their corresponding conditional probabilities; and 3) define the weights α_r for every $r \in R$. Before specifying our algorithm, we need to define counts in the relational setting:

$$C(l_1 \dots l_k) = |\{i | s_1 \dots s_m \in S \text{ and } l_1 \dots l_k \preceq_\theta s_i \dots s_{i+k}\}|. \tag{5.5}$$

Our algorithm to learn r-grams is specified in Algorithm 4. In Line 2 of the algorithm, one computes all r-grams that occur in the data. Notice that no identifiers occur in these r-grams (cf. $\bar{\Sigma}^*$). This can be realized using either a frequent relational sequence miner, such as MineSeqLog (Lee and De Raedt, 2003), or using an on-the-fly approach. In the latter case, a relational gram with body $l_1 ... l_k$ is only built and added to R when and if it is needed to evaluate $P_R(g \mid g_1 ... g_{n-1})$ with $l_1 ... l_k \preceq g_1 ... g_k$ on unseen data. In Line 4, all possible literals l_k^i are sought that occur also in the data. They are maximally specific, which means that they do not contain constant-variables (cf. condition 2 of Definition 5.2.4). Line 5 then computes the maximum likelihood estimates and Lines 6–8 the weight α_r. Here S(r) denotes the set of all ground subsequences $g_1 ... g_{n-1} g$ appearing in the data which are subsumed by r. The likelihood $L(r)$ of r defined in Line 5 is a measure for how well the distribution defined by r matches the sample distribution. The α_r as in Line 7 is then defined in terms of $|S(r)|$ and the parameter γ, which controls the trade-off between smoothness and discrimination. Highly discriminative (specific) rules have higher likelihood than more general ones as they are able to fit the sample distribution better, and thus receive more weight if $\gamma > 0$.

5.3 Experimental Evaluation

This section reports on an empirical evaluation of the proposed method in several real-world domains. More specifically, we seek to answer the following questions:

(Q5.1) Are r-grams competitive with other state-of-the-art approaches for relational sequence classification?

(Q5.2) Is relational abstraction, especially of identifiers, useful?

Experiments were carried out on real-world sequence classification problems from three domains. In the **Unix** Shell domain (Greenberg, 1988; Jacobs and Blockeel, 2003), the task is to classify users as *novice programmers* or *non-programmers* based on logs of 3773 shell sessions containing 94537 commands (constants) and their arguments (identifiers). To reproduce the setting used by Jacobs and Blockeel (2003), we sampled 10 subsets of 50/1000 instances each from the data, measured classification accuracy on these using 10-fold cross-validation, and averaged the results. In the **Protein** fold classification domain, the task is to classify proteins as belonging to one of five folds of the SCOP hierarchy (Hubbard *et al.*, 1997). Strand names are treated as identifiers, all other ground terms as constants. This problem has been used as a benchmark before (Kersting *et al.*, 2006; Kersting and Gärtner, 2004), and we reproduce the experimental setting used in this earlier work: the same 200 examples per fold are used for training, and the remaining examples as the test set. In the Context **Phone** domain, data about user communication behavior has been gathered using a software running on Nokia Smartphones that automatically logs communication and context data. In our study, we only use information about incoming and outgoing calls and text messages. Phone numbers are identifiers, other ground terms constants. The task in **Phone I** is to discriminate between real sequences of events and "corrupted" ones, which contain the same sequence elements but in random order. For $k \in \{20, 40, 60, 80, 100\}$, 5 subsets of size k were sampled randomly, 5-fold cross-validation performed and averaged for each k. In **Phone II**, the task is to classify communication logs as belonging to one of three users, based only on their communication patterns but without referring to actual phone numbers in the event sequence.

In all domains sequence classification is performed by building an r-gram model R_C for each class C and labeling unseen sequences s with the class that maximizes $P_C(s)P(C)$. We used bigram models in the Phone II domain and trigram models for all other domains, and the smoothing parameter γ was set to 1 in all experiments. Learning the r-gram model was done on-the-fly as explained in Section 5.2.3.

Table 5.1 compares the classification accuracy of r-grams with accuracy results from the literature in the Protein Fold and Unix Shell domains. In the Protein Fold domain, a hand-crafted Logical Hidden Markov Model achieves 74% accuracy (Kersting *et al.*, 2006). This has been improved to 82.4% by a fisher kernel approach, in which the gradient of the likelihood function of the Logical Hidden Markov Model is used as input in a support vector machine (Kersting and Gärtner, 2004). The Unix Shell log classification problem was originally tackled using a k-nearest neighbor method based on customized

Table 5.1: Comparison of classification accuracy of r-grams to Logical Hidden Markov models and Fisher kernels in the Protein Fold domain, and to k-nearest neighbor and C4.5 in the Unix Shell domain. For protein fold prediction, a single split into training and test set is used. In the Unix Shell domain, 10 subsets of 50/1000 examples each are randomly sampled, accuracy determined by 10-fold cross-validation and averaged.

Domain	**r-grams**	**LOHMM**	**LOHMM + FK**
Protein	83.3	74.0	82.7
Domain	**r-grams**	**kNN**	**C4.5**
Unix-50	93.8 ± 2.7	91.0	88.8
Unix-1000	97.2 ± 0.4	95.3	94.7

Table 5.2: Accuracy comparison of r-grams to n-grams, and to n-grams w/o IDs. For Protein/Unix domains settings are as before. For the Phone I domain, 5 subsets of size 100 have been sampled from the data, a 5-fold cross-validation is performed on each set and results are averaged. For Phone II, results are based on one 5-fold cross-validation. Results for **n-grams** are based on one sample only.

Domain	**r-grams**	**n-grams**	**n-grams w/o IDs**
Protein	83.3	79.7	83.3
Unix-50	93.8 ± 2.7	74.4	95.6 ± 2.7
Unix-1000	97.2 ± 0.4	76.3	97.1 ± 0.4
Phone I	95.0 ± 2.0	30.0	86.8 ± 3.3
Phone II	93.3 ± 14.9	33.3	86.7 ± 18.3

sequence similarity (Jacobs and Blockeel, 2003). In the same paper, the authors present results for a C4.5 decision tree learner using a bag-of-words representation. In both cases, r-grams yield competitive classification accuracy, which is a positive answer to question **(Q5.1)**. Furthermore, we note that even using a naïve implementation r-grams are computationally efficient. Times for building an r-gram model ranged from 3 to 240 seconds in the presented experiments[1].

In a second set of experiments, the effect of using relational abstraction was examined in more detail. More precisely, r-grams were compared to n-grams which implement non-relational smoothing as outlined in Section 5.1, treating the atoms in Σ as flat symbols. For these experiments, we tried keeping identifiers in the events (**n-grams**) or removing them from the data (**n-grams w/o IDs**). Accuracy results for the Protein Fold, Unix Shell and Context Phone domains are given in Table 5.2. If identifiers are treated as normal constants, accuracy is reduced in all cases, especially for the identifier-rich Unix and Context Phone domains. This is not surprising, as most identifiers appearing in the test data have

[1]All experiments were run on standard PC hardware with 3.2GHz processor and 2GB of main memory.

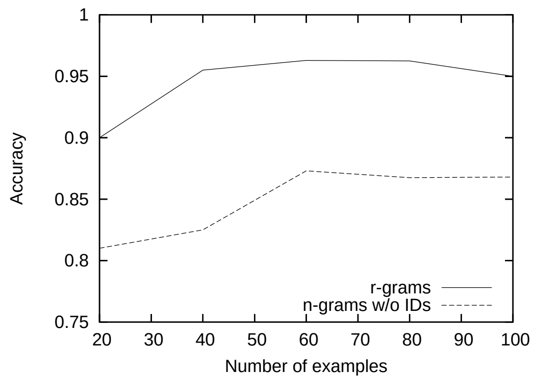


Figure 5.1: Accuracy for different training set sizes ranging from 20 to 100 examples in the Phone I domain. For each size, 5 sets are sampled, a 5-fold cross-validation is performed and the result averaged.

never been observed in the training data and thus the corresponding event has probability zero. When identifiers are removed from the data, performance is similar as for r-grams in the Protein Fold and Unix Shell domain, but worse in the Context Phone domains. On Phone I, r-grams significantly outperform n-grams w/o IDs (unpaired sampled t-test, p = 0.01). The other differences in accuracy in Table 5.2 are not significant. Figure 5.1 shows more detailed results on Phone I for different numbers of training examples. It can be observed that relational abstraction is particularly helpful for small numbers of training examples. To summarize, there are domains where it is possible to ignore identifiers in the data, but in other domains relational abstraction is essential for good performance. Therefore, question **(Q5.2)** can be answered affirmatively as well.

5.4 Related and Future Work

The r-grams formalism is related to other work on analyzing relational and logical sequences. This includes most notably, the RMMs by Anderson *et al.* (2002) who presented a relational Markov model approach, and the LOHMMs by Kersting *et al.* (2006) who introduced Logical Hidden Markov Models. RMMs define a probability distribution over ground sequences of atoms *without identifiers*. Furthermore, they do not employ unification (or variable propagation) but realize shrinkage through the use of taxonomies over the constant values appearing in the atoms and decision trees to encode the probability values. RMMs only consider first order Markov models (the next state only depends on the previous one), so RMMs do not smooth over sequences of variable length. RMMs have been applied to challenging applications in web-log analysis. Whereas RMMs upgrade Markov models, LOHMMs upgrade HMMs to work with logical sequences. As r-grams, they allow for unification and offer also the ability to work with identifiers. In addition, they enable one to work with structured terms (and functors). However, as RMMs, they only consider first order Markov models. Furthermore, they do not smooth distributions using models of different specificity. Finally, MineSeqLog (Lee and De Raedt, 2003) is a frequent relational sequences miner that employs subsumption to test whether a pattern matches a sequence. One might consider employing it to tackle the first step in the r-gram algorithm.

There are several directions for further work. In the current r-gram model, structured first-order terms including functors are not considered. The formalism could be extended to include functors, further extending its applicability to complex domains. Moreover, different smoothing techniques could be considered. For instance, *back-off smoothing* is a popular smoothing technique for propositional grams. The basic idea behind back-off smoothing is to revert to a lower-order gram if counts for the full n-gram are below a certain threshold, as this can imply imprecise probability estimates. Such techniques could in principle also be applied in the r-gram formalism. Perhaps most importantly, it is interesting to apply the work to further challenging artificial intelligence applications.

Chapter 6

Markov Chains for Sequences of Interpretations*

After presenting a simple model for learning from sequences of atoms in the previous chapter, we now discuss an extended setting in which sequence elements are full logical interpretations. Specifically, we introduce the CPT-L system, a simple extension of the Markov chain framework to logical interpretations.

As a motivating application domain—that will also be used to evaluate our method empirically, see Section 6.3—we will consider massively multiplayer online games (MMOGs). As discussed above, such games are characterized by complex state descriptions including multiple agents and relations between them, and thus constitute a natural test-bed for approaches that learn from sequences of interpretations.

One challenge in such games is to build a dynamic probabilistic model of high-level player behavior, such as players joining or leaving alliances and concerted actions by players within one alliance. Such a model of human cooperative behavior in this type of world can be useful in several ways. Analysis of in-game social networks is not only interesting from a sociological point of view but could also be used to visualize aspects of the gaming environment or give advice to inexperienced players (for instance, which alliance to join). More ambitiously, the model could be used to build computer-controlled players that mimic the cooperative behavior of human players, form alliances and jointly pursue goals that would be impossible to attain otherwise. Mastering these social aspects of the game will be crucial to building smart and challenging computer-controlled opponents, which are currently lacking in most MMOGs. Finally, the model could also serve to detect non-human players in todays MMOGs — accounts which are played by automatic scripts to give one player an unfair advantage, and are typically against game rules.

*This chapter builds on joint work with Ingo Thon, published in (Thon *et al.*, 2008).

6.1 CPT-L: A Markov Model for Sequences of Interpretations

To work towards an efficient model for sequences of interpretations, let us start with a simple Markov model approach as used in the previous section. In the new setting, a first-order Markov assumption entails

$$\begin{aligned} P(I_0, ..., I_T) &= P(I_0) \prod_{t=1}^{T} P(I_t \mid I_{t-1}, ..., I_0) \\ &= P(I_0) \prod_{t=1}^{T} P(I_t \mid I_{t-1}). \end{aligned}$$

where $I_0, ..., I_T$ is an observed sequence of interpretations over a first-order alphabet Σ, that is, $I_t \subseteq hb(\Sigma)$ (cf. Example 5.0.2). The crucial challenge in this approach is to define the conditional distribution $P(I_t \mid I_{t-1})$ for arbitrary logical interpretations I_t, I_{t-1}. Note that, in contrast to propositional models, a naïve representation of the conditional by a "table" with one entry for any pair $(I_t, I_{t-.1})$ of interpretations is infeasible, as the number of interpretations is exponential in the size of the logical language considered.

In the rest of this section, we show how such conditional distributions can be realized by employing *CP-logic* (Vennekens *et al.*, 2006), a recent expressive logic for modeling causality. More specifically, we discuss the *CPT-L* framework, which adapts CP-logic to the sequential setting. The semantics of CPT-L is based on CP-logic, a probabilistic first-order logic that defines probability distributions over interpretations (Vennekens *et al.*, 2006). CP-logic has a strong focus on causality and constructive processes: an interpretation is incrementally constructed by a process that adds facts which are probabilistic *outcomes* of other already given facts (the *causes*). CPT-L combines the semantics of CP-logic with that of (first-order) Markov processes. Causal influences only stretch from I_t to I_{t+1} (Markov assumption), are identical for all time-steps (stationarity), and all causes and outcomes are observable. Models in CPT-L are also called CPT-theories, and can be formally defined as follows:

Definition 6.1.1 (CPT-Theory)**.** *A* **CPT-theory** *is a set of rules of the form*

$$r = (h_1 : p_1) \vee \ldots \vee (h_n : p_n) \leftarrow b_1, \ldots, b_m$$

where the h_i are logical atoms and the b_i are literals over Σ, and $p_i \in [0, 1]$ probabilities s.t. $\sum_{i=1}^{n} p_i = 1$.

It will be convenient to refer to $b_1, ..., b_m$ as the body $body(r)$ of the rule and to $(h_1 : p_1) \vee \ldots \vee (h_n : p_n)$ as the head $head(r)$ of the rule. We shall also assume that the rules are range-restricted, that is, that all variables appearing in the head of the rule also appear in its body. Rules define conditional probabilistic events: the intuition behind a rule is that whenever $b_1\theta, ..., b_m\theta$ holds for a substitution θ in the current state I_t, exactly one of the $h_i\theta$ in the head will hold in the next state I_{t+1}. In this way, the rule models a (probabilistic) causal process as the condition specified in the body causes one (probabilistically chosen) atom in the head to become true in the next time-step.

Example 6.1.1. *Consider the following CPT-rule for the* blocks world *domain introduced in Example 5.0.2:*

$$(on(a, floor) : 0.9) \vee (on(a, c) : 0.1) \leftarrow free(a), on(a, c), move(a, floor).$$

which represents that we try to move a block A from block C to the table. This action succeeds with a probability of 0.9*. Note that $free/1$ is a background predicate (see below).*

We now show how a CPT-theory defines a distribution over sequences $I_0, ..., I_T$ of relational interpretations over a first-order alphabet Σ. Let us first define the concept of the applicable ground rules in an interpretation $I_t \subseteq hb(\Sigma)$. From a CPT-theory, the rule $(h_1 : p_1\theta) \vee \ldots \vee (h_n : p_n\theta) \leftarrow b_1\theta, \ldots, b_m\theta$ is obtained for a substitution θ. A ground rule r is applicable in I_t if and only if $body(r) = b_1\theta, \ldots, b_m\theta$ is true in I_t, denoted $I_t \models b_1\theta, \ldots, b_m\theta$.

One of the main features of CPT-theories is that they are easily extended to include *background knowledge*. The background knowledge B can be any logic program, cf. (Bratko, 1990). In the presence of background knowledge, a ground rule is applicable in an interpretation I_t if $b_1\theta, \ldots, b_m\theta$ can be logically derived from I_t together with the logic program B, denoted $I_t \models_B b_1\theta, \ldots, b_m\theta$.

Example 6.1.2. *To illustrate the use of background knowledge, reconsider the blocks-world domain introduced in Example 5.0.2. A possible background theory B for this domain is given by the two clauses*

$$free(floor) \leftarrow$$
$$free(B) \leftarrow \neg on(X, B)$$

indicating that a block is free if there is no other block stacked on top of it, and the floor is always free.

The set of all applicable ground rules in state I_t will be denoted as $\mathbf{R}_t$. Each ground rule applicable in I_t will cause one of its head elements to become true in I_{t+1}. More formally, let $\mathbf{R}_t = \{r_1, ..., r_k\}$. A *selection* σ is a mapping $\{(r_1, j_1), ..., (r_k, j_k)\}$ from ground rules r_i to indices j_i denoting that head element $h_{ij_i} \in head(r_i)$ is selected. The probability of a selection σ is

$$P(\sigma) = \prod_{i=1}^{k} p_{j_i}, \qquad (6.1)$$

where p_{j_i} is the probability associated with head element h_{ij_i} in r_i. In the stochastic process to be defined, I_{t+1} is a possible successor for the state I_t if and only if there is a selection σ such that $I_{t+1} = \{h_{1\sigma(1)}, ..., h_{k\sigma(k)}\}$. We shall say that σ *yields* I_{t+1} in I_t, denoted $I_t \xrightarrow{\sigma} I_{t+1}$, and define

$$P(I_{t+1}|I_t) = \sum_{\sigma : I_t \xrightarrow{\sigma} I_{t+1}} P(\sigma). \tag{6.2}$$

Example 6.1.3. *Consider the theory*

$$\begin{aligned} r_1 &= a : 0.2 \vee b : 0.8 \leftarrow \neg a, \neg b \\ r_2 &= a : 0.5 \vee b : 0.5 \leftarrow a \\ r_3 &= a : 0.7 \vee nil : 0.3 \leftarrow a \end{aligned}$$

Starting from $I_t = \{a\}$ only the rules r_2 and r_3 are applicable, so $\mathbf{R}_t = \{r_2, r_3\}$. The set of possible selections is

$$\{(r_2, j_2), (r_3, j_3) \mid j_2, j_3 \in \{1, 2\}\}.$$

The possible successor states I_{t+1} are therefore

$$\begin{aligned} &I^1_{t+1} = \{a\} \textit{ with } P(I^1_{t+1} \mid I_t) = 0.5 \cdot 0.7 + 0.5 \cdot 0.3 = 0.5 \\ &I^2_{t+1} = \{b\} \textit{ with } P(I^2_{t+1} \mid I_t) = 0.5 \cdot 0.3 = 0.15 \\ &I^3_{t+1} = \{a, b\} \textit{ with } P(I^3_{t+1} \mid I_t) = 0.5 \cdot 0.7 = 0.35 \end{aligned}$$

As for propositional Markov processes, the probability of a sequence $I_0, ..., I_T$ given an initial state I_0 is defined by

$$P(I_0, ..., I_T) = P(I_0) \prod_{t=0}^{T} P(I_{t+1} \mid I_t). \tag{6.3}$$

We now have—in direct analogy to the propositional case—that by Equation (6.3) a CPT-theory defines a distribution over sequences of interpretations:

Theorem 6.1.1 (Semantics of a CPT theory)**.** *Given an initial state I_0, a CPT-theory defines a discrete-time stochastic process, and therefore for $T \in \mathbb{N}$ a distribution $P(I_0, ..., I_T)$ over sequences of interpretations of length T.*

Proof. The theorem follows directly from the semantics of CP-logic (Vennekens *et al.*, 2006) and the way a Markov process defines a distribution over all sequences of a fixed length T. According to the semantics of CP-logic, it holds that

$$\sum_{I_{t+1} \subseteq hb(\Sigma)} P(I_{t+1} \mid I_t) = 1,$$

that is, CPT-L defines a distribution over the next ground state $I_{t+1} \subseteq hb(\Sigma)$ given the current ground state $I_t \subseteq hb(\Sigma)$ by means of Equation (6.2). Thus, a CPT-L theory defines a ground Markov process, and it follows that

$$\sum_{s=I_0,...,I_T} P(s) = 1,$$

where $I_t \subseteq hb(\Sigma)$ for $t = 1, ..., T$. $\square$

It is instructive to compare the semantics given by Theorem 6.1.1 with that of the r-gram model given by Theorem 5.2.1. r-grams do not directly define a distribution over ground sequences, but variablized sequences with local object identifier constraints. This is a flexible way to deal with identifiers in sequences, which typically cannot be modeled explicitly. For CPT-L, we avoid these complications by only modeling a distribution over sequences *given an initial state* I_0 (cf. Theorem 6.1.1). When modeling real-world domains in CPT-L, this initial state can contain all identifiers appearing in the sequence, thus we again do not need to sample them. However, the approach pursued in r-grams is slightly more flexible as it does not require us to list all identifiers a priori in I_0. Exploring identifier abstraction for CPT-L is an interesting direction for future work.

6.2 Inference and Parameter Estimation

As for other probabilistic models, we can now ask several questions about the introduced CPT-L model (confer Problem 2.1.2):

- **Sampling**: how to sample sequences of interpretations $I_0, ..., I_T$ from a given CPT-theory $\mathcal{T}$ and initial interpretation I_0?
- **Inference**: given a CPT-theory $\mathcal{T}$ and a sequence of interpretations $I_0, ..., I_T$, what is $P(I_0, ..., I_T \mid \mathcal{T})$?
- **Parameter Estimation**: given the structure of a CPT-theory $\mathcal{T}$ and a set of sequences of interpretations, what are the maximum-likelihood parameters of $\mathcal{T}$?
- **Prediction**: Let $\mathcal{T}$ be a CPT-theory, $I_0, ..., I_t$ a sequence of interpretations, and F a first-order formula that constitutes a certain property of interest. What is the probability that F holds at time $t + d$, $P(I_{t+d} \models_B F \mid \mathcal{T}, I_0, ..., I_t)$?

Sampling from a CPT-theory $\mathcal{T}$ given an initial interpretation I_0 is straightforward due to the causal semantics employed in CP-logic. For $t \geq 0$, I_{t+1} can be constructed from I_t by finding all groundings $r\theta$ of rules $r \in \mathcal{T}$, and sampling for each $r\theta$ a head element to be added to I_{t+1}. Algorithmic solutions for solving the inference, parameter estimation, and prediction problem are presented in turn in the rest of this section.

6.2.1 Inference

Because of the Markov assumption (Equation (6.3)), the crucial task for solving the inference problem is to compute $P(I_{t+1} \mid I_t)$ for given I_{t+1} and I_t. According to Equation (6.2), this involves summing the probabilities of all selections yielding I_{t+1} from I_t. However, the number of possible selections σ is exponential in the number of ground rules $|\mathbf{R}_t|$ applicable in I_t, so a naïve generate-and-test approach is infeasible. Instead, we present an efficient approach for computing $P(I_{t+1} \mid I_t)$ without explicitly enumerating all selections yielding I_{t+1}, which is strongly related to the inference technique discussed by De Raedt *et al.* (2007). The problem is first converted to a DNF formula over boolean variables such that assignments to variables correspond to selections, and satisfying assignments to selections yielding I_{t+1}. The formula is then compactly represented as a binary decision diagram (BDD), and $P(I_{t+1} \mid I_t)$ efficiently computed from the BDD using dynamic programming. Although finding satisfying assignments for DNF formulae is a hard problem in general, the key advantage of this approach is that existing, highly optimized BDD software packages can be used.

The conversion of a given inference problem to a DNF formula f is realized as follows:

1. Initialize $f := true$
2. Compute applicable ground rules
$$\mathbf{R}_t = \{r\theta | body(r\theta) \text{ is true in } I_t\}$$
3. For all rules $(r = (p_1 : h_1, ..., p_n : h_n) \leftarrow b_1, ..., b_m)$ in $\mathbf{R}_t$ do:
   (a) $f := f \wedge (r.h_1 \vee ... \vee r.h_n)$, where $r.h$ denotes the proposition obtained by concatenating the name of the rule r with the ground literal h resulting in a new propositional variable $r.h$ (if not $h_i = nil$).
   (b) $f := f \wedge (\neg r.h_i \vee \neg r.h_j)$ for all $i \neq j$
4. For all facts $l \in I_{t+1}$
   (a) Initialize $g := false$
   (b) for all $r \in \mathbf{R}_t$ with $p : l \in head(r)$ do $g := g \vee r.l$
   (c) $f := f \wedge g$

Boolean variables of the form $r.h$ represent that head element h was selected in rule r[1]. The second step of the algorithm computes all applicable rules, the third step assures that selections are obtained, and the final step assures that the selection generates the interpretation I_{t+1}. It is easily verified that the satisfying assignments for the formula f correspond to the selections yielding I_{t+1}.

[1] Variables $r.h$ are standardized apart in case head elements coincide after grounding.

Example 6.2.1. *The following formula f is obtained for the transition $\{a\} \rightarrow \{a, b\}$ and the CPT-theory given in Example 6.1.3.*

$$\underbrace{(r2.a \vee r2.b)}_{3.a} \wedge \underbrace{(\neg r2.a \vee \neg r2.b) \wedge (\neg r3.a \vee \neg r3.nil)}_{3.b} \wedge \underbrace{(r2.a \vee r3.a) \wedge r2.b}_{4}$$

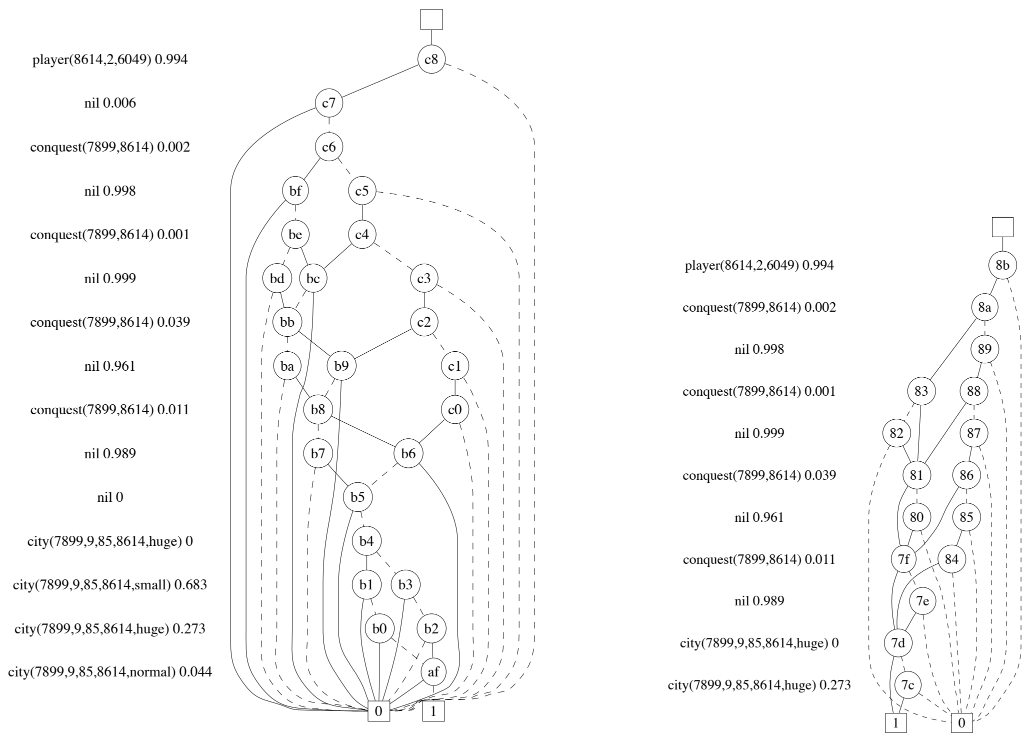
The parts of the formula are annotated with the steps in the construction algorithm that generated them.

From the formula f, a *reduced ordered binary decision diagram* (BDD) (Bryant, 1986) is constructed. Let $x_1, ..., x_n$ denote an ordered set of boolean variables (such as the $r.h$ contained in f). A BDD is a rooted, directed acyclic graph, in which nodes are annotated with variables and have out-degree 2, indicating that the variable is either true or false. Furthermore, there are two terminal nodes labeled with 0 and 1. Variables along any path from the root to one of the two terminals are ordered according to the given variable ordering. The graph compactly represents a boolean function f over variables $x_1, ..., x_n$: given an instantiation of the x_i, we follow a path from the root to either 1 or 0 (indicating f is true or false). Furthermore, the graph must be reduced, that is, it must not be possible to merge or remove nodes without altering the represented function (cf. Bryant, 1986, for details). Figure 6.1, left, shows an example BDD.

From the BDD graph, $P(I_{t+1} \mid I_t)$ can be computed in linear time using dynamic programming. This is realized by a straightforward modification of the algorithm for inference in ProbLog theories (De Raedt *et al.*, 2007). The algorithm exploits that paths in the BDD from the root node to the 1-terminal correspond to satisfying assignments for f, and thus selections yielding I_{t+1}. By sweeping through the BDD from top to bottom contributions from all such selections are summed up (Equation (6.2)) without explicitly enumerating all paths. The efficiency of this method crucially depends on the size of the BDD graph, which in turn depends strongly on the chosen variable ordering $x_1, ..., x_n$. Unfortunately, computing an optimal variable ordering is NP-hard. However, existing implementations of BDD packages contain sophisticated heuristics to find a good ordering for a given function in polynomial time.

Interestingly, it is possible to further reduce complexity for the particular problem we are interested in by adapting a different semantics in the BDD. A *zero-suppressed binary decision* diagrams (or ZDD) is an alternative form of graphical representation in which variables appear in a path only if their positive branch is not directly connected to the terminal 0 (Minato, 1993). Figure 6.1 shows example BDD and ZDD structures that represent the same function. We now show that a reduced ZDD representation of f will always be smaller than (or identical to) the corresponding BDD representation for CPT-L:

Theorem 6.2.1. *Let f be a formula resulting from the conversion of a CPT-L inference problem, G its BDD representation, and G' its ZDD representation (for a fixed variable ordering). Then $size(G') \leq size(G)$.*


player(8614,2,6049) 0.994
nil 0.006
conquest(7899,8614) 0.002
nil 0.998
conquest(7899,8614) 0.001
nil 0.999
conquest(7899,8614) 0.039
nil 0.961
conquest(7899,8614) 0.011
nil 0.989
nil 0
city(7899,9,85,8614,huge) 0
city(7899,9,85,8614,small) 0.683
city(7899,9,85,8614,huge) 0.273
city(7899,9,85,8614,normal) 0.044
player(8614,2,6049) 0.994
conquest(7899,8614) 0.002
nil 0.998
conquest(7899,8614) 0.001
nil 0.999
conquest(7899,8614) 0.039
nil 0.961
conquest(7899,8614) 0.011
nil 0.989
city(7899,9,85,8614,huge) 0
city(7899,9,85,8614,huge) 0.273

(a) BDD representation

(b) ZDD representation

Figure 6.1: Graphical representation of a formula f resulting from the conversion of a CPT-L inference problem represented as a BDD and ZDD.

Proof. We first show that in G every path Q from the root to the 1-terminal contains all variables appearing in f. Assume $r.h_1 \notin Q$, and let $r.h_2, ..., r.h_l$ denote the variables corresponding to the other head elements of rule r. Because of the constraint added in step 3. of the conversion, f can only be true if exactly one of the $r.h_1, ..., r.h_l$ is true. However, this cannot be verified by looking at any subset of the variables, and therefore they must all be contained in the path. Because all variables appear in every path from the root to 1, the graph structure G is also a faithful representation of f under the ZDD semantics. If G as a ZDD is fully reduced, $G = G'$ because reduced ZDDs, as BDDs, are a canonical representation. Otherwise, G can be further reduced to the ZDD G' with $size(G') < size(G)$. □

Typically a ZDD representation of f will be more compact than the BDD representation, as shown in Figure 6.1.

6.2.2 Parameter Estimation

Assume the structure of a CPT-theory is given, that is, a set $\mathcal{T} = \{r_1, ..., r_k\}$ of rules of the form

$$r_i = (h_{i1} : p_{i1}) \vee \ldots \vee (h_{in} : p_{in}) \leftarrow b_{i1}, \ldots, b_{im},$$

where $\pi = \{p_{ij}\}_{i,j}$ are the unknown parameters to be estimated from a set of training sequences D. A standard approach is to find max-likelihood parameters $\pi^* = \arg\max_\pi P(D \mid \pi)$. To determine a model parameter p_{ij}, we essentially need to know the number of times head element h_{ij} has been selected in an application of the rule r_i in the training data, which will be denoted by κ_{ij}. However, the quantity κ_{ij} is not directly observable. To see why this is so, first consider a single transition $I_t \rightarrow I_{t+1}$ in one training sequence. We know the set of rules $\mathbf{R}_t$ applied in the transition; however, there are in general many possible selections σ of rule head elements yielding I_{t+1}. The information which selection was used, that is, which rule has generated which fact in I_{t+1}, is hidden. We now derive an efficient Expectation-Maximization algorithm in which the unobserved variables are the selections used at every transition, and κ_{ij} the sufficient statistics. To keep the notation uncluttered, we present the expectation step $\mathbb{E}[\kappa_{ij} \mid \pi, D]$ for a single transition $\tau = I_t \rightarrow I_{t+1}$; contributions from different transitions and different training sequences simply sum up. Let $\Gamma = \{\sigma \mid I_t \xrightarrow{\sigma} I_{t+1}\}$ denote the set of selections yielding τ. The expectation is

$$\begin{aligned}\mathbb{E}[\kappa_{ij} \mid \pi, \tau] &= \sum_\sigma P(\delta_{ij} \mid \sigma, \pi, \tau) \\ &= \sum_\sigma P(\delta_{ij} \mid \sigma) P(\sigma \mid \pi, \tau) \\ &= \sum_{\sigma \in \Gamma} P(\delta_{ij} \mid \sigma) \frac{P(\sigma \mid \pi)}{\sum_{\sigma' \in \Gamma} P(\sigma' \mid \pi)} \end{aligned} \tag{6.4}$$

where δ_{ij} is an indicator variable representing that head h_{ij} was selected in rule r_i. Note that $P(\delta_{ij} \mid \sigma)$ is simply 1 if the head is selected in σ and 0 otherwise, and $P(\sigma \mid \pi)$ is defined by Equation (6.1). Given the expectation, the maximization step is

$$p_{ij}^{(new)} = \frac{\mathbb{E}[\kappa_{ij} \mid \pi, D]}{\sum_j \mathbb{E}[\kappa_{ij} \mid \pi, D]}.$$

The key algorithmic challenge is to compute the expectation given by Equation (6.4) efficiently. As outlined above, the set Γ of selections yielding the observed transitions can be compactly represented as the set of paths from the root to the 1-terminal in a (possibly zero-suppressed) decision diagram.

By analogy to the inference problem, the summation given by Equation (6.4) can be performed in linear time given the BDD (ZDD) structure. This is realized by a dynamic programming algorithm similar to the forward-backward algorithm in hidden Markov models (Rabiner, 1989) that sweeps through the BDD structure twice to accumulate the sufficient statistics κ_{ij}. Details of the algorithm are straightforward but somewhat involved, and omitted for lack of space. Note that the presented Expectation-Maximization algorithm, by taking the special structure of our model into account, is significantly more efficient than general-purpose parameter learning techniques employed in CP-logic.

6.2.3 Prediction

Assume we are given a (partial) observation sequence $I_0, ..., I_t$, a CPT-theory $\mathcal{T}$, and a property of interest F (represented as a first-order formula), and would like to compute $P(I_{t+d} \models_B F \mid I_0, ..., I_t, \mathcal{T})$. For instance, a robot might like to know the probability that a certain world state is reached at time $t + d$, given its current world model and observation history. Note that the representation as a first-order formula enables one to express richer world conditions than queries on (sets of) atoms, as they are typically supported in statistical relational learning systems. In CPT-L,

$$P(I_{t+d} \models_B F \mid I_0, ..., I_t, \mathcal{T}) = P(I_{t+d} \models_B F \mid I_t, \mathcal{T})$$

as the world model is Markov. Powerful statistical relational learning systems are in principle able to compute this quantity exactly by "unrolling" the world model into a large dynamic graphical model. However, this is computationally expensive as it requires to marginalize out all (unobserved) intermediate world states $I_{t+1}, ..., I_{t+d-1}$. In contrast, inference in CPT-theories draws its efficiency from the full observability assumption.

As an alternative approach, we propose a straightforward sample-based approximation to $P(I_{t+d} \models_B F \mid I_t, \mathcal{T})$. Given I_t, independent samples can be obtained from the conditional distribution $P(I_{t+1}, ..., I_{t+d} \mid I_t, \mathcal{T})$ by simply sampling according to $\mathcal{T}$ from the initial state I_t. Ignoring $I_{t+1}, ..., I_{t+d-1}$ and checking F in I_{t+d} yields independent samples of the boolean event $I_{t+d} \models_B F$ from the distribution $P(I_{t+d} \models_B F \mid I_t, \mathcal{T})$. The proportion of positive samples of this variable will thus quickly approach the true probability $P(I_{t+d} \models_B F \mid I_t, \mathcal{T})$.

6.3 Experimental Evaluation

This section presents a preliminary evaluation of the proposed CPT-L system. The main goal of the evaluation is to prove that CPT-L can handle sequences of interpretations of reasonable complexity, and scales well to the number of objects present in the domain. More specifically, we seek to answer the following two questions:

(Q6.1) Are the learning and inference techniques proposed for CPT-L in Section 6.2 effective and efficient?

(Q6.2) Can the CPT-L model capture the relational structure in a real-world domain?

To answer questions **(Q6.1)** and **(Q6.2)**, CPT-L has been evaluated in two domains. First, we discuss experiments in a stochastic version of the well-known *blocks world* domain, an artificial domain that enables us to perform controlled and systematic experiments, for example, with regard to the scaling behavior of the proposed algorithms. Second, the model is evaluated on real-world data collected from a live server of a massively multi-player online strategy game. Experiments in these two domains are now presented in turn.

6.3.1 Experiments in a Stochastic Blocks World Domain

As an artificial test bed for CPT-L, we performed experiments in a stochastic version of the well-known *blocks world* domain. The domain was chosen because it is truly relational and also serves as a popular artificial world model in agent-based approaches such as planning and reinforcement learning. Application scenarios involving agents that act and learn in an environment are one of the main motivations for CPT-L. In such scenarios world-transition dynamics typically stem from *actions* carried out by the agents according to some *policy*. In the blocks-world domain discussed in this section, we assume that the policy of the agent is known and the task is to probabilistically model transition dynamics given the policy. It is straightforward to represent such conditional world models in CPT-theories by including the policy as part of the background knowledge.

World Model

The blocks world we consider consists of a table and a number of blocks. Every block rests on exactly one other block or the table, denoted by a fact $on(A, B)$. Blocks come in different sizes, denoted by $size\_of(A, N)$ with $N \in \{1, ..., 4\}$. A predicate $free(B) \leftarrow not(on(A, B))$ is defined in the background knowledge. Additionally, a background predicate $stack(A, S)$ defines that block A is part of a stack of blocks, which is represented by its lowest block S. Actions derived from the policy are of the form $move(A, B)$. If both A and B are free, the action moves block A on B with probability $1 - \epsilon$, with probability ϵ the world state does not change. Furthermore, a stack S can start to jiggle, represented by $jiggle(S)$. A stack can start to jiggle if its top block is lifted, or a new block is added to it. Furthermore, stacks can start jiggling without interference from the agent, which is more likely if they contain many blocks and large blocks are stacked on top of smaller ones. Stacks that jiggle collapse in the next time-step, and all their blocks fall on the table. Two example rules from this domain are

$$(jiggle(S) : 0.2) \vee (nil : 0.8) \leftarrow move(A, B), stack(A, S)$$

$$(jiggle(S) : 0.2) \vee (nil : 0.8) \leftarrow move(A, B), stack(B, S),$$

they describe that stacks can start to jiggle if blocks are added to or taken from a stack. Furthermore, we consider a simple policy that tries to build a large stack of blocks by repeatedly stacking the free block with second-lowest ID on the free block with lowest ID. This strategy would result in one large stack of blocks if the stack never collapsed.

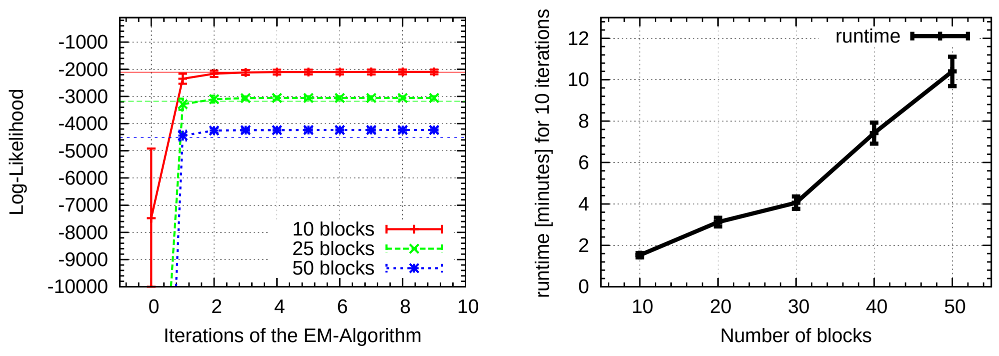


Figure 6.2: Left graph: per-sequence log-likelihood on the training data as a function of the EM iteration. Right graph: Running time of EM as a function of the number of blocks in the world model.

Results in the Blocks-World Domain

In a first experiment, we explore the convergence behavior of the EM algorithm for CPT-L. The world model together with the policy for the agent, which specifies which block to stack next, is implemented by a (gold-standard) CPT-theory $\mathcal{T}$, and a training set of 20 sequences of length 50 each is sampled from $\mathcal{T}$. From this data, the parameters are re-learned using EM. Figure 6.2, left graph, shows the convergence behavior of the algorithm on the training data for different numbers of blocks in the domain, averaged over 15 runs. It shows rapid and reliable convergence. Figure 6.2, right graph, shows the running time of EM as a function of the number of blocks. The scaling behavior is roughly linear, indicating that the model scales well to reasonably large domains. Absolute running times are also low, with about 1 minute for an EM iteration in a world with 50 blocks[2]. This is in contrast to other, more expressive modeling techniques which typically scale badly to domains with many objects. The theory learned (Figure 6.2) is very close to the ground truth ("gold standard model") from which training sequences were generated. On an independent test set (also sampled from the ground truth), log-likelihood for the gold standard model is -4510.7, for the learned model it is -4513.8, while for a theory with randomly initialized parameters it is -55999.4 (50 blocks setting). Manual inspection of the learned model also shows that parameter values are on average very close to those in the gold-standard model.

The experiments presented so far show that relational stochastic domains of substantial size can be represented in CPT-L. The presented algorithms for inference and learning are effective and efficient, and scale well with the size of the domain. This affirmatively

[2]All experiments were run on standard PC hardware, 2.4GHz Intel Core 2 Duo processor, 1GB memory.

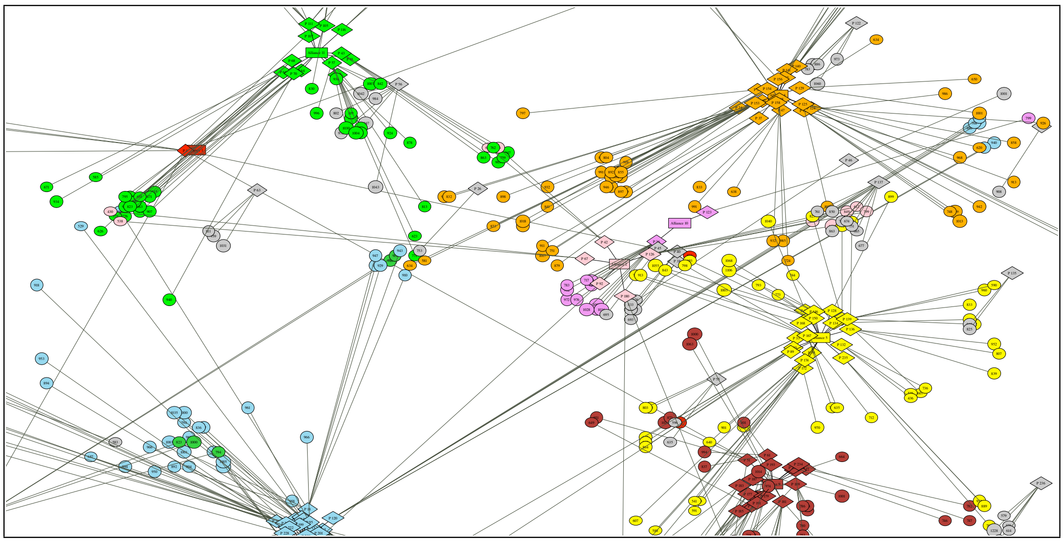


Figure 6.3: High-level view of a (partial) game world in Travian. Circular nodes indicate cities, shown in their true positions on the game's grid-map. Diamond-shaped nodes indicate players, and are connected to all cities currently owned by the player. Rectangular nodes indicate alliances, and are connected to all players currently members of the alliance. Moreover, players and cities are color-coded according to their alliance affiliation.

answers Question **(Q6.1)**.

6.3.2 Experiments in a Massively Multi-player Online Game

We consider the MMOG *Travian*, a commercial, large-scale strategy game with a player community of about 3.000.000 players worldwide[3]. In Travian, players are spread over several independent *game worlds*, with approximately 20.000–30.000 players interacting in a single world. Travian game play follows a classical strategy game setup. A game world consists of a large *grid-map*, and each player starts with a single *city* located on a particular tile of the map. During the course of the game, players harvest *resources* from the environment, improve their cities by construction of *buildings* or research of *technologies*, or found new cities on other (free) tiles of the map. Additionally, players can build different military units which can be used to attack and conquer other cities on the map, or trade resources on a global marketplace.

In addition to these low-level game play elements, there are high-level aspects of game play involving *multiple players*, which need to cooperate and coordinate their playing to achieve otherwise unattainable game goals. More specifically, in Travian players dynamically organize themselves into *alliances*, for the purpose of jointly attacking and defend-

[3]`www.travian.com`; `www.traviangames.com`

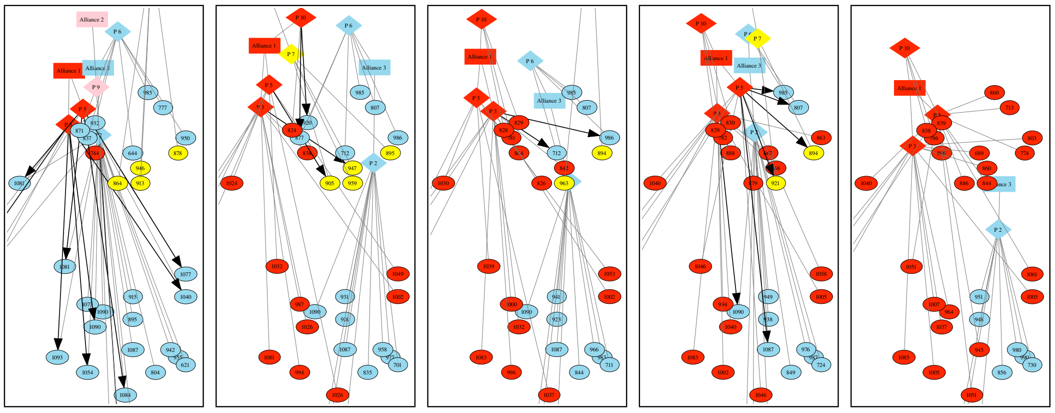


Figure 6.4: Travian game dynamics visualized as changes in the game graph (for $t = 1, 2, 3, 4, 5$). Bold arrows indicate conquest attacks by a player on a particular city.

ing, trading resources or giving advice to inexperienced players. Such alliances constitute social networks for the players involved, where diplomacy is used to settle conflicts of interests and players compete for an influential role in the alliance. In the following, we will take a high-level view of the game and focus on modeling player interaction and co-operation in alliances rather than low-level game elements such as resources, troops and buildings. Figure 6.3 shows such a high-level view of a (partial) Travian game world, represented as a graph structure relating cities, players and alliances which we will refer to as a *game graph*. It shows that players in one alliance are typically concentrated in one area of the map—traveling over the map takes time, and thus there is little interaction between players far away from each other.

We are interested in the *dynamic* aspect of this world: as players are acting in the game environment (for example, by conquering other players' cities and joining or leaving alliances), the game graph will continuously change, and thereby reflect changes in the social network structure of the game. As an example for such transition dynamics, consider the sequence of game graphs shown in Figure 6.4. Here, three players from the red alliance launch a concerted attack against territory currently held by the yellow and blue alliances, and partially conquer it.

From a machine learning perspective, domains such as Travian pose three main challenges: 1) world state descriptions are inherently relational, as the interaction between (groups of) agents is of central interest, 2) the transition behavior of the world is strongly stochastic, and 3) a relatively large number of objects and relations is needed to build meaningful models, as the defining element of environments such as MMOGs are interactions among *large* sets of agents. Thus, we need an approach that is both computationally efficient and able to represent complex relational state descriptions and stochastic world dynamics.

Data Collection and Preprocessing

The data used in the experiments was collected from a "live" Travian server with approximately 25.000 active players. Over a period of three months (December 2007, January 2008, February 2008), high-level data about the current state of the game world was collected once every 24 hours. This included information about all cities, players, and the alliance structure in the game. For cities, their size and position on the map are available; for players, the list of cities they own; and for alliances the list of players currently affiliated with that alliance. From all available data, we extracted 30 sequences of local game world states. Each sequence involves a subset of 10 players, which are tracked over a period of one month (10 sequences each for December, January and February). Player sets are chosen such that there are no interactions between players in different sets, but a high number of interactions between players within one set. Cities that did not take part in any conquest event were removed from the data, leaving approximately 30–40 cities under consideration for every player subset.

World Model

The game data was represented using predicates $city(C, X, Y, S, P)$ (city C of size S at coordinates X, Y held by player P), $allied(P, A)$ (player P is a member of alliance A), $conq(P, C)$ (indicating a conquest attack of player P on city C) and $alliance\_change(P, A)$ (player P changes affiliation to alliance A). A predicate $distance(C_1, C_2, D)$ with $D \in \{near, medium, far\}$ computing the (discretized) distance between cities was defined in the background knowledge. The final state descriptions (game graphs) on average contain approximately 50 objects (nodes) at every step in time, and relations between them. Sequences consists of between 29 and 31 such state descriptions.

We defined a world model in CPT-L that expresses the probability for player actions such as conquests of cities and changes in alliances affiliation, and updates the world state accordingly. Player actions in Travian—although strongly stochastic—are typically explainable from the social context of the game: different players from the same alliance jointly attack a certain territory on the map, there are retaliation attacks at the alliance level, or players leave alliances that have lost many cities in a short period of time. From a causal perspective, actions are thus triggered by certain (relational) patterns that hold in the game graph, which take into account a player's alliance affiliation together with the actions carried out by other alliance members. Such patterns can be naturally expressed in CPT-L as bodies of rules which trigger actions encoded in the head of the rule. We manually defined a number of simple rules capturing such typical game patterns. As an example, consider the rules

$$\begin{aligned}
&conq(P, C) : 0.039 \vee nil : 0.961 \quad \leftarrow \\
&\qquad conq(P, C'), city(C', \_, \_, \_, P'), city(C, \_, \_, \_, P') \\
&conq(P, C) : 0.011 \vee nil : 0.989 \quad \leftarrow \\
&\qquad city(C, \_, \_, \_, P''), allied(P, A), allied(P', A), conq(P', C'), city(C', \_, \_, \_, P'')
\end{aligned}$$

The first rule encodes that a player is likely to conquest a city of a player he already attacked in the previous time-step. The second rule generalizes this pattern: a player P is likely to attack a city C of player P'' if an allied player has attacked P'' in the previous time-step.

Moreover, the world state needs to be updated given the players' actions. After a conquest attack $conq(P, C)$, the city C changes ownership to player P in the next time-step. If several players execute conquest attacks against the same city in one time-step, one of them is chosen as the new owner of the city with uniform probability (note that such simultaneous conquest attacks would not be observed in the training data, as only one snapshot of the world is taken every 24 hours). Similarly, an $alliance\_change(P, A)$ event changes the alliance affiliation of player P to alliance A in the next time-step.

Results in the Travian Domain

We consider the task of predicting the "conquest" action $conq(P, C)$ based on a learned generative model of world dynamics. The collected sequences of (local) game states were split into one training set (sequences collected in December 2007) and two test sets (sequences collected in January 2008 and sequences collected in February 2008). Maximum-likelihood parameters of a hand-crafted CPT-theory $\mathcal{T}$ as described above were learned on the training set using EM. Afterwards, the learned model was used to predict the player action $conq(P, C)$ on the test data in the following way. Let S denote a test sequence with states $I_0, ..., I_T$. For every $t_0 \in \{0, ..., T-1\}$, and every player p and city c occurring in S, the learned model is used to compute the probability that the conquest event $conq(p, c)$ will be observed in the next world state, $P(I_{t_0+1} \models conq(p, c) \mid \mathcal{T}, I_0, ..., I_{t_0})$. This probability is obtained from the sampling-based prediction algorithm described in Section 6.2. The prediction is compared to the known ground truth (whether the conquest event occurred at that time in the game or not). Instead of predicting whether the player action will be taken in the next step, we can also predict whether it will be taken within the next k steps, by computing

$$P(I_{t_0+1} \models conq(p, c) \vee ... \vee I_{t_0+k} \models conq(p, c) \mid \mathcal{T}, I_0, ..., I_{t_0}).$$

This quantity is also easily obtained from the prediction algorithm for CPT-L. Figure 6.5, left, shows ROC curves for this experiment with different values $k \in \{1, 2, 3, 4, 5\}$, evaluated on the first test set (January 2008). Figure 6.5, right, shows the corresponding AUC values as a function of k for both test sets. The achieved area under the ROC curve is substantially above 0.5 (random performance), indicating that the learned CPT-theory $\mathcal{T}$ indeed captures some characteristics of player behavior and obtains a reasonable ranking of player/city pairs (p/c) according to the probability that p will conquer c. Moreover, the model is able to predict conquest actions several steps in the future, although AUC is slightly lower for larger k. This indicates that uncertainty associated with predictions accumulates over time. Finally, predictions for the first test set (January 2008) are slightly more accurate than for the second test set (February 2008). This is not surprising as

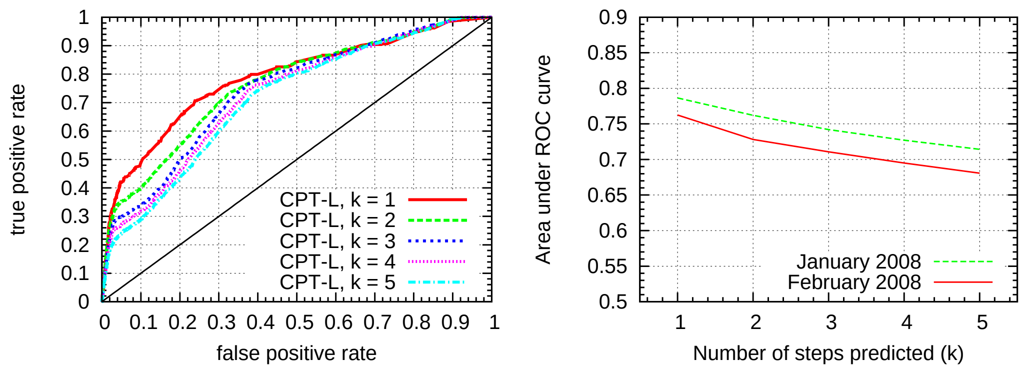


Figure 6.5: Left figure: ROC curve for predicting that a city C will be conquered by a player P within the next k time-steps, for $k \in \{1, 2, 3, 4, 5\}$. The model was trained on 10 sequences of local game state descriptions from December 2007, and tested on 10 sequences from January 2008. Right figure: AUC as a function of the number k of future time-steps considered in the same experiment. Additionally, AUC as a function of k is shown for 10 test sequences from February 2008.

the model has been trained from sequences collected in December 2007, and indicates a slight change in game dynamics over time. In summary, we conclude that player actions in Travian are indeed to some degree predictable from the social context of the game, and CPT-L is able to learn such patterns from the data. This indicates an affirmative answer to Question **(Q6.2)**.

Parameter learning for the CPT-L theory $\mathcal{T}$ on the training set takes approximately 30 minutes, and the model needed 5 iterations of EM to converge. Predicting the probability of $conq(p, c)$ for all player/city pairs and the next k time-steps starting from a particular world state takes approximately 1 minute.

6.4 Related and Future Work

There are relatively few existing approaches that can probabilistically model sequences of relational state descriptions. CPT-L can be positioned with respect to them as follows. First, statistical relational learning systems such as Markov Logic (Richardson and Domingos, 2006), CP-logic (Vennekens *et al.*, 2006), Probabilistic Relational Models (Getoor *et al.*, 2001) or Bayesian Logic Programs (Kersting *et al.*, 2006) can be used in this setting by adding an extra time argument to predicates (then called *fluents*). However, inference and learning in these systems is computationally expensive: they support very general models including hidden states, and are not optimized for sequential data. A second class of techniques, for instance (Zettlemoyer *et al.*, 2005), uses transition models based on (stochastic) STRIPS rules. This somewhat limits the transitions that can be expressed, as only one rule "fires" at every point in time, and it is difficult to model several processes that change the state of the world concurrently (such as an agent's actions and naturally occurring world changes). In contrast, such scenarios are naturally modeled in CP-logic and thus CPT-L. Another approach designed to model sequences of relational state descriptions are relational simple-transition models (Fern, 2005). In contrast to CPT-L, they focus on domains where the process generating the data is hidden, and inferring these hidden states from observations. This is a harder setting than the fully observable setting discussed here, and typically only approximate inference is possible (Fern, 2005).

Finally, there is also the recent work on aligning sequences of interpretations by Karwath *et al.* (2008). While alignments do not provide a generative model for such sequences, they can be used for tasks such as sequence classification, by comparing alignments to positive and negative training sequences.

CPT-L can be seen as the simplest extension of the Markov model framework to sequences of interpretations. The main direction for future work is to further evaluate the trade-off between representational power and scaling behavior in the proposed framework, and to explore how the model can be extended without sacrificing efficiency. One possible extension concerns the modeling of hidden states, that is, facts in an interpretations that are not observable in the given training and test sequences. Another possible extension would be to consider (logical) constraints on the interpretations that are allowed to be sampled from the model. Note that such constraints, by rendering some interpretations invalid, lead to the loss of probability mass. Thus, the model would have to be accordingly normalized. We are currently exploring how this can be done efficiently, using similar dynamic programming approaches as those used for inference and learning in CPT-L today.

Conclusions Part II

Part II of the thesis has discussed statistical models for sequences of logical atoms (in Chapter 5) and complete logical interpretations (in Chapter 6). Compared to modeling traditional sequences over a finite alphabet of fixed symbols, the relational setting is significantly harder, as the number of possible sequence elements is exponential in the size of the logical alphabet considered (or even infinite in the presence of functors). Consequently, traditional approaches to sequence modeling cannot directly be applied in the relational setting, as they would suffer from data sparseness and efficiency problems.

Relational modeling approaches control this complexity by relational abstraction and possibly further restrictions on the probability distributions that can be represented. The two approaches presented in Chapter 5 and Chapter 6 are based on the well-known Markov model framework. Markov models employ two simplifying assumptions: that data is fully observable and that the future is independent of the past given the present (Markov assumption). These assumptions significantly reduce complexity compared to more expressive approaches such as hidden Markov models and dynamic Bayesian networks.

Specifically, Chapter 5 has introduced r-grams, which extend n-grams to the relational case. n-grams extend traditional Markov models by mixing Markov chains of different order. This increases expressivity compared to a simple first-order Markov chain, but still allows n-grams to be applied to large-scale application domains (for instance, natural language corpora with hundreds of thousands of words (Manning and Schütze, 1999)). In r-grams, distributions can be smoothed both by lowering the order of the gram and by relational generalization. Furthermore, relational generalization enables us to abstract away from identifiers appearing in the data, further reducing model complexity.

Chapter 6 has introduced the CPT-L model for the more general setting of modeling sequences of complete logical interpretations. CPT-L is again based on the Markov model framework. However, in this more general setting, the Markov process transitions from the interpretation at time t—a *set* of logical atoms—to a new interpretation at time $t + 1$. To model such transitions between sets of atoms, CPT-L employs CP-logic, an expressive probabilistic causal logic (Vennekens *et al.*, 2006). However, by restricting CP-logic to fully observable sequential data, the resulting inference and learning problems in CPT-L are more tractable than for general CP-logic. We showed how these problems can be solved efficiently using dynamic programming in data structures based on binary decision diagrams.

To summarize, this part of the thesis has shown that modeling sequences of relational data items is an interesting problem setting that is relevant in various application domains. At the same time, the combinatorial explosion in possible sequence elements renders traditional, propositional techniques inapplicable. We have contributed two simple statistical relational modeling techniques for relational sequences that deliberately restrict expressivity compared to more general SRL modeling techniques, and thereby maintain computational efficiency. This efficiency is essential in many real-life application domains.

The approaches discussed in this part assume that training data is fully observable. While this assumption makes inference and learning easier, it is not always realistic. An interesting direction for future work is thus to extend the presented approaches towards models for partially observable data.

Part III

Embedded SRL: Application-Specific Approaches

Outline Part III

Statistical relational learning aims at creating general and expressive modeling frameworks that are applicable in a wide variety of application domains. Such frameworks are useful as they enable us to employ principled and well-developed systems quickly in new application areas, without having to re-develop a new system for every new domain encountered. However, in most real-world applications we are only interested in solving relatively specific inference and learning problems. This part of the thesis discusses "downgrading" general SRL techniques to an application-specific, more restricted problem setting. The resulting systems will be less expressive and general than their underlying formalism, but typically more efficient and effective at solving the particular tasks encountered in the application domain.

The domains we consider are structured in a way that is not easy to capture using standard propositional modeling techniques. Even if this structure is not explicitly relational, it is typically easy to model in a relational formalism. Statistical relational frameworks can thus serve as a guideline and motivation for developing more application-specific structured statistical models. We will refer to such models as *embedded SRL*, as they embed more general SRL frameworks into specific application domains. Embedded SRL can be seen as one way of improving efficiency at the cost of generality and expressivity.

Part III of the thesis discusses two such embedded SRL models, namely for the *haplotyping* and *activity recognition* domains. Both models are based on the hidden Markov model (HMM) framework discussed in Section 2.1.2. The structure of the models can be represented using, for instance, logical hidden Markov models (LoHMMs), a framework upgrading traditional HMMs to sequences of atoms (Kersting *et al.*, 2006). However, only a small part of the representational power of LoHMMs is needed in these applications, and it is thus easier to directly specify structured probabilistic models which are not explicitly relational.

Chapter 7 discusses a structured probabilistic model for the haplotyping application. Haplotyping is the task of inferring the phase in diploid genotype measurements, and constitutes an important intermediate step in so-called gene mapping studies that seek to uncover the genetic basis of complex diseases. We begin by defining the problem of population-based haplotype reconstruction as a statistical inference task. Afterwards, a basic structured HMM for haplotype reconstruction is introduced. Expressivity can be increased by extension to a higher-order model, however, as for standard (hidden) Markov models, this entails a combinatorial explosion in the number of parameters. We introduce sparse higher-order models that only include evolutionarily preserved haplotype fragments, which greatly reduces complexity. An application-specific structure learning algorithm is presented to learn such sparse models. An empirical evaluation on artificial and real haplotype data shows that the resulting haplotyping system is competitive with state-of-the art approaches developed in the bioinformatics community.

Chapter 8 discusses a structured probabilistic model for the activity recognition problem. In the domain we consider, the goal is to infer the activity of a user from a stream of RFID sensor observations indicating the sequence of objects a user has been interacting with. The activities under consideration are hierarchically structured: an activity consist of several substeps that are typically executed in a certain order. Moreover, users typically interleave activities in time. We present a model which explicitly takes this domain structure into account. Specifically, we derive the simplest extension of the hidden Markov model framework that takes into account several interleaved hidden processes—the activities—which together generate the observable sequence. Unfortunately, exact inference even in this simple extension is NP-hard. We develop a problem-specific approximate inference algorithm that is tractable and yields near-optimal results in the domain under consideration. Empirical results show that the resulting activity recognition system is superior to a standard (unstructured) hidden Markov model approach.

Chapter 7

A Structured Hidden Markov Model for Haplotype Analysis*

This chapter is concerned with the problem of *haplotype reconstruction*. Haplotype reconstruction is an important intermediate step in gene association studies, and collecting and analyzing haplotype data has received much attention recently (The International HapMap Consortium, 2005). We propose a structured probabilistic model for haplotype reconstruction based on the hidden Markov model framework (see Section 2.1.2), and introduce a domain-specific structure learning algorithm to identify conserved haplotype fragments in the population under consideration.

The data encountered in haplotype analysis consists of sequences of pairs. Such sequences are not easily represented using a propositional alphabet of flat symbols, as this would conceal the underlying inner structure in a pair. However, sequences of pairs are easily represented as sequences of logical atoms, and thus haplotyping is subsumed by the setting of learning from sequences of logical atoms. In this sense, haplotyping can be seen as an (extremely restricted) instance of sequential statistical relational learning. The model we present in this chapter will not be explicitly relational, but rather tailored to the setting of modeling sequences of pairs. However, it is a special case of a logical hidden Markov model (LoHMM), a general sequential SRL framework extending HMMs to sequences of logical atoms (Kersting *et al.*, 2006). While standard LoHMMs could be applied to haplotyping directly, the domain-specific model we develop in this chapter can be implemented more efficiently. Moreover, we present a domain-specific structure learning algorithm for our model, as the general-purpose LoHMM structure learner is clearly not applicable.

The chapter is organized as follows. We begin with an introduction to haplotyping and some relevant biological background. Section 7.2 describes our domain-specific model

*This chapter builds on (Landwehr *et al.*, 2006a, 2007a; Landwehr and Mielikäinen, 2008).

for the haplotyping problem. Section 7.3 discusses structure learning in the proposed framework. Finally, an empirical evaluation of the proposed method on real-world haplotype data is presented in Section 7.4.

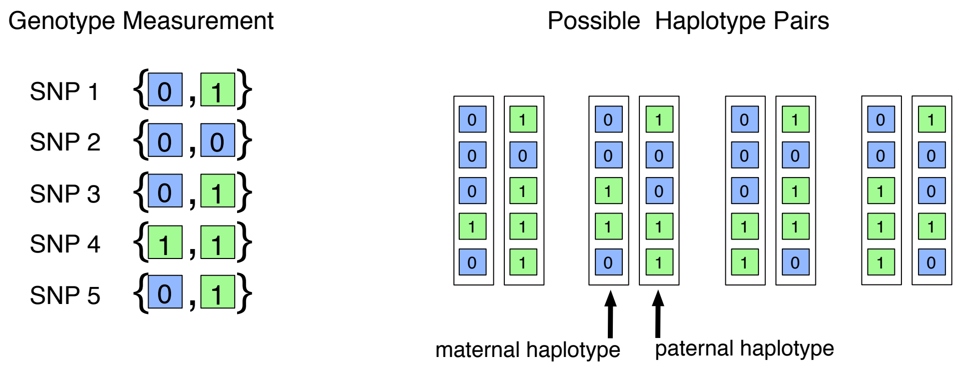
7.1 Population-based Haplotype Reconstruction

Analysis of genetic variation in human populations is critical to the understanding of the genetic basis for complex diseases. Most studied differences in DNA are single-nucleotide variations at particular positions in the genome, which are called *single nucleotide polymorphisms* (SNPs). The positions are also called *markers* and the different possible values *alleles*. A *haplotype* is a sequence of SNP alleles along a region of a chromosome, and concisely represents the (variable) genetic information in that region. In the search for DNA sequence variants that are related to common diseases (so-called *gene mapping* studies), haplotype-based approaches have become a central theme (The International HapMap Consortium, 2005). This is because knowledge of the haplotype structure allows the selection of relatively few "tag" SNPs to distinguish sequence variants, which can improve efficiency of genome-wide association studies.

In diploid organisms such as humans there are two *homologous* (almost identical) copies of each chromosome, namely the maternal and the paternal chromosome. Accordingly, there are two haplotypes. As it does not matter whether a haplotype has been inherited from the mother or the father, the haplotype information for an individual is an unordered pair. Current practical laboratory measurement techniques produce a *genotype*—for m markers, a sequence of m unordered pairs of alleles. The genotype reveals which two alleles are present at each marker, but not their respective chromosomal origin. In order to obtain haplotypes from genotype data, the hidden order of the two alleles needs to be reconstructed. This hidden order is also called the *phase* information. That is, we need to obtain an unordered pair of sequences (the haplotypes) from a sequence of unordered pairs (the genotypes) by assigning alleles to the respective haplotype.

More formally, a haplotype h is a sequence of alleles in markers $i = 1, \ldots, m$, and will be denoted by $h = h[1], ..., h[m]$. In most cases, only two alternative alleles occur at an SNP marker, so we can assume that $h \in \{0,1\}^m$. A genotype g is a sequence of unordered pairs $g[i] = \{h_g^1[i], h_g^2[i]\}$ of alleles in markers $i = 1, \ldots, m$. Hence, $g \in \{\{0,0\}, \{1,1\}, \{0,1\}\}^m$ where we represent unordered pairs as multisets. A marker with alleles $\{0,0\}$ or $\{1,1\}$ is *homozygous* whereas a marker with alleles $\{0,1\}$ is *heterozygous*.

Example 7.1.1. *Consider the genotype information depicted below (left part of figure). There are five SNP markers, each with two possible alleles* $0, 1$. *Markers two and four are measured as homozygous, the other markers as heterozygous. Note that the genotype information is a sequence of unordered pairs of alleles. There are four possible pairs of haplotypes corresponding to this genotype observation (right part of figure).*



Note that it is not directly observable which of these is the true underlying haplotype pair for the given genotype observation. The problem of haplotype reconstruction *is concerned with finding this true haplotype pair for a given genotype.*

There are two alternative approaches for obtaining haplotype pairs from genotypes. If family trios are available (that is, genotype measurements from each individual and its two parents), most of the ambiguity in the phase can be resolved analytically. If no family trio information is available, population-based computational methods have to be used to estimate the haplotype pair for each genotype. They basically try to infer the *most likely* haplotype pairs based on statistical models of the haplotypes in a population. Because trios are more difficult to recruit and more expensive to genotype, population-based approaches are often the only cost-effective method for large-scale studies. Consequently, the study of such techniques has received much attention (see Salem *et al.*, 2005; Halldórsson *et al.*, 2004, for comprehensive literature reviews on population-based haplotyping approaches).

Given only one genotype observation, there is in general no argument for preferring any of the $2^{m'-1}$ possible haplotype pairs as the "most likely" one (where m' is the number of heterozygous markers). However, if a set of genotype observations for different individuals in a population is given, it is possible to indirectly identify "likely" haplotypes in the population from the genotype observations and use this information to solve the haplotype reconstruction problem. Formally, this problem can be defined as follows:

Problem 7.1.1 (Population-based Haplotype Reconstruction)**.**

Given

- *a set of SNP markers* $\{M_1, ..., M_m\}$*;*
- *a multiset* $\mathcal{G}$ *of genotypes in markers* $M_1, ..., M_m$*;*

Find *a mapping* $\mathcal{H}^* : \mathcal{G} \rightarrow \{0,1\}^m \times \{0,1\}^m$ *associating each* $g \in \mathcal{G}$ *with a pair*

$\langle h_g^1, h_g^2 \rangle$ *of haplotypes such that*

$$\begin{aligned}\mathcal{H}^* &= \underset{\mathcal{H}\ \textit{consistent}}{\arg\max}\ P(\mathcal{H} \mid \mathcal{G}) \\ &= \underset{\mathcal{H}\ \textit{consistent}}{\arg\max} \prod_{g\in\mathcal{G}} P(\langle h_g^1, h_g^2 \rangle \mid \mathcal{G})\end{aligned}$$

where a reconstruction $\mathcal{H}$ is consistent iff

$$\forall g \in \mathcal{G}, \forall i = 1, \ldots, m : g[i] = \{h_g^1[i], h_g^2[i]\}$$

and we have assumed that genotypes are sampled independently from identical distributions. It is also usually assumed that the sample $\mathcal{G}$ is in Hardy-Weinberg equilibrium, meaning that

$$P(\langle h_g^1, h_g^2 \rangle \mid \mathcal{G}) = P(h_g^1 \mid \mathcal{G}) P(h_g^2 \mid \mathcal{G})$$

for a distribution $P(h \mid \mathcal{G})$ over haplotypes.

The key to haplotype reconstruction is thus to accurately estimate the distribution $P(H \mid \mathcal{G})$ over haplotypes h, given the available genotype information $\mathcal{G}$. This can be achieved as follows. We will define a structured probabilistic model λ for the distribution $P(G, H_g^1, H_g^2)$ over genotype observations g and underlying haplotypes h_g^1, h_g^2 that explicitly takes into account the domain structure given by haplotype-genotype consistency and the assumption of Hardy-Weinberg equilibrium. Note that only the genotype g will be observable, while the corresponding haplotype pair h_g^1, h_g^2 is hidden. The model λ can be learned from the available genotype observations $\mathcal{G}$. Afterwards, the haplotype reconstruction $\langle h_g^1, h_g^2 \rangle$ of a genotype $g \in \mathcal{G}$ can be obtained by a hidden state inference:

$$\begin{aligned}\langle h_g^1, h_g^2 \rangle &= \underset{\langle h_1, h_2 \rangle}{\arg\max}\ P_\lambda(h^1, h^2 \mid g) \\ &= \underset{\langle h_1, h_2 \rangle}{\arg\max}\ \frac{P_\lambda(g, h^1, h^2)}{\sum_{\bar{h}^1, \bar{h}^2} P\lambda(\bar{h}^1, \bar{h}^2, g)} \qquad (7.1)\end{aligned}$$

where P_λ is the probability estimate given by the learned λ (that is, based on $\mathcal{G}$).

The genetic variation in SNPs is mostly due to two causes: *mutation* and *recombination*. Mutations are relatively rare, they occur with a frequency of about 10^{-8}. While SNPs are themselves results of ancient mutations, mutations are usually ignored in statistical haplotype models due to their rarity. Recombination introduces variability by breaking up the chromosomes of the two parents and reconnecting the resulting segments to form a new and different chromosome for the offspring. Because the probability of a recombination event between two markers is lower if they are near to each other, there is a statistical correlation (so-called *linkage disequilibrium*) between markers which decreases with increasing marker distance. Statistical approaches to haplotype modeling are based on exploiting such patterns of correlation.

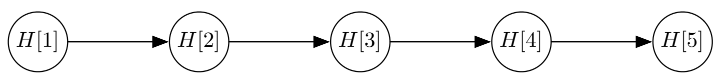


Figure 7.1: A left-to-right Markov model over haplotypes (for $m = 5$). Random variables $H[i]$ represent the allele at marker $i \in \{1, ..., m\}$.

7.2 A Basic Model for Haplotype Reconstruction

We first discuss modeling the probability distribution over haplotypes, that is, we define a model μ for a distribution $P(H)$. This will later be extended to a model λ for the joint distribution over haplotypes and genotypes, that is, for $P(G, H_g^1, H_g^2)$. Motivated by the observation that linkage disequilibrium is stronger between markers that are close together, we model the distribution over haplotypes by a left-to-right Markov model. It defines a chain of random variables $H[1], ..., H[m]$, where a random variable $H[i]$ represents the allele $h[i] \in \{0, 1\}$ at marker $i \in \{1, ..., m\}$[1]. The defined distribution is

$$\begin{aligned} P_\mu(H) &= P_\mu(H[1], ..., H[m]) \\ &= P_\mu(H[1]) \prod_{i=2}^{m} P_\mu(H[i] \mid H[i-1]) \end{aligned}$$

where the distributions $P_\mu(H[1])$ and $P_\mu(H[i] \mid H[i-1])$ will be estimated from data. The model is depicted in Figure 7.1 (for $m = 5$) in standard graphical model notation, confer Section 2.1.1. At this point, an important distinction needs to made compared to the definition of a *stationary* Markov process made in Chapter 2 (Def. 2.1.5). Stationary Markov processes model the development of a process over time, and we can naturally assume that transition probabilities do not depend on the current time step, so $P(X_{t+1} \mid X_t) = P(X_{t+i+1} \mid X_{t+i})$. In haplotyping, this assumption is not appropriate, because patterns of linkage disequilibrium will be different for different marker positions. Thus, $P(h[t] \mid h[t-1]) \neq P(h[t+i] \mid h[t+i-1])$ in general.

The model μ is not directly applicable in haplotype reconstruction, because in reality only genotypes are observed whereas the phase information is hidden. We propose the following structured probabilistic model for $P(G, H_g^1, H_g^2)$, taking into account the relationship between haplotypes and genotypes and the assumption of Hardy-Weinberg equilibrium. The model (depicted in Figure 7.2 for $m = 5$) consists of three chains of variables. The chain $H_g^1[1], ..., H_g^1[m]$ represents the allele values along h_g^1, $H_g^2[1], ..., H_g^2[m]$ represents the allele values along h_g^2, and $G[1], ..., G[m]$ represents the unordered allele pairs corresponding to the genotype observations. Note that for the corresponding values $h_g^1[i], h_g^2[i] \in \{0, 1\}$, while $g[i] \in \{\{0, 0\}, \{0, 1\}, \{1, 1\}\}$ is a multiset corresponding to

[1] as in Section 2.1, we use upper-case letters to denote random variables and lower-case letter to denote their values.

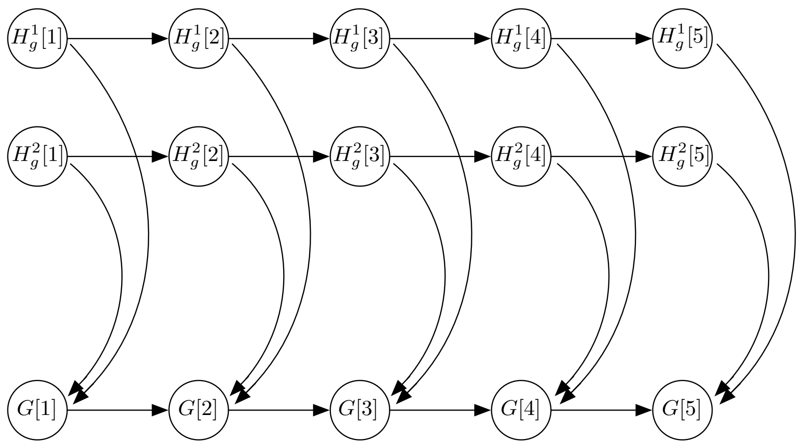


Figure 7.2: A structured probabilistic model over haplotypes and genotypes.

an unordered pair. The defined distribution is

$$\begin{aligned}P_\lambda(G, H_g^1, H_g^2) &= P_\lambda(G[1], ..., G[m], H_g^1[1], ..., H_g^1[m], H_g^2[1], ..., H_g^2[m]) \\ &= P_\lambda(H_g^1[1])P_\lambda(H_g^2[1])P_\lambda(G[1] \mid H_g^1[1], H_g^2[1]) \\ &\prod_{i=2}^{m} P_\lambda(H_g^1[i] \mid H_g^1[i-1])P_\lambda(H_g^2[i] \mid H_g^2[i-1])P_\lambda(G[i] \mid H_g^1[i], H_g^2[i]).\end{aligned}$$

We now discuss how the domain structure is incorporated into the proposed model. First, the genotype observation—an unordered pair of the form $\{0,0\}$,$\{0,1\}$, or $\{1,1\}$—is a deterministic function of the two haplotype alleles:

$$P(g[i] \mid h_g^1[i], h_g^2[i]) = \begin{cases} 1 : g[i] = \{h_g^1[1], h_g^2[1]\} \\ 0 : \text{otherwise.} \end{cases} \tag{7.2}$$

Thus, the distribution $P_\lambda(G[i] \mid H_g^1[i], H_g^2[i])$ is fixed to the deterministic function given by Equation (7.2). Furthermore, we need to account for the Hardy-Weinberg equilibrium assumption. This means that h_g^1, h_g^2 are sampled independently from one distribution $P(H)$ over haplotypes. In fact, this distribution is naturally given by the model μ outlined above. Thus, we define

$$P_\lambda(h_g^1[1]) = P_\lambda(h_g^2[1]) = P_\mu(h[1]) \tag{7.3}$$

and

$$P_\lambda(h_g^1[i] \mid h_g^1[i-1]) = P_\lambda(h_g^2[i] \mid h_g^2[i-1]) = P_\mu(h[i] \mid h[i-1]) \tag{7.4}$$

for $i \in \{2, ..., m\}$. In other words, parameters for the chains $H_g^1[1], ..., H_g^1[m]$ and $H_g^2[1], ..., H_g^2[m]$ are enforced to be identical ("parameter tying"). To summarize, all parameters of λ have been re-expressed as parameters of the model μ on haplotypes outlined above.

Note that the model depicted in Figure 7.2 can be seen as a structured extension of the hidden Markov model framework discussed in Section 2.1.2: haplotypes correspond to hidden states and genotypes to observations. The model is also closely related to the *factorial hidden Markov model* (fHMM) introduced by Ghahramani and Jordan (1997). The differences are that 1) transition distributions in our model are tied for the two chains representing the haplotypes, while in fHMMs chains have independent transition distributions; 2) parameters in an fHMM are tied over time (stationarity); and 3) in our model the distribution $P(G[i] \mid H_g^1[i], H_g^2[i])$ takes the special form given by Equation (7.2).

Let us now return to the haplotype reconstruction problem (Problem 7.1.1). As outlined above, the task is to first estimate the probability distribution $P(H \mid \mathcal{G})$ over haplotypes given the available genotype information, and then use this information to find the most likely haplotype pair for each genotype in $\mathcal{G}$. These two steps are now straightforward to implement. First, λ can be trained directly from the genotype observations $\mathcal{G}$ using the EM algorithm (Dempster *et al.*, 1977). In fact, the resulting algorithm is a straightforward extension of the well-known forward-backward algorithm implementing EM in hidden Markov models (Rabiner, 1989). By training λ, we (implicitly) estimate the distribution $P(H)$ over haplotypes. Second, once λ is trained, finding the most likely haplotype pair for a given genotype corresponds to solving the hidden state inference problem given by Equation (7.1). In the proposed model this problem is naturally solved by an extension of the Viterbi algorithm used for computing the most likely hidden state sequence in HMMs (Rabiner, 1989). Note that the consistency constraint is automatically enforced in λ, as inconsistent combinations of haplotype and genotype observations have probability zero according to Equation (7.2).

7.2.1 Logical Hidden Markov Model Representation

Logical hidden Markov models (LoHMMs) upgrade traditional hidden Markov models to deal with sequences of logical atoms (Kersting *et al.*, 2006). The key idea underlying LoHMMs is to employ logical atoms as structured (output and state) symbols. More specifically, LoHMMs define *abstract* states such as $s(A, B)$ where s is the state name and A, B are logical variables. An abstract state represents a set of ground states, namely all ground states logically subsumed by the abstract state. For instance, $s(1, 0)$ is a possible ground state for the abstract state $s(A, B)$. Abstract transitions such as $s(X, Y) \rightarrow s'(1, Y)$ describe how the model transitions between abstract states, and variable unification is used to share information between different states and observations. Variants of the Expectation-Maximization and Viterbi algorithms used with standard HMMs can be derived for learning and inference in LoHMMs.

LoHMMs are a powerful and general representation formalism for statistical relational sequence models. In fact, the model outlined in the previous section was derived as a special case of a LoHMM model we had designed for the problem initially. Basically, we used LoHMMs as a rapid prototyping language to experiment with several different model architectures. After an effective representation was found in LoHMMs, we translated this representations to the propositional model outlined above. In the propositional model, inference and learning algorithms can be designed to be much more efficient. As an illustration, and to emphasize the relationship of this work to more general SRL frameworks, we now briefly outline the original LoHMM model for haplotyping.

First, note that genotypes are easily encoded as sequences of logical atoms by a predicate $pair(X, Y)$, which can be grounded to $pair(0, 0)$ (homozygous 0), $pair(1, 1)$ (homozygous 1), and $pair(0, 1)$ (heterozygous). Using logical variables and unification, the two individual alleles in the pair can be accessed. Second, the two (ordered) haplotype alleles underlying the pair can be represented by an internal hidden state $s_t(H_1, H_2)$ for $t \in \{1, ..., m\}$. The mapping from a hidden state $s_t(H_1, H_2)$ to a genotype observation $pair(X, Y)$ (given by Equation 7.2 in our model) is easily defined using logical unification: the model outputs $pair(X, X)$ if the current state is subsumed by the abstract state $s(X, X)$, and $pair(0, 1)$ otherwise. Finally, the assumption of Hardy-Weinberg equilibrium requires that a new state $s_{t+1}(H'_1, H'_2)$ is sampled from the current state $s_t(H_1, H_2)$ by independently sampling H'_1 and H'_2 from the same distribution (based on the respective current state). This sampling process can be implemented by a "sampling" state that is traversed twice, where the variable to be re-sampled is first bound to H_1 and afterwards to H_2.

Although conceptually simple, the full specification of the LoHMM model is somewhat involved and beyond the scope of this chapter. It is included in Appendix A.1 for the interested reader. Applying the standard EM and Viterbi algorithm for LoHMMs in this model coincides with EM and Viterbi in the propositional model depicted in Figure 7.2. However, initial experiments using the standard XANTHOS engine for LoHMMs[2] showed that the computational overhead due to the general-purpose framework used in LoHMMs reduced the computational efficiency of the model significantly: the domain-specific propositional model is several orders of magnitude more efficient.

7.3 Haplotype Structure Learning

The main limitation of the model presented so far is that it only takes into account dependencies between adjacent markers. Expressivity of the model can be increased by taking into account higher-order marker dependencies. However, this leads to an explosion in the number of parameters. We thus propose to learn a sparse high-order model structure that takes long-range marker dependencies into account, but prunes the model to only represent haplotype fragments that are frequent in the population under consideration.

7.3.1 Higher-order Models and Sparse Distributions

Expressivity of Markov-chain based haplotype models can be increased by using a Markov model of order $k > 1$ for the underlying haplotype distribution (Eronen *et al.*, 2004). Recall that the model λ is characterized by a distribution $P_\lambda(H) = P_\mu(H)$ over haplotypes,

[2]`http://people.csail.mit.edu/kersting/profile/`

as given by Equation (7.3) and Equation (7.4). This distribution can be generalized to a k-order Markov chain by

$$P_\lambda(H) = \prod_{t=1}^{m} P_\lambda(H[t] \mid H[t-1], ..., H[t-k]),$$

where $H[t-1], ..., H[t-k]$ is a shorthand for $H[t-1], ..., H[\max\{1, t-k\}]$. Unfortunately, the number of parameters in such a model increases exponentially with the history length k. Fortunately, observations on real-world data (as in Daly *et al.*, 2001) show that only few conserved haplotype fragments from the set of 2^k possible binary strings of length k actually occur. This can be exploited by modeling sparse distributions, where fragment probabilities which are estimated to be very low are set to zero. More precisely, let $p = P_\lambda(h[t] \mid h[t-1], ..., h[t-k])$ and define for some small $\epsilon > 0$ new probabilities

$$\hat{P}_\lambda(h[t] \mid h[t-1], ..., h[t-k]) = \begin{cases} 0 & \text{if } p \leq \epsilon; \\ 1 & \text{if } p > 1-\epsilon; \\ p & \text{otherwise} \end{cases}$$

corresponding to a regularized model.

This defines a sparse distribution, which can be represented using fewer parameters. In the regularized distribution, it holds that $P(h[i] \mid h[t-1], ..., h[t-k]) = 1$ for some histories $h[t-1], ..., h[t-k]$. This case represents that the only way of extending $h[t-k], ..., h[t-1]$ as a haplotype fragment is by $h[i]$. One can usually assume that the number of haplotype fragments of length k actually appearing in a population is relatively small, at least compared to the exponential number s^k of binary strings of length k. If the cut-off value ϵ is chosen correctly, the regularized distribution will thus be orders of magnitude more compact than the full distribution which involves $O(2^k)$ parameters. Note that to capture long-range dependencies, we are interested in models with relatively large k ($k = 30$ was used in our experiments). The corresponding sparse Markov model, in which transitions with probability 0 are removed, will reflect the pattern of conserved haplotype fragments present in the population. The learning and inference algorithms discussed in the previous section can be adapted to directly work on this sparse structure, greatly reducing computational complexity. How such a sparse model structure can be learned without ever constructing the prohibitively complex distribution non-regularized distribution P will be discussed in the next section.

7.3.2 SPAMM: A Level-wise Structure Learning Algorithm

To construct the sparse order-k model structure, we propose a learning algorithm—called **SPAMM** for **Spa**rse **M**arkov **M**odeling—that iteratively refines structured models λ of increasing order (Algorithm 5). More specifically, the idea of SPAMM is to identify conserved fragments using a level-wise search, that is, by extending short fragments (in low-order models) to longer ones (in high-order models), and is inspired by the well-known Apriori data mining algorithm (Agrawal *et al.*, 1996). The algorithm starts with a

Algorithm 5 The level-wise SPAMM learning algorithm.

```
Initialize $k := 1$
$\lambda_1 :=$ INITIAL-MODEL()
$\lambda_1 :=$ EM-TRAINING($\lambda_1$)
repeat
    $k := k + 1$
    $\lambda_k :=$ EXTEND-AND-REGULARIZE($\lambda_{k-1}$)
    $\lambda_k :=$ EM-TRAINING($\lambda_k$)
until $k = k_{max}$
```

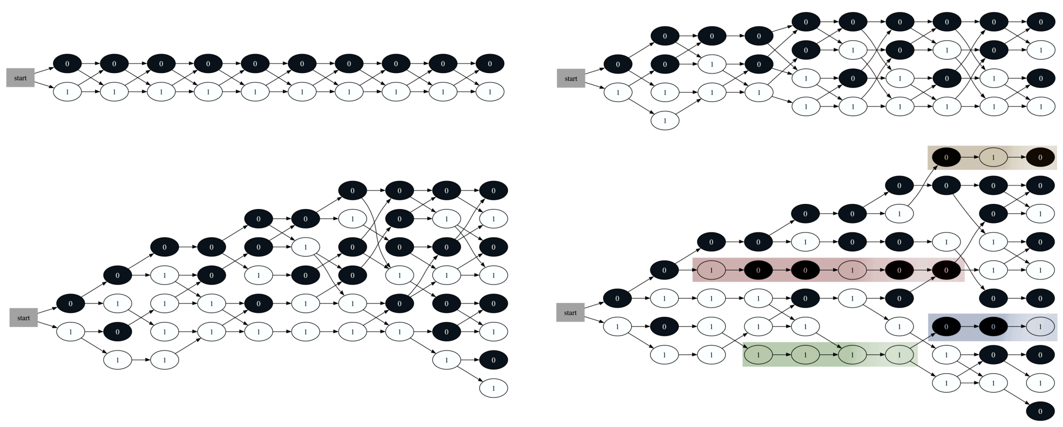


Figure 7.3: Visualization of the SPAMM Structure Learning Algorithm. Sparse models $\lambda_1, ..., \lambda_4$ of increasing order learned on the Daly dataset are shown. Black/white nodes encode more frequent/less frequent allele in population. Conserved fragments identified in λ_4 are highlighted.

first-order Markov model λ_1 where initial parameters are set to $\dot{P}_{\lambda_1}(h[t] \mid h[t-1]) = 0.5$ for all $t \in \{1, ..., m\}, h[t], h[t-1] \in \{0, 1\}$. The model λ_1 is then trained from the available genotype data (method EM-TRAINING(λ_1) in Algorithm 5).

The function EXTEND-AND-REGULARIZE(λ_{k-1}) takes as input a model of order $k-1$ and returns a model λ_k of order k. In λ_k, initial transition probabilities are set to

$$\dot{P}_{\lambda_{k+1}}(h[t] \mid h[t-k, t-1]) = \begin{cases} 0 & \text{if } P_{\lambda_k}(h[t] \mid h[t-k+1, t-1]) \leq \epsilon; \\ 1 & \text{if } P_{\lambda_k}(h[t] \mid h[t-k+1, t-1]) > 1-\epsilon; \\ 0.5 & \text{otherwise.} \end{cases}$$

Thus, transitions are removed if the probability of the transition conditioned on a shorter history is smaller than ϵ. This procedure of iteratively training, extending and regularizing Markov models of increasing order is repeated up to a maximum order k_{max}.

Figure 7.3 visualizes the sparse model structure learned in the first 4 iterations of the SPAMM algorithm on a real-world dataset. The set of all haplotypes that have positive probability according to the model is given by all paths from left to right through the graph structure. Note that in the initial model λ_1 all 2^m haplotypes are possible. For the later models many transitions are pruned, conserved fragments are isolated and the number of states in the final model is significantly smaller than for a full model of that order.

After the models of increasing order have been trained, they can be used to solve the actual haplotype reconstruction problem. For a given genotype g, a reconstructed haplotype pair $\langle h_g^1, h_g^2 \rangle_k$ can be obtained from every model λ_k. At the same time, the Viterbi algorithm computes $\mathbf{P}_{\lambda_k}(\langle h_g^1, h_g^2 \rangle_k \mid g)$, an estimate of the confidence of the reconstruction. In SPAMM, the reconstruction $\langle h_g^1, h_g^2 \rangle_{k^*}$ with the highest confidence is returned as the final solution:

$$k^* = \underset{k \in \{1,\ldots,k_{max}\}}{\arg\max} \; P_{\lambda_k}(\langle h_g^1, h_g^2 \rangle_k \mid g).$$

7.4 Experimental Evaluation

The proposed method was implemented in the SPAMM haplotyping system[3]. This section evaluates SPAMM by comparing it against well-established haplotyping systems developed in the bioinformatics community in term of effectiveness and efficiency. Specifically, the experimental evaluation will address the following questions:

(Q7.1) Is SPAMM competitive with other haplotyping systems in terms of reconstruction accuracy?

(Q7.2) Is SPAMM competitive for the related task of reconstructing masked genotypes?

(Q7.3) Does SPAMM scale well to long marker maps?

The following other state-of-the art haplotype reconstruction systems were included in the experimental study: PHASE version 2.1.1 (Stephens and Scheet, 2005), FASTPHASE version 1.1 (Scheet and Stephens, 2006), GERBIL as included in GEVALT version 1.0 (Kimmel and Shamir, 2005), HIT (Rastas *et al.*, 2005) and HaploRec (variable order Markov model) version 2.0 (Eronen *et al.*, 2006). All methods were run using their default parameters. Default parameters for SPAMM are $\epsilon = 0.1$ and $k_{max} = 30$. The FASTPHASE system, which also employs EM for learning a probabilistic model, uses a strategy of averaging results over several random restarts of EM from different initial parameter values. This reduces the variance component of the reconstruction error and alleviates the problem of local minima in EM search. As this is a general technique applicable also to our method, we list results for FASTPHASE with averaging (FASTPHASE) and without averaging (FASTPHASE-NA).

The methods were compared using publicly available real-world datasets, and larger datasets simulated with the Hudson coalescence simulator (Hudson, 2002). As real-world

[3]Implementation available from `http://www.cs.kuleuven.be/~niels/haplotyping.html`

Table 7.1: Reconstruction Accuracy on Yoruba and Daly Data. Normalized switch error is shown for the Daly dataset, and average normalized switch error over the 100 datasets in the Yoruba-20, Yoruba-100 and Yoruba-500 dataset collections.

Method	Yoruba-20	Yoruba-100	Yoruba-500	**Daly**
PHASE	**0.027**	**0.025**	*n.a.*	0.038
FASTPHASE	0.033	0.031	**0.034**	**0.027**
SpaMM	0.034	0.037	0.040	0.033
HAPLOREC	0.036	0.038	0.046	0.034
FASTPHASE-NA	0.041	0.060	0.069	0.045
HIT	0.042	0.050	0.055	0.031
GERBIL	0.044	0.051	*n.a*	0.034

data, we used a collection of datasets from the Yoruba population in Ibadan, Nigeria (The International HapMap Consortium, 2005), and the well-known dataset of Daly et al (Daly *et al.*, 2001), which contains data from a European-derived population. For these datasets, family trios are available, and thus true haplotypes can be inferred analytically.

For the Yoruba population, we sampled 100 sets of 500 markers each from distinct regions on chromosome 1 (**Yoruba-500**), and from these smaller datasets by taking only the first 20 (**Yoruba-20**) or 100 (**Yoruba-100**) markers for every individual. There are 60 individuals in the dataset after preprocessing, with an average fraction of missing values of 3.6%. For the **Daly** dataset, there is information on 103 markers and 174 individuals available after data preprocessing, and the average fraction of missing values is 8%. The number of genotyped individuals in these real-world datasets is rather small. For most disease association studies, sample sizes of at least several hundred individuals are needed(Wang *et al.*, 2005), and we are ultimately interested in haplotyping such larger datasets. Unfortunately, we are not aware of any publicly available real-world datasets of this size, so we have to resort to simulated data. We used the well-known Hudson coalescence simulator (Hudson, 2002) to generate 50 artificial datasets, each containing 800 individuals (**Hudson** datasets). The simulator uses the standard Wright-Fisher neutral model of genetic variation with recombination. To come as close to the characteristics of real-world data as possible, some alleles were masked (marked as missing) after simulation.

The accuracy of the reconstructed haplotypes produced by the different methods was measured by normalized switch error. The switch error of a reconstruction is the minimum number of recombinations needed to transform the reconstructed haplotype pair into the true haplotype pair. To normalize, switch errors are summed over all individuals in the dataset and divided by the total number of switch errors that could have been made. For more details on the methodology of the experimental study, confer (Landwehr *et al.*, 2007a).

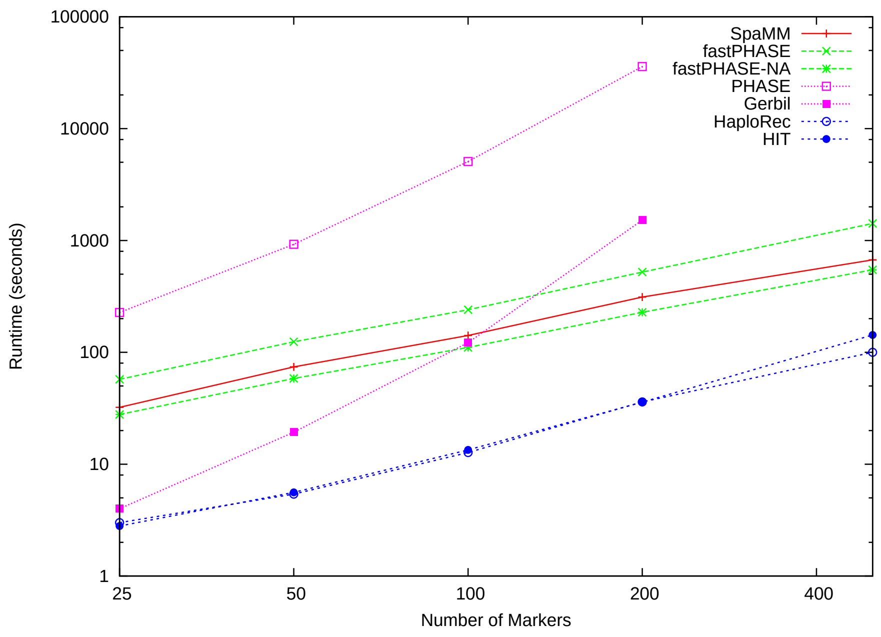


Figure 7.4: Runtime as a Function of the Number of Markers. Average runtime per dataset on Yoruba datasets for marker maps of length 25 to 500 for SpaMM, fastPHASE, fastPHASE-NA, PHASE, Gerbil, HaploRec, and HIT are shown (logarithmic scale). Results are averaged over 10 out of the 100 datasets in the Yoruba collection.

Table 7.1 shows the normalized switch error for all methods on the real-world datasets Yoruba and Daly. For the dataset collections Yoruba-20, Yoruba-100 and Yoruba-500 errors are averaged over the 100 datasets. PHASE and Gerbil did not complete on Yoruba-500 in two weeks[4]. Overall, the PHASE system achieves highest reconstruction accuracies. After PHASE, fastPHASE with averaging is most accurate, then SpaMM, and then HaploRec. Figure 7.4 shows the average runtime of the methods for marker maps of different lengths. The most accurate method PHASE is also clearly the slowest. fastPHASE and SpaMM are substantially faster, and HaploRec and HIT very fast. Gerbil is fast for small marker maps but slow for larger ones. For fastPHASE, fastPHASE-NA, HaploRec, SpaMM and HIT, computational costs scale linearly with the length of the marker map, while the increase is superlinear for PHASE and Gerbil, so computational costs quickly become prohibitive for longer maps.

[4]All experiments were run on standard PC hardware with a 3.2GHz processor and 2GB of main memory.

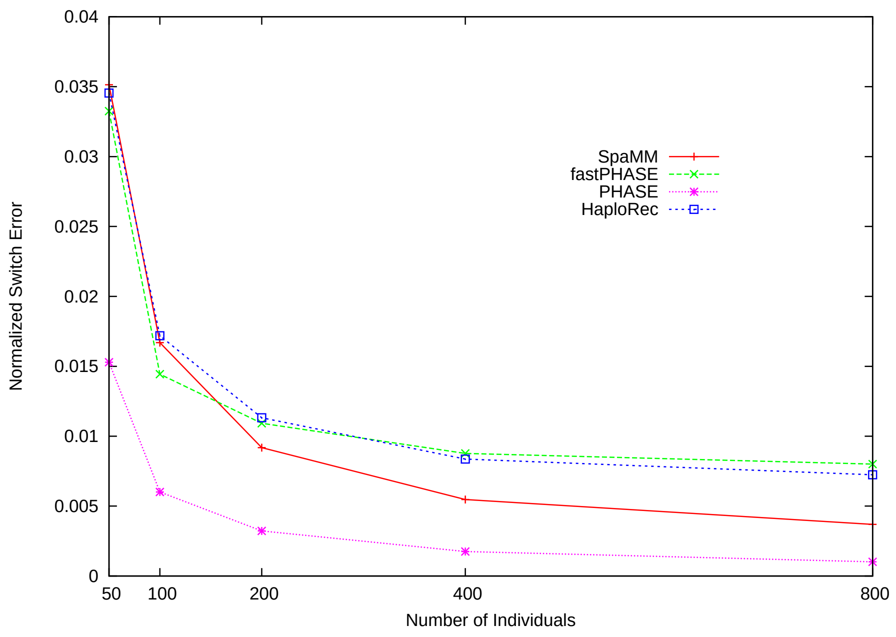


Figure 7.5: Reconstruction Accuracy as a Function of the Number of Samples Available. Average normalized switch error on the Hudson datasets as a function of the number of individuals for SpaMM, fastPHASE, PHASE and HaploRec is shown. Results are averaged over 50 datasets.

Performance of the systems on larger datasets with up to 800 individuals was evaluated on the 50 simulated Hudson datasets. As for the real-world data, the most accurate methods were PHASE, fastPHASE, SpaMM and HaploRec. Figure 7.5 shows the normalized switch error of these four methods as a function of the number of individuals (results of Gerbil, fastPHASE-NA, and HIT were significantly worse and are not shown). PHASE was the most accurate method also in this setting, but the relative accuracy of the other three systems depended on the number of individuals in the datasets. While for relatively small numbers of individuals (50–100) fastPHASE outperforms SpaMM and HaploRec, this is reversed for 200 or more individuals. We can conclude that SpaMM yields a competitive trade-off of reconstruction accuracy and computational efficiency on the datasets under consideration, and scales well to long marker maps. This indicates an affirmative answer to questions **(Q7.1)** and **(Q7.3)**.

A problem closely related to haplotype reconstruction is that of genotype imputation. Here, the task is to infer the most likely genotype values (unordered allele pairs) at marker

Table 7.2: Average Error for Reconstructing Masked Genotypes on Yoruba-100. From 10% to 40% of all genotypes were masked randomly. Results are averaged over 100 datasets.

Method	10%	20%	30%	40%
FASTPHASE	**0.045**	**0.052**	**0.062**	**0.075**
SpaMM	0.058	0.066	0.078	0.096
FASTPHASE-NA	0.067	0.075	0.089	0.126
HIT	0.070	0.079	0.087	0.098
GERBIL	0.073	0.091	0.110	0.136

Table 7.3: Average Error for Reconstructing Masked Genotypes on Hudson. From 10% to 40% of all genotypes were masked randomly. Results are averaged over 50 datasets.

Method	10%	20%	30%	40%
FASTPHASE	0.035	0.041	0.051	0.063
SpaMM	**0.017**	**0.023**	**0.034**	**0.052**
FASTPHASE-NA	0.056	0.062	0.074	0.087
HIT	0.081	0.093	0.108	0.127
GERBIL	0.102	0.122	0.148	0.169

positions where genotype information is missing, based on the observed genotype information. With the exception of HAPLOREC, all haplotyping systems included in this study can also impute missing genotypes. To test imputation accuracy, between 10% and 40% of all markers were masked randomly, and then the marker values inferred by the systems were compared to the known true marker values. Table 7.2 shows the accuracy of inferred genotypes for different fractions of masked data on the Yoruba-100 datasets and Table 7.3 on the simulated Hudson datasets with 400 individuals per dataset. PHASE was too slow to run in this task as its runtime increases significantly in the presence of many missing markers. Evidence from the literature (Scheet and Stephens, 2006) suggests that for this task, FASTPHASE outperforms PHASE and is indeed the best method available. In our experiments, on Yoruba-100 FASTPHASE is most accurate, SPAMM is slightly less accurate than FASTPHASE, but more accurate than any other method (including FASTPHASE-NA). On the larger Hudson datasets, SPAMM is significantly more accurate than any other method. This answer Question **(Q7.2)** affirmatively.

To summarize, our experimental results confirm PHASE as the most accurate but also computationally most expensive haplotype reconstruction system (Scheet and Stephens, 2006; Stephens and Scheet, 2005). If more computational efficiency is required, FASTPHASE yields the most accurate reconstructions on small datasets, and SPAMM is preferable for larger datasets. SPAMM also infers missing genotype values with high accuracy. For small datasets, it is second only to FASTPHASE; for large datasets, it is substantially more accurate than any other method in our experiments.

7.5 Related and Future Work

The proposed haplotyping method is related to a number of other haplotype reconstruction approaches, most notably HAPLOREC (Eronen *et al.*, 2004), HIT (Rastas *et al.*, 2005), and FASTPHASE (Scheet and Stephens, 2006). HIT also determines conserved fragments of haplotypes, and HIT and FASTPHASE use similar probabilistic modeling, parameter learning, and haplotype inference strategies.

Specifically, the idea of embedding a model on haplotypes into a model on genotypes in which the genotype phase is the hidden state information, and learning this model using EM, is related to the methodology used in the HIT (Rastas *et al.*, 2005) and FASTPHASE (Scheet and Stephens, 2006) systems. In HIT, haplotypes are modeled as recombinations of a set of "founder" haplotypes, that is, haplotypes of an assumed small population of "founder" individuals from which the current population developed. An instance of the EM algorithm is derived to directly estimate the founders from genotype observations. In FASTPHASE, haplotypes are modeled using local clusters, and cluster membership of a haplotype is determined by a hidden Markov model. Again, an instance of the EM algorithm for estimating the clusters directly from genotype data can be derived. Moreover, the idea of using frequent fragments to build Markov models for haplotypes has also been used in the HAPLOREC method (Eronen *et al.*, 2004). In HAPLOREC, a set of fragments (of any length) that are frequent according to the current model is kept, and updated after each iteration of the EM algorithm.

Furthermore, we performed related work on *combining haplotype reconstructions* obtained from different haplotyping systems. This was achieved by defining appropriate distance functions between haplotype pairs, and employing dynamic programming algorithms to compute mean or median reconstructions based on these distances. More details on combinations of haplotype reconstructions can be found in (Kääriäinen *et al.*, 2007; Landwehr and Mielikäinen, 2008).

More generally, the presented method is also related to sequential statistical relational learning frameworks, such as RMMs by Anderson *et al.* (2002), LoHMMs presented by Kersting *et al.* (2006), and the r-gram and CPT-L approaches presented in Chapter 5 and Chapter 6 of this thesis.

The most interesting direction for future work is to employ similarly structured models in related application areas. Haplotype data yields insight into the organization of the human genome: how individual markers are inherited together, the distribution of variation in the genome, or regions which have been evolutionary conserved (indicating locations of important genes). At the data analysis level, we are therefore interested in analyzing the structure in populations—to determine, for example, the difference in the genetic make-up of a case and a control population—and the structure in haplotypes. Understanding these underlying structures, rather than focusing on the haplotype reconstruction problem only, should be an important field in which structured and relational learning techniques can be beneficial. Moreover, an important direction to explore is how existing prior knowledge can best be used in such problem settings.

Chapter 8

Interleaved Hidden Markov Models for Activity Recognition*

Activity Recognition is the problem of inferring the activity a user is currently performing based on sensor data collected in the user's environment. The ability to recognize human activities from sensory information is essential in many application domains. For instance, smart systems must be able to recognize the current context of a user and the activity she is performing in order to suggest or take actions in an intelligent manner. This chapter introduces a structured probabilistic model for the recognition of *interleaved* activities, and domain-specific techniques for learning such models from data and performing activity inference.

The specific problem considered in this chapter is *activity of daily living*, or *ADL*, recognition (Wang *et al.*, 2007). The activities of daily living we consider have two characteristics: (1) they are hierarchically structured, meaning that an activity consists of several substeps, and (2) activities are typically interleaved in time as a user is switching back and forth between different tasks. Taken together, these characteristics present significant challenges for standard (unstructured) approaches such as propositional hidden Markov models. In existing approaches to activity modeling the hierarchical structure in activities is therefore often ignored.

We develop a domain-specific structured probabilistic model that takes into account hierarchical structure and activity interleaving explicitly. The model can be seen as the simplest generalization of a hidden Markov model to represent multiple, interleaved processes. As for the haplotyping model discussed in the previous section, the model turns out to be representable in the logical hidden Markov model framework. Unfortunately, exact inference in the proposed model can be proven to be NP-hard. However, we present a tractable and effective domain-specific inference algorithm that builds on the chain-

*This chapter builds on (Landwehr, 2008).

wise Viterbi algorithm used in factorial hidden Markov models (Ghahramani and Jordan, 1997).

The chapter is organized as follows. ADL recognition will be discussed in more detail in the next section. In Section 8.2 the model will be presented, and we will briefly sketch how it can be represented as a logical hidden Markov model. Section 8.3 discusses inference and learning. Section 8.4 presents experimental results, and Section 8.5 discusses related work.

8.1 Recognizing Activities of Daily Living

The specific activity recognition scenario considered in this chapter is as follows. Assume a user is performing some activity of daily living (such as making breakfast). Assume further that the objects typically used in the activity under consideration are equipped with small radio frequency identification (RFID) tags, and the user is wearing a mobile RFID sensor in a bracelet around the wrist. The sensor identifies objects which are close (approximately 10–15 cm) to the wrist of the user; thereby, we observe the sequence of objects a user has been touching while performing the activity. The task is to infer the current activity from this stream of sensor data. In the light of recent advances in RFID technology, which allow tags to be cheaply mass-produced and readers to be made wearable, such application scenarios are attracting increasing research interest from both academia and industry (Wang *et al.*, 2007).

Example 8.1.1. *Let the activities under consideration be* toast bread*,* add flavor to toast*,* boil water*, and* add flavor to tea*. A possible sequence of observed objects (sensor readings), together with true activity labels, could then be as follows:*

Activity Tag	ToastBread								FlavorToast											BoilWater					FlavorTea					
Sensor Reading	toast	toast	toaster	toaster	toaster	toaster	toast	toast	knife	knife	knife	butter	butter	toast	toast	knife	knife	jam	jam	water	water	water	stove	stove	cup	spoon	spoon	sugar	sugar	cup
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

In the training data, sequences would be labeled with the true activity at every point in time by a human observer. For test sequences, the sensor reading is observed and the task is to predict the correct activity label for every point in time.

Many different approaches have been developed for activity recognition problems. They are based on different classes of sensors, such as cameras (Harville and Li, 2004), pressure mat sensors (Orr *et al.*, 2000), RFID tags (Wang *et al.*, 2007), body-worn accelerometers (Bao and Intille, 2004), or GPS data (Liao *et al.*, 2005a). Most activity recognition systems employ probabilistic models, including HMMs (Patterson *et al.*, 2005) or more complex models such as dynamic Bayesian networks (Wang *et al.*, 2007) or conditional random fields (Liao *et al.*, 2005a). They have also focuses on modeling

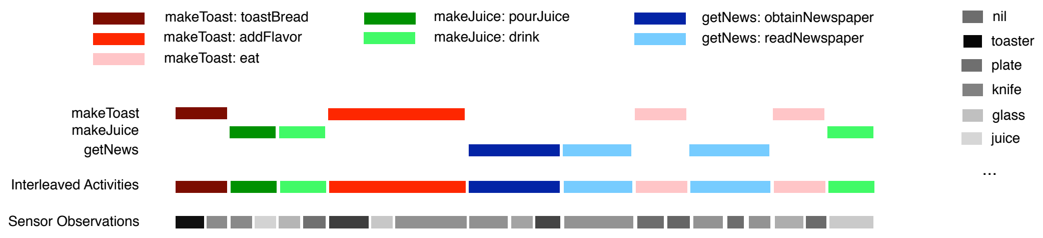


Figure 8.1: Interleaving in an activity recognition domain. Three high-level activities (makeToast, makeJuice, getNews) with corresponding basic activities are interleaved in time as a user switches between them. Different activities can produce identical sensor observations, and therefore neither the interleaving nor the actual activities are directly observable from the sensor data.

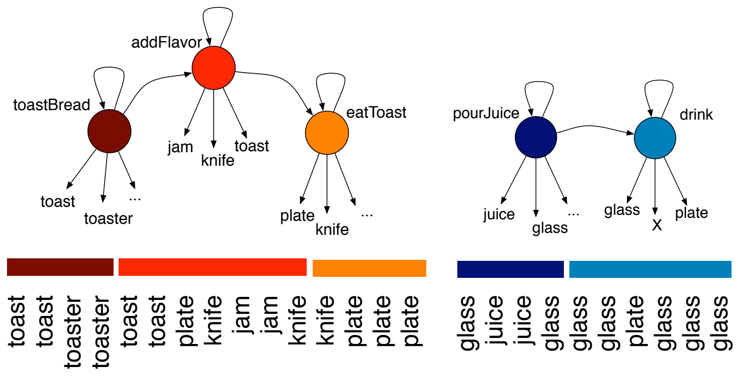
different aspects of human activity, such as object usage (Patterson *et al.*, 2005), activity duration (Duong *et al.*, 2005), or hierarchical activities (Riesenhuber and Poggio, 1999).

The approach discussed in this chapter is focused on recognizing *interleaved* activities, that is, scenarios in which users often switch between executing different tasks. More specifically, we take into account two aspects of real-world environments that complicate the simple activity recognition setting outlined above. First, activities are often hierarchically structured, as sets of *basic* activities can be grouped into *high-level* activities. For instance, there could be a high-level activity *make toast* that consists of basic activities *toast bread*, *flavor toast*, and *eat*. It is useful to take this structure into account: the likelihood of activity *flavor toast* being performed now depends on whether we (indirectly) observed the activity *toast bread* at an earlier point in time. Second, high-level activities are often *interleaved* in time as a user is switching back and forth between different tasks, as illustrated in Figure 8.1. In this example, a user is having breakfast, which consists of high-level activities *make toast*, *make juice*, and *get news* with corresponding basic activities.

8.2 A Model for Interleaved Processes

A popular simple approach for activity recognition are hidden Markov models: each (atomic) activity is modeled as a hidden state in the HMM, and this hidden state emits a certain subset of object tags typically associated with the activity. This approach is applicable if there is one set of atomic activities without hierarchical structure. The activities we consider are hierarchically structured, that is, every activity consists of a number of substeps. In this case, it is natural to model each activity as one HMM, where hidden states correspond to the substeps within the activity, and emit the object tags associated with this substep. However, now the resulting set of HMMs needs to be matched against the given sequence of object tags during activity recognition. That is, we need to determine which observations have been generated by which activity. This can be accomplished, for instance, using particle filters.

Example 8.2.1. *Consider high-level activities* make toast *and* make juice *with basic activities* toast bread, add flavor, eat toast *and* pour juice, drink *respectively. This situation can be represented by two hidden Markov models as follows:*



Note that the hidden Markov models are given in automaton *notation, as introduced in Section 2.1.2. Each node corresponds to one state the hidden state variable can take on, and arrows between nodes indicate possible transitions from one state to another (for example, from state* toastBread *to state* addFlavor*). Actual probability values have been omitted.*

The use of a set of HMMs in this way is appropriate if activities are carried out sequentially. However, the approach is not directly applicable if activities interleave as in Figure 8.1. The interleaved scenario can be modeled with one standard HMM by “flattening” the three activities shown in Figure 8.1 into one process with 7 states (as in Patterson *et al.*, 2005). This model is shown in Figure 8.2. It can account for interleaved activities as transitions from any low-level activity to any other low-level activity are possible. However, part of the problem structure is lost, and the number of parameters representing transition probabilities is inflated. In this chapter, we instead propose to model the activities as three *different* processes which interleave in time. This has the advantage of decoupling transition dynamics within one high-level activity from the interleaving behavior, yielding a more concise representation of the overall dynamics.

We now present a structured probabilistic sequence model in which observations are generated by multiple, interleaved hidden processes. More specifically, we will pursue the simplest generalization of a hidden Markov model to multiple processes: individual processes are characterized by first-order Markov chains, and the switching mechanism by which they interleave is again Markov. Note that this is the most restricted model that can capture the particular domain structure we are interested in. We deliberately avoid the expressivity—and associated complexity—of more general models.

Let the random variables $Y_1, ..., Y_T$ denote a sequence of observations, where the Y_t take on one of D discrete values. A hidden Markov model μ defines a sequence

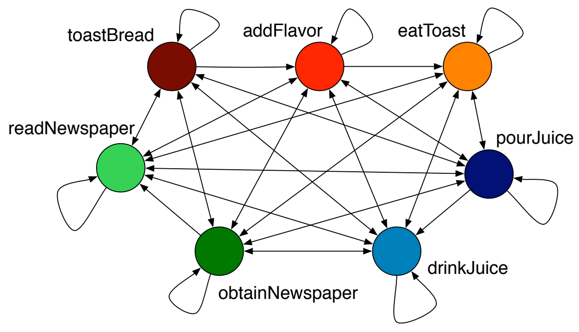


Figure 8.2: A HMM for the activity recognition domain in which all low-level activities are collapsed into one process with 7 states (no emissions are shown to improve readability).

$X_1, ..., X_T$ of hidden state variables, with $X_t \in \{1, ..., K\}$ and K the number of different states the hidden process can take on (cf. Section 2.1.2). To simplify notation, assume that there is a special start state 0 the process is in at time $t = 0$, that is, $X_0 = 0$. The first transition is from X_0 to $X_1 \in \{1, ..., K\}$, and afterwards the first output Y_1 is emitted. The HMM is characterized by initial state probabilities $a_{0i} = P(X_1 = i \mid X_0 = 0)$, state transition probabilities $a_{ij} = P(X_t = j \mid X_{t-1} = i)$ for $t \geq 2$ and emission probabilities $b_{il} = P(Y_t = l \mid X_t = i)$ for $t \geq 1$. The joint distribution of observations $\mathbf{Y} = Y_1, ...Y_T$ and hidden states $\mathbf{X} = X_1, ..., X_T$ is given by

$$P(\mathbf{X}, \mathbf{Y}) = \prod_{t=1}^{T} P(X_t \mid X_{t-1}) P(Y_t \mid X_t).$$

We will also refer to $\mathbf{X}$ as the hidden process that generated the observations $\mathbf{Y}$.

We propose a model for multiple, interleaved hidden processes as follows. Intuitively, an additional *switching process* controls a token that is handed from process to process, and determines which of the processes is active at a particular point t in time. The active process transitions to a new state and outputs the observation Y_t, while all other processes remain "frozen" in time. More formally, let $\mu_1, ..., \mu_M$ be hidden Markov models with initial state probabilities $a_{0i}^{(m)}$, transition probabilities $a_{ij}^{(m)}$ and emission probabilities $b_{il}^{(m)}$. For ease of notation, we assume the number of states K is identical for all μ_m, but the model trivially generalizes to processes with state spaces of different size. Let furthermore $\bar{\mu}$ be a Markov process with states $\{1, ..., M\}$, initial state probabilities d_{0i} and transition probabilities d_{ij}. Let Z_t denote a random variable representing the state of $\bar{\mu}$ at time t, and $S_t^{(m)}$ denote random variables representing the state of process μ_m at time t for $1 \leq m \leq M$. $Z_t \in \{1, ..., M\}$ determines the *active* process at time t, and we will refer to $\bar{\mu}$ as the *switching process*. At every step t in time, a new active process is sampled from $\bar{\mu}$ with probability $P(Z_t = j \mid Z_{t-1} = i) = d_{ij}$. Afterwards, the states of

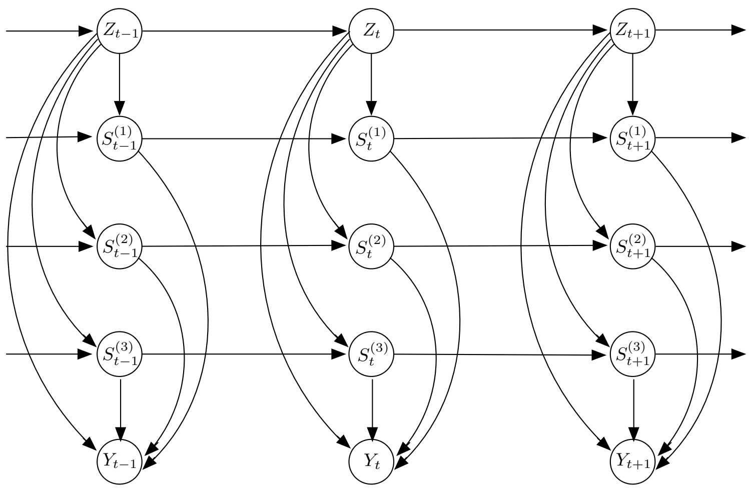


Figure 8.3: Interleaved mixture of hidden Markov models in dynamic Bayesian network notation (for $M = 3$).

$\mu_1, ..., \mu_M$ are updated according to

$$P(S^{(m)}_t = j \mid S^{(m)}_{t-1} = i, Z_t = k) = \begin{cases} a^{(m)}_{ij} & k = m; \\ \delta_{ij} & k \neq m, \end{cases} \tag{8.1}$$

where $\delta_{ii} = 1$ and $\delta_{ij} = 0$ for $i \neq j$. In other words, a process μ_m transitions to a new state with probability given by its transition matrix if it is active at time t, and stays in its old state otherwise. Finally, the probability of emitting symbol Y_t is

$$P(Y_t = l \mid S^{(1)}_t = i_1, ..., S^{(M)}_t = i_M, Z_t = k) = b^{(k)}_{i_k l} \tag{8.2}$$

That is, it is given by the emission probability of the process that is active at time t. Let $\mathbf{S}_t = S^{(1)}_t, ..., S^{(M)}_t$, $\mathbf{Z} = Z_1, ..., Z_T$ and $\mathbf{S} = \mathbf{S}_1, ..., \mathbf{S}_T$. Then

$$P(\mathbf{Z}, \mathbf{S}, \mathbf{Y}) = \prod_{t=1}^{T} P(Z_t \mid Z_{t-1}) P(Y_t \mid \mathbf{S}_t, Z_t) \prod_{m=1}^{M} P(S^{(m)}_t \mid S^{(m)}_{t-1}, Z_t) \tag{8.3}$$

We will refer to this model as an *interleaved mixture of hidden Markov models*. It is represented by the dynamic Bayesian network structure given in Figure 8.3. The model is structurally related to a factorial hidden Markov model (Ghahramani and Jordan, 1997), shown in Figure 8.4. However, the structure is extended by the additional chain of Z_t nodes that determine the currently active process. Note that although the structure is densely connected, the set of parameters is simply the union of the parameter sets of

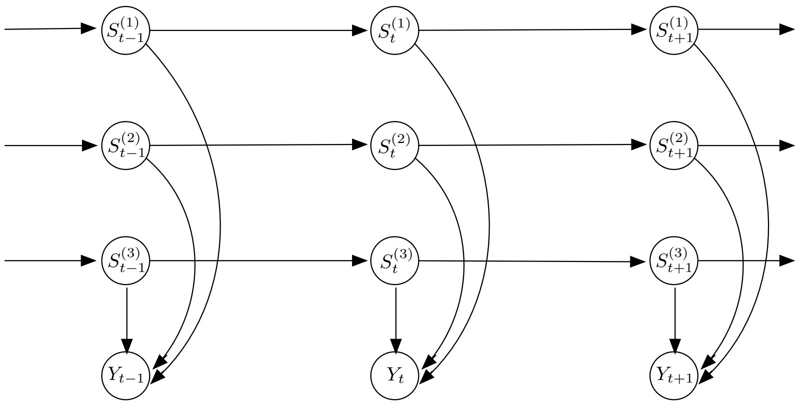


Figure 8.4: Factorial hidden Markov model in dynamic Bayesian network notation (for $M = 3$).

the constituent HMMs $\mu_1, ..., \mu_M$ and the switching process $\bar{\mu}$ (Equation (8.3) and Equation (8.2)).

The following alternative interpretation of the model can be given. Let $\mathbf{z}$ denote an interleaving[1] and let $t^m_1, ..., t^m_{T^m}$ denote the sequence positions for which $z_t = m$ for $m \in \{1, ..., M\}$. That is, $\mathbf{Y}_{\downarrow\mu_m} = Y_{t^m_1}, ..., Y_{t^m_{T^m}}$ is the projection of $\mathbf{Y}$ to elements generated by μ_m, and $\mathbf{S}_{\downarrow\mu_m} = S^{(m)}_{t^m_1}, ..., S^{(m)}_{t^m_{T^m}}$ the corresponding hidden state variables. It is easily verified that

$$P(\mathbf{z}, \mathbf{S}, \mathbf{Y}) = P(\mathbf{z}) \prod_{m=1}^{M} P_{\mu_m}(\mathbf{Y}_{\downarrow\mu_m}, \mathbf{S}_{\downarrow\mu_m}),$$

where $P_{\mu_m}(\mathbf{Y}_{\downarrow\mu_m}, \mathbf{S}_{\downarrow\mu_m})$ is the joint distribution of hidden states $\mathbf{S}_{\downarrow\mu_m}$ and observations $\mathbf{Y}_{\downarrow\mu_m}$ in the original HMM μ_m. This reformulation gives rise to an intuitive approach for sampling from $\mathbf{Y}$: first sample an interleaving pattern $\mathbf{z}$ from $\bar{\mu}$, and afterwards $\mathbf{Y}_{\downarrow\mu_m}$ from μ_m for $1 \leq m \leq M$.

8.2.1 Logical Hidden Markov Model Representation

To illustrate the relationship of the domain-specific structured model introduced in the previous section to more general SRL approaches, let us briefly discuss how the model can be represented in the logical hidden Markov model framework (Kersting *et al.*, 2006). The key observation for realizing this is that the current state of the model in Figure 8.3 is determined by the joint state of the variables $S^{(1)}_t, ..., S^{(M)}_t$ and Z_t. That is, it is determined by the states of the constituent processes and the switching process. This joint

[1]As introduced in Section 2.1, we denote random variables with upper-case letters, and their instantiations with lower-case letters.

state is easily represented by a predicate $p(S_t^{(1)}, ..., S_t^{(M)}, Z_t)$. Two important domain properties now need to be enforced. First, in the transition from $p(S_t^{(1)}, ..., S_t^{(M)}, Z_t)$ to a new joint state $p(S_{t+1}^{(1)}, ..., S_{t+1}^{(M)}, Z_{t+1})$ only the state $S_{t+1}^{(a)}$ of the active process is re-sampled (that is, $Z_{t+1} = a$), while for $k \neq m$, it holds that $S_{t+1}^{(k)} = S_t^{(k)}$. This "selective sampling" can be compactly represented in LoHMMs based on abstract states such as $p(s_t^{(1)}, S_t^{(2)}, ..., S^{(M)}, 1)$: this state subsumes all ground states in which the first process is active and is in state $s_t^{(1)}$. We then need to transition to an abstract state $p'(s_{t+1}^{(1)}, S_t^{(2)}, ..., S^{(M)}, 1)$, in which $s_t^{(1)}$ has been re-sampled to $s_{t+1}^{(1)}$, and all other process states are left unchanged according to the variable bindings. Second, the output symbol needs to be sampled from the currently active state. Again, this can be achieved using abstract states such as $q(s_t^{(1)}, S_t^{(2)}, ..., S^{(M)}, 1)$ subsuming all ground states in which the first process is active and is in state $s_t^{(1)}$ (and the states of all other processes are ignored).

The full specification of the LoHMM model involves states for 1) re-sampling the switching chain, 2) re-sampling the currently active process, and 3) emitting the corresponding output symbol. It is described in more detail in Appendix A.2.

Note that the logical hidden Markov models should be seen as one way of implementing structured probabilistic sequence models such as the proposed interleaved mixtures of HMMs, factorial hidden Markov models, or related approaches. While a custom implementation of the propositional model will typically be much more efficient, more general SRL formalism such as LoHMMs can be helpful for developing the structure of the propositional model and as a language for rapid prototyping. Once a suitable way of modeling the domain has been found, the model can be grounded into a propositional implementation such as the one described in the previous section.

8.3 Inference and Parameter Estimation

A key task in the activity recognition domains we have in mind is *hidden state inference*: find

$$(\mathbf{z}^*, \mathbf{s}^*) = \arg\max_{\mathbf{z},\mathbf{s}} P(\mathbf{z}, \mathbf{s} \mid \mathbf{y}) \tag{8.4}$$

for a given sequence $\mathbf{y}$ of observations (cf. Problem 2.1.2 in Chapter 2). In contrast to a standard HMM, hidden state inference in the proposed structured model involves simultaneously finding a segmentation of $\mathbf{y}$ into subsequences $\mathbf{y}_{\downarrow\mu_m}$ generated by μ_m (the $\mathbf{z}^*$), and most likely hidden states for $\mathbf{y}_{\downarrow\mu_m}$ in μ_m (the $\mathbf{s}^*$).

8.3.1 Exact Inference

Two special cases of the problem are trivial. For $M = 1$, the model coincides with a hidden Markov model, and the Viterbi algorithm returns the most likely hidden states in time $O(K^2T)$ (Rabiner, 1989). Moreover, if the output symbol sets of $\mu_1, ..., \mu_M$ are disjoint, the interleaving **z** is directly observable, and **s** can be obtained by running M instances of Viterbi in time $O(MK^2T)$.

The more interesting case of $M \geq 2$ and non-disjoint output symbol sets is inherently more difficult due to its combinatorial nature—the M constituent chains are coupled via the switching process and observations, and thus cannot be handled independently. Accordingly, exact graphical model inference (for example, with the junction tree algorithm) applied to the model in Figure 8.3 has costs exponential in M, because the cliques at Y_t are of size $O(M)$. In fact, for *general* graphical model structures of this form there is no tractable inference algorithm available (Dagum and Chavez, 1993). However, the conditional distributions $P(Y_t \mid \mathbf{S}_t, Z_t)$ have a particularly simple form, which could make the problem easier. Unfortunately, this is not the case:

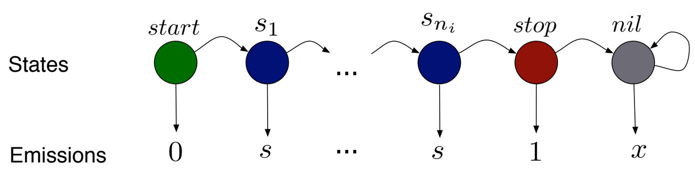
Theorem 8.3.1. *Exact inference for interleaved mixtures of hidden Markov models is NP-hard.*

The theorem is proved by reduction from the strongly NP-hard *3-partition problem* (Garey and Johnson, 1975):

Problem 8.3.1 (3-Partition Problem)**.** *Let $\mathcal{S}$ be a multiset of $M = 3N$ positive integers. Is there a partition of $\mathcal{S}$ into subsets $S_1, ..., S_N$ of size three each such that the sum over the integers in each subset is the same?*

Proof of Theorem 8.3.1. In order to define the reduction, assume we are given a 3-partition problem, with a multiset $\mathcal{S} = \{n_1, ..., n_M\}$ of positive integers. The problem has a solution if there is partition of $\mathcal{S}$ into $N = M/3$ subsets of size three each such that the sum over the integers in each subset is $B = \frac{1}{N}\sum_{i=1}^{M} n_i$.

We now define an interleaved mixture of HMMs λ and an observation sequence y such that $P(y \mid \lambda) > 0$ iff there is a solution to the underlying 3-partition problem. The model λ has M processes, one for each integer $n_i \in \mathcal{S}$. The process corresponding to integer n_i has states $start$,$s_1, ..., s_{n_i}$,*end*,and nil. It deterministically transitions from $start$ through $s_1, ..., s_{n_i}$, to end and then to nil. The process then stays in nil indefinitely. Furthermore, it has deterministic emission which are 0 for the state $start$, s for the states $s_1, ..., s_{n_i}$, 1 for the state end, and x for the state nil. The process is depicted below in automaton notation:



Initial state and transition probabilities for the switching process in λ are uniform. Now consider the following sequences y^* and y:

$$y^* = 000\underbrace{s...s}_{B}111 \qquad\qquad y = \underbrace{y^*...y^*}_{N}$$

It is easy to see that $P(y^* \mid \lambda) > 0$ iff there are three integers $n_{i_1}, n_{i_2}, n_{i_3} \in \mathcal{S}$ such that $n_{i_1} + n_{i_2} + n_{i_3} = B$. Similarly, it is clear that $P(y \mid \lambda) > 0$ iff there are N disjoint subsets $S_1, ..., S_N \subseteq \mathcal{S}$ of three integers each such that $\sum_{n_i \in S_k} n_i = B$ for $k \in \{1, ..., N\}$. Thus, the original 3-partition problem has a solution iff $P(y \mid \lambda) > 0$.

Note that the size of λ is polynomial in the size of M and the numbers $n_1, ..., n_M$. However, $n_1, ..., n_M$ can be exponential in M. Thus, a crucial point is the *strong* NP-hardness of the 3-partition problem: the problem is NP-hard even if numbers in the input are given in unary notation (or, equivalently, if integers in $\mathcal{S}$ are polynomially bounded in M).

□

8.3.2 Approximate Inference

Approximate inference in graphical models has received much attention, and a variety of techniques are available (see Jordan and Weiss, 2002, for an overview). The simplest class of methods are Markov chain Monte Carlo (MCMC) approaches. The idea behind MCMC is that while it might not be feasible to compute expectations or maximizing values for a particular distribution $p(x)$, it can be possible to obtain (approximate) samples from it, and then compute sample-based approximations to the desired quantities. Formally, an MCMC method defines a Markov chain whose state space is the joint state of all variables involved in the distribution $p(x)$ to be estimated, and whose stationary distribution coincides with $p(x)$. Samples drawn from the Markov chain for large enough t thus approximate samples from $p(x)$.

In Gibbs sampling, for instance, iterative conditional resampling of random variables in the network defines a Markov process whose stationary distribution—under certain conditions—will be the conditional distribution in Equation (8.4). However, Gibbs sampling is not an effective inference method in our case, because the Markov process defined by the Gibbs sampler is not ergodic. There can be two state configurations with positive probability that cannot be transformed into each other by single-variable changes without passing through an invalid (probability zero) configuration, such as any configuration with $S_{t-1}^{(m)} \neq S_t^{(m)}$ but $Z_t \neq m$. This effectively traps the Gibbs sampler in a subspace of all configurations and prevents MCMC convergence.

The problem is that Gibbs sampling, by updating only one variable at a time, ignores the specific model structure. Instead, we have to resort to approximate inference methods that better exploit model structure. Examples include structured variational approximations (Ghahramani and Jordan, 1997) and an iterative approximate inference method known as the chainwise Viterbi algorithm (Saul and Jordan, 1999). These algorithms are used in factorial HMMs for computing EM statistics and hidden state inference. In the rest of the Section, we present an extension of chainwise Viterbi for solving the problem given by Equation (8.4).

The idea behind chainwise Viterbi is to repeatedly solve tractable sub-problems of the (intractable) global optimization problem. For factorial hidden Markov models, the natural sub-problem to solve is to optimize hidden states in one chain $\mathbf{S}^{(m)} = S_1^{(m)}, ..., S_T^{(m)}$ conditioned on the current states of the other chains:

$$\begin{aligned}\mathbf{s}_{new}^{(m)} =& \arg\max_{\mathbf{s}^{(m)}} P(\mathbf{s}^{(m)} \mid \{\mathbf{s}^{(l)} : l \neq m\}, \mathbf{y}) \\ =& \arg\max_{\mathbf{s}^{(m)}} P(\mathbf{s}^{(1)}, ..., \mathbf{s}^{(M)}, \mathbf{y}).\end{aligned}$$

In the dynamic Bayesian network representing an interleaved mixture of HMMs (Figure 8.3), there are two different types of hidden chains: the chains $\mathbf{S}^{(1)}, ..., \mathbf{S}^{(M)}$ representing the constituent processes $\mu_1, ..., \mu_M$ and the chain $\mathbf{Z}$ representing the switching process $\bar{\mu}$. Assume first that $\mathbf{Z}$ is kept fix, and the goal is to conditionally optimize a chain $\mathbf{S}^{(m)}$. This is straightforward: for a given interleaving pattern, the chains $\mathbf{S}^{(1)}, ..., \mathbf{S}^{(M)}$ become independent given $\mathbf{Z}$ and $\mathbf{Y}$ due to the special form of the conditional distributions $P(Y_t \mid \mathbf{S}_t, Z_t)$, cf. Equation (8.2). They can thus be optimized independently with standard Viterbi.

We therefore focus on the task of optimizing $\mathbf{Z}$ given $\mathbf{S}^{(1)}, ..., \mathbf{S}^{(M)}$. A straightforward update

$$\mathbf{z}_{new} = \arg\max_{\mathbf{z}} P(\mathbf{z}, \mathbf{s}^{(1)}, ..., \mathbf{s}^{(M)}, \mathbf{y})$$

is not very effective: as a process μ_m can only change state at time t if it is active, we know from $S_t^{(m)} \neq S_{t-1}^{(m)}$ that $Z_t = m$. Thus, the joint state of $\mathbf{S}^{(1)}, ..., \mathbf{S}^{(M)}$ essentially determines $\mathbf{Z}$. To change the state of Z_t from m to n, it is necessary to also update $S_t^{(m)}$ and $S_t^{(n)}$ to reflect that μ_n is now active at time t. The solution is to jointly optimize *two* constituent chains $\mathbf{S}^{(m)}$, $\mathbf{S}^{(n)}$ and the switching chain $\mathbf{Z}$ by

$$(\mathbf{s}_{new}^{(m)}, \mathbf{s}_{new}^{(n)}, \mathbf{z}_{new}) = \arg\max_{\mathbf{s}^{(m)}, \mathbf{s}^{(n)}, \mathbf{z}} P(\mathbf{s}, \mathbf{y}, \mathbf{z}). \tag{8.5}$$

Intuitively speaking, this update makes it possible to re-assign observations that have so far been attributed to process μ_m to process μ_n, by changing some Z_t from m to n and updating $\mathbf{S}^{(m)}$ and $\mathbf{S}^{(n)}$ accordingly. If it is repeatedly applied with different process indices m, n, the interleaving can be arbitrarily revised. Algorithm 6 describes this chain-wise update scheme in pseudocode. The method consistent-configuration($\mathcal{M}$) initializes the states of the hidden variables to some positive-probability configuration[2]. When choosing $m, n \in \{1, ..., M\}$ different strategies are possible; we assume the algorithm repeatedly cycles through all pairs $n \neq m$. If the update step (8.5) is implemented exactly, $P(\mathbf{s}, \mathbf{z}, \mathbf{y})$ will increase unless the hidden state configuration is left unchanged. Thus, the algorithm will always converge (though not necessarily to the true global optimum).

An efficient implementation of the update step (8.5) is crucial for fast inference. This can be achieved by dynamic programming in the spirit of the Viterbi algorithm (Rabiner, 1989). Moreover, the particularly restrictive form of the model (basically, that only the

[2]This is trivial if observation probabilities are always non-zero, as, for instance, in Laplace-smoothed models.

Algorithm 6 Chainwise Viterbi for interleaved mixtures of hidden Markov models

```
Input: model $\mathcal{M}$, observations $\mathbf{Y}$
$(\mathbf{S}, \mathbf{Z})$ := consistent-configuration($\mathcal{M}$)
while not converged do
    choose $m, n \in \{1, ..., M\}, m \neq n$
    let $(\mathbf{S}^{(m)}, \mathbf{S}^{(n)}, \mathbf{Z}) := \arg\max_{\mathbf{S}^{(m)}, \mathbf{S}^{(n)}, \mathbf{Z}} P(\mathbf{S}, \mathbf{Z}, \mathbf{Y})$
end while
return $\mathbf{S}, \mathbf{Z}$
```

active chain changes state at any point in time) can be exploited. This yields a much faster inference algorithm than for *general* graphical models with the DAG structure given in Figure 8.3, as will be briefly outlined now.

To simplify notation, assume that $n = 1$ and $m = 2$. In analogy to the Viterbi algorithm, define

$$\delta_{ijk}[t] = \max_{\mathbf{D}} P(\mathbf{D}, S_t^{(1)} = i, S_t^{(2)} = j, Z_t = k, \mathbf{Y}, \mathbf{S}^{(3)}, ..., \mathbf{S}^{(M)})$$

with

$$\mathbf{D} = \{S_1^{(1)}, ..., S_{t-1}^{(1)}, S_1^{(2)}, ..., S_{t-1}^{(2)}, Z_1, ..., Z_{t-1}\}.$$

Initialization of $\delta_{ijk}[1]$ is straightforward. For the recursive definition of $\delta_{ijk}[t]$, let

$$C[k] = \prod_{m=3}^{M} P(S_t^{(m)} = s_t^{(m)} \mid S_{t-1}^{(m)} = s_{t-1}^{(m)}, Z_t = k),$$

where $s_{t-1}^{(m)}$, $s_t^{(m)}$ for $m \geq 3$ denote the current values of the fixed chains $\mu_3, ..., \mu_M$. Now two cases have to be considered. If $k \geq 3$, chains $1, 2$ cannot have changed state, and

$$\delta_{ijk}[t] = \max_{k'=1,\dots,M} \delta_{ijk'}[t-1] d_{k'k} b_{sy}^{(k)} C[k]$$

with $s = S_t^{(k)}$ and $y = Y_t$. This quantity can be computed in time $O(M)$. If $k \in \{1, 2\}$, we have to take into account state changes on the chains being optimized. Assume without loss of generality that $k = 1$. Now

$$\delta_{ij1}[t] = \max_{k'=1,\dots,M} \max_{i'=1,\dots,K} \delta_{i'jk'}[t-1] d_{k'1} a_{i'i}^{(1)} b_{iy}^{(1)} C[1],$$

with $y = Y_t$. This quantity can be computed in time $O(KM)$. There are $O(K^2MT)$ values of the form $\delta_{ijk}[t]$ to compute. However, time for computing all values is bounded by $O(K^2M(M + K)T)$, as the case $k \in \{1, 2\}$ only appears $O(K^2T)$ times.

The maximum probability of a hidden state configuration is

$$\max_{\mathbf{s}^{(1)},\mathbf{s}^{(2)},\mathbf{z}} P(\mathbf{s},\mathbf{z},\mathbf{y}) = \max_{ijk} \delta_{ijk}[T],$$

and a maximizing configuration is found by keeping track of where maxima occur in backtrace variables.

It is instructive to compare the complexity of the outlined chainwise Viterbi algorithm to inference in an HMM where hidden states are "flattened" into a single process. This HMM has a state space of size KM, and standard Viterbi has thus complexity $O(K^2M^2T)$, similar to the $O(K^2M(M+K)T)$ for a single update step in chainwise Viterbi. However, several such update steps will be needed before convergence.

8.3.3 Parameter Estimation

As for the other models discussed in the this thesis, an interesting question is how we can learn the proposed interleaved mixture of Markov chains from data. As for normal hidden Markov models, we assume that the basic structure of the domain is known, that is, we know how many processes exist and what their hidden states are. The task then becomes to learn the parameters of the model from data:

Problem 8.3.2 (Parameter Learning for Interleaved Mixtures of Hidden Markov Models)**.**

Given

- *a number M of hidden processes;*
- *for every $m \in \{1, ..., M\}$ the number of states K_m of the m-th process*[3]*;*
- *a set $S = \{s_1, ..., s_N\}$ of observation sequences;*

Find

$$\arg\max_{\Theta} P(S \mid \Theta) = \prod_{i=1}^{N} P(s_i \mid \Theta)$$

where Θ is the set of all parameters in the processes $\mu_1, ..., \mu_M$ and the switching process $\bar{\mu}$, and $P(s_i \mid \Theta)$ is the likelihood of the observation sequence s_i under the model with parameters Θ as defined by Equation (8.3).

There are different possible settings for this parameter learning problem. In the activity recognition setting discussed in Section 8.4, both sensor observations and activities are given for the training set. That is, the observed sequences $s_1, ..., s_M$ contain not only sensor observations but also the corresponding (normally hidden) activity information. In this fully observable case, maximum-likelihood model parameters can essentially

[3]Note that here we slightly generalize the model to allow for differently sized state spaces in the different processes

be determined by counting. More generally, if the interleaving is known for the training data (that is, we know which part of each sequence has been generated by which process), the problem reduces to independently estimating the parameters of $\mu_1, ..., \mu_M$ with the standard Baum-Welch algorithm (Rabiner, 1989). In an unsupervised learning setting, expectation-maximization including the unknown interleaving $\mathbf{Z}$ is a natural choice. However, for the same reasons as discussed in Section 8.3, exact computation of the expectation step will be infeasible. In factorial hidden Markov models, this problem is solved elegantly by a structured variational approximation, and exploring variational inference methods for the interleaved mixture model presented in this chapter is an interesting direction for future work. A simple alternative is to employ *hard EM*: instead of computing exact expectations, hidden states are set to their max-likelihood values given the observations, and expectations determined by counting. Together with the chainwise Viterbi algorithm discussed in Section 8.3.2 this yields a tractable method which is straightforward to implement.

8.4 Experimental Evaluation

The proposed model has been evaluated in an activity of daily living (ADL) recognition domain, where the goal is to infer a user's activity from a stream of dense RFID sensor data. The dataset has been collected in a real RFID environment at Intel Research Seattle[4]. Objects are equipped with small RFID tags, and the user is wearing a lightweight RFID reader in a bracelet around the wrist. Whenever the reader comes close (10–15 centimeters) to a tagged object, the object tag is recorded. The sequence of observed tags thus indicates the objects a user has been interacting with while performing the activity.

We recorded activities involved in making breakfast at home, as this domain showcases the kind of interleaving behavior we are interested in (cf. Figure 8.1). The dataset consists of 20 sequences of RFID tag observations collected from 5 different persons having breakfast. Sequences are hand-labeled with the true current activity based on a human observer. There are 18 basic activities organized into 6 high-level activities, 24 different classes of tagged objects (including *nil* if no object was observed), and a total of 4597 timepoints to be classified. Timepoints at which no activity is taking place and activities with a coverage of less than 1% were removed, leaving 14 activities and 3545 timepoints in the dataset. The average number of segments into which a high-level activity is broken up because of interleaving is 3.95. There is significant overlap between observations associated with different activities, either because the same object is used in different activities or noise in the sensor data. More specifically, the average overlap in the set of observations associated with two different activities is 40.6%.

A standard approach in ADL recognition is based on HMMs: each basic activity corresponds to a hidden state, and sensor data to observations. In the described domain this means that all activities are "flattened" into one hidden process, and their hierarchical

[4] `http://www.intel-research.net/seattle/`

Table 8.1: Average cross-validated accuracy for MAJORITY, MAJORITY/OBSERVATION, HMM, HMMMIX and HMMMIX* on the ADL dataset. • indicates that result for HMMMIX is significantly better than result for other method (paired two-sided t-test, $p = 0.05$).

Method	**Accuracy**
MAJORITY	21.2 ± 25.4•
MAJORITY/OBSERVATION	71.4 ± 10.3•
HMM	84.0 ± 9.8•
HMMMIX	$\mathbf{86.0} \pm 8.6$
HMMMIX*	$\mathbf{86.0} \pm 8.6$

structure is lost. This approach will serve as a baseline, denoted by HMM. Alternatively, high-level activities can be modeled as separate hidden processes using the model described in Section 8.2. Here we consider a slight extension of this model: state transition probabilities in the active hidden process μ_{Z_t} depend not only on the previous state but also on whether or not the process has just become active; that is, $Z_t \neq Z_{t-1}$. The motivation for this extension is that high-level activities are typically interrupted at a point where the basic activity changes as well. It is straightforward to generalize the model and algorithms discussed in Section 8.2 and Section 8.3 to include this dependency.

Each high-level activity A is represented as a process μ_A, and the state space of μ_A are the basic activities associated with A. Note that the method, when applied to a given observation sequence, will automatically chose the (approximately) most likely subset of high-level activities that explains the observations. This model, together with the approximate inference technique discussed in Section 8.3.2 will be denoted as HMMMIX. In the chainwise Viterbi algorithm, hidden states are initialized to the most likely activity given the current sensor observation (as observed in the training data). Furthermore, a version with exact inference (denoted HMMMIX*) is run for comparison.

The experimental study seeks to answer the following two questions:

(Q8.1) Does reconstruction accuracy increase if high-level activities are modeled as separate processes?

(Q8.2) Does the approximate inference algorithm for HMMMIX yield results similar to exact inference?

The rationale behind **(Q8.1)** is that modeling high-level activities as separate processes will capture transition dynamics more concisely, as it decouples dynamics within a high-level activity from the switching dynamics. This is reflected in the number of model parameters: The "flattened" HMM representation requires $O((MK)^2) = O(M^2K^2)$ parameters to specify transition dynamics, while HMMMIX only requires $O(M^2+MK^2)$ parameters.

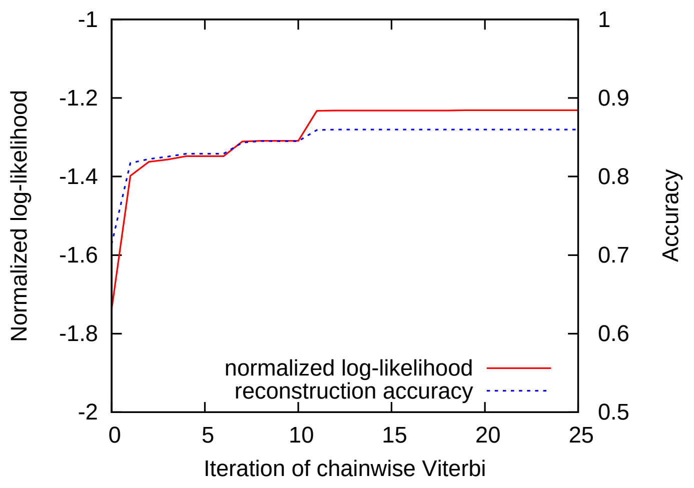


Figure 8.5: Normalized log-likelihood and reconstruction accuracy of the current hidden state configuration as a function of the number of iterations in chainwise Viterbi. Results are averaged over all test sequences.

To evaluate the different approaches, we performed a leave-one-sequence-out cross-validation. On the respective training set, models are estimated from fully observable training data, that is, information on both sensor observations and activities is available. Given a test sequence, the most likely joint state of hidden variables in the model is determined, yielding a prediction of the current basic activity at every point in time. This is compared to the known true activity, and average prediction accuracy is computed. Table 8.1 shows reconstruction accuracy for HMM, HMMMIX and HMMMIX*. Additionally, accuracy for always predicting the most frequent activity (MAJORITY), and the most frequent activity given a particular sensor observation (MAJORITY/OBSERVATION) are shown. HMMMIX significantly outperforms HMM (paired two-sided t-test, $p = 0.05$), and predictions made by HMMMIX and HMMMIX* are identical in this experiment. This affirmatively answers questions **(Q8.1)** and **(Q8.2)**. Figure 8.5 shows the convergence behavior of chainwise Viterbi. The normalized log-likelihood of the current configuration of hidden states and the reconstruction accuracy given by this configuration are plotted as a function of the algorithm iteration. As expected, both likelihood and accuracy increase as the algorithm repeatedly revises the current interleaving. Furthermore, convergence occurs after a small number of iterations.

There are two sources of information for predicting the activity at a point t in time: the current sensor observation, and transition dynamics for activities (which capture the influence of past and future observations on the current prediction). The MAJORITY/OBSERVATION approach already performs well; this indicates that much information is obtained simply from the current sensor observation. To further investigate the influence of

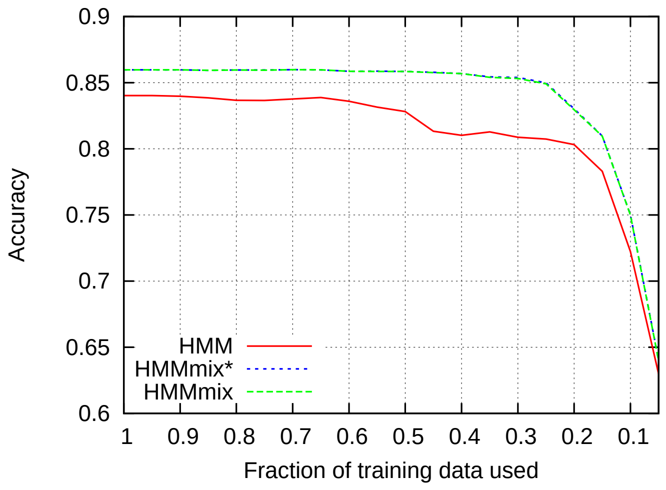


Figure 8.6: Reconstruction accuracy as a function of the fraction γ of training sequences used to estimate model transition parameters, while emission probabilities are estimated from all available training data. Results are averaged over 5 runs of cross-validation.

transition dynamics on reconstruction accuracy, the following experiment was carried out. When estimating a model from data, only a randomly selected fraction γ of the training sequences is used to estimate transition probabilities, while all available data is used to estimate emission probabilities. Figure 8.6 shows reconstruction accuracy as a function of γ. The experiment confirms that HMMMIX outperforms HMM, and that approximate inference gives solutions very close to those of exact inference (solutions differ slightly, but the curves for HMMMIX and HMMMIX* in Figure 8.6 are indistinguishable). Moreover, the difference between HMM and HMMMIX is most pronounced if only 20% to 40% of training sequences are used to estimate transition parameters. This supports the hypothesis that the more concise representation of transition dynamics in HMMMIX (with fewer model parameters) explains its superior performance, as a concise representation matters most if training data is sparse.

8.5 Related and Future Work

Although there exists a large body of related work, to the best of our knowledge the particular setting of data generated from interleaved processes has not been addressed before. A simplified version has been discussed in (Batu *et al.*, 2004); however, tractable inference algorithms are not explored in this work. Simplicial mixtures of Markov chains, which employ a generative semantics similar to latent Dirichlet allocation, also address a similar problem (Girolami and Kabán, 2003). However, they restrict the constituent processes to be Markov rather than hidden Markov. A further class of models assumes several hidden processes that run in parallel, and that observations stem from their joint state. Examples include *factorial hidden Markov models* (Ghahramani and Jordan, 1997), *hidden Markov decision trees* (Jordan *et al.*, 1996), *coupled hidden Markov models* (Brand, 1997) and *mixed hidden Markov models* (Altman, 2007). In contrast to our approach, these models focus on factorizing complex state spaces into cross-products of simpler components, rather than modeling interleaved processes. Another related technique are *switching state-space models* (SSSMs) (Ghahramani and Hinton, 1998), in which several processes run in parallel and an additional *switch* variable selects one *active* process from which the current observation is generated. SSSMs are different in that processes run concurrently, while an interleaving of processes is characterized by the fact that an inactive process is stopped and only resumes when it becomes active again. This creates additional dependencies between processes which cannot be modeled in a SSSM. Finally, *hierarchical hidden Markov models* model hierarchical structure within the hidden process that generates the observations (Fine *et al.*, 1998). However, the component processes cannot interleave, and thus the model is not appropriate in our domain.

As a structured probabilistic model the presented approach is more generally also related to sequential SRL techniques such as LoHMMs (Kersting *et al.*, 2006) or the r-gram and CPT-L model presented in Chapter 5 and Chapter 6. In particular, there exist extensions of LoHMMs to model hierarchical processes which have been applied to similar activity recognition problems (Natarajan *et al.*, 2008). These Logical Hierarchical Hidden Markov Models (LoHiHMMs) focus on modeling hierarchies in activities (such as goal-subgoal decomposition), and perform inference based on particle filters. In contrast to the approach presented in this chapter, LoHiHMMs are a relatively general SRL formalism in that they are still directly based on first-order logic, while our model has been grounded to a particular application domain.

Furthermore, related work includes activity recognition approaches based on HMMs such as (Patterson *et al.*, 2005) and (Zhang *et al.*, 2007) or dynamic Bayesian networks (Wang *et al.*, 2007). The proposed method not only labels sequence positions but returns a structured parse of the sequence in terms of a set of hidden processes. Thus, it is also related to segmentation models, grammar-based approaches, and more generally models for predicting structured data (see Bakir *et al.*, 2007, for an overview).

More generally, the approach presented in this chapter is also related to another approach we developed to solve activity recognition problems, namely *relational transformation-based tagging* (Landwehr *et al.*, 2007c, 2008). Relational transformation-based tagging employs inductive logic programming principles to upgrade the transformation-based tagging algorithm of Brill (1995) to the relational case. In contrast to the method described in this chapter, the approach employs a fully relational data representation, making it possible to supply background knowledge to the learner. However, it does not make use of statistical modeling techniques. For a more detailed discussion of relational tagging in activity recognition, see (Landwehr *et al.*, 2007c, 2008).

There are two main directions for future work. One is to consider semi- and unsupervised learning settings, in which only a few or no activity labels are available at the time of learning. As discussed in Section 8.3.3, this scenario could be addressed using either a “hard” EM approach that is relatively straightforward to implement, or, more ambitiously, using variational inference method. Another interesting direction for future work is to apply the proposed model to other application domains that exhibit a similar interleaving structure as the activity recognition domain considered in this chapter.

Conclusions Part III

Part III of the thesis has discussed two specific application problems, namely haplotype reconstruction and activity recognition. Both of these domains contain significant structure; traditional propositional models such as hidden Markov models are thus not easily applicable. Expressive and general SRL frameworks could be used to model this structure. However, we have instead presented two domain-specific structured probabilistic models, which are both based on the hidden Markov model framework. Compared to more general SRL frameworks, the resulting systems are less widely applicable but more efficient at solving the particular task encountered in the application domain. They can thus be seen as one way of "downgrading" more general SRL frameworks towards more efficiency at the cost of reduced generality. Nevertheless, we also showed how the two application domains can be modeled using logical hidden Markov models (LoHMMs), an expressive and general SRL framework for sequential data (Kersting *et al.*, 2006). In fact, for the haplotype application we started by designing a LoHMM model, and later translated it into a domain-specific propositional model to improve efficiency.

The haplotyping problem is concerned with reconstructing the hidden phase information in diploid genotype measurements. The proposed method, called SPAMM for sparse Markov modeling, was compared to several other state-of-the-art haplotyping systems on real-world genotype datasets with 60–100 individuals and larger simulated datasets with up to 800 individuals. In the experimental study, the well-established PHASE system was the most accurate, but also computationally most demanding haplotype reconstruction method. SPAMM is slightly less accurate but much faster, and scales well to long marker maps. SPAMM was particularly accurate for datasets with several hundred genotype samples. As large datasets are ultimately needed for successful disease association studies, the presented method is a promising alternative to existing approaches.

The activity recognition problem is concerned with inferring a user's activity from a stream of RFID sensor observations indicating the objects the user has been interacting with. The key challenge in the considered activity recognition scenario is that activities are hierarchically structured and interleave in time. Consequently, we proposed a structured model, which explicitly takes activity hierarchies and interleaving into account. An empirical evaluation using real-world activity data confirmed that the model is superior to an approach based on flat (propositional) HMMs. Note that the proposed structured probabilistic model should be applicable not only in activity recognition, but in all situations where only the interleaved output of several independent processes can be observed. Several other potential application domains can be identified. Consider, for instance, a log of web server requests, and assume we have no definite knowledge about which request has been issued by which user (for example, because of proxy use). Clearly, there is no single hidden Markov process that accounts for the sequence of observed requests. Instead, there are multiple processes, one per user, which interleave to generate the sequence of observations. Other application areas in which interleaving occurs are intrusion detection scenarios, in which legitimate network traffic is interleaved with a series of harmful requests, or intron/exon detection in gene identification.

Finale

Chapter 9

Summary and Future Work

This chapter summarizes the main contributions presented in the thesis, draws some conclusions, and discusses important directions for future work.

9.1 Thesis Summary

Statistical Relational Learning (SRL) is a relatively young research area at the intersection of probabilistic modeling, relational and logic-based representations, and machine learning. Interest in SRL has continuously increased, as machine learning techniques are increasingly being applied in application domains where data is complex, structured, and heterogeneous. Examples for such domains include areas in bioinformatics that are concerned with structured entities such as molecules, proteins, or pathways; networked data from the world wide web; user data collected in scenarios where different (human or artificial) agents interact; or relational databases which are widely used in business domains. Accordingly, the development of effective SRL techniques has been identified as one of the most important challenges in machine learning and artificial intelligence (Getoor and Taskar, 2007; De Raedt *et al.*, 2008; Kersting, 2006).

SRL frameworks are often developed by combining relational learning techniques, such as inductive logic programming, with elements from statistical modeling. Relational learning has been successfully employed in many application domains, for example in bio- and chemoinformatics (Bratko and Muggleton, 1995). Advantages of relational learning include the ability to model complex, structured domains; the ability to incorporate background knowledge; and the interpretability of the resulting models. However, relational learning approaches do not handle noise and uncertainty as effectively as statistical modeling techniques. By combining statistical and relational learning techniques, SRL promises to combine the expressivity and interpretability of relational representations with the robustness and accuracy of statistical learners.

Unfortunately, such combinations also combine the complexity of statistical and relational representations when learning models from data. As an example, consider structure learning in knowledge-based model construction approaches (see Chapter 2). Here, learning requires a search through relational model structures as in relational learning, but in order to evaluate a structure in a principled way a statistical learning problem such as likelihood maximization needs to be solved. Depending on the domain and model space, this can become infeasible. If it is, the structure needs to be pre-defined by the user, or must be obtained from a different learning method that is run independently of the actual statistical-relational learner, as in (Richardson and Domingos, 2004). In general, there is a trade-off involved between the expressivity and generality of a formalism and the efficiency of performing inference and learning, as a higher expressivity entails a larger search space for these problems. Propositional learning techniques, which are typically efficient but limited in their representational power, are at one extreme of this trade-off. SRL systems that try to combine the full power of statistical and relational models are at the other extreme.

This thesis has explored statistical relational learning techniques of limited expressivity, which occupy an intermediate position in the outlined expressivity-efficiency trade-off. The resulting systems have been shown to be both efficient and effective at solving challenging problems in a variety of application domains. Specifically, the idea has been explored in three different settings: 1) relational classification problems, 2) relational sequence models, and 3) embedded SRL methods that are tailored to a particular application domain. We now briefly summarize the key contributions from each of these three parts.

In Part I, *dynamic propositionalization* approaches for relational classification problems were introduced. Propositionalization is one of the simplest and most restricted approaches to statistical relational learning (Kramer and De Raedt, 2001). In contrast to traditional (static) propositionalization approaches, dynamic propositionalization tightly couples relational learning with statistical modeling, leading to the selection of better rule sets. Dynamic propositionalization has been explored based on probabilistic graphical models (in the NFOIL system, Chapter 3) and kernel methods (in the KFOIL system, Chapter 4). For both approaches, we have presented *incremental learning algorithms*, which make the systems computationally efficient enough to deal with reasonably large real-world problems. It was shown empirically that dynamic propositionalization outperforms static propositionalization and (non-statistical) relational learning techniques. Dynamic propositionalization can also naturally be applied in a *multi-task* setting, which further increases the accuracy and efficiency of the approach (Chapter 4).

Part II of the thesis has discussed upgrading simple probabilistic sequence models to the relational case. Specifically, we have proposed upgrades of Markov chains to learn from *sequences of logical atoms* and *sequences of logical interpretations*. Markov chains are one of the most basic and most efficient probabilistic sequence models, and thus serve as a natural starting point for developing efficient models for relational sequences. For learning from sequences of atoms, n-gram models were extended to the relational case. In the resulting *r-grams*, smoothed distributions can be obtained by *relational generalization* of the gram, or shortening grams as in the propositional case (Chapter 5). For learning from sequences of interpretations, we introduced the *CPT-L* system, which combines the Markov model framework with *CP-logic* (Chapter 6). CP-logic is a recently proposed causal probabilistic logic, and can naturally define transition distributions for complete logical interpretations. r-grams and the CPT-L model deliberately restrict expressivity by requiring *fully observable data* and employing a *Markov assumption*. This leads to increased computational efficiency compared to more powerful techniques that can model hidden states or non-Markovian processes. For CPT-L we also introduced efficient inference and learning algorithms based on *binary decision diagrams*, which have been shown to scale well to large domain sizes.

Part III of the thesis has explored a different way of trading expressivity for efficiency in SRL, namely by developing *domain-specific* models. The resulting *embedded* SRL systems are based on general statistical relational modeling principles, but tailored to perform a particular task in a particular domain. In this setting, more efficient domain-specific inference and learning algorithms can be developed. Specifically, we have proposed structured probabilistic models for the *haplotype reconstruction* and *activity recognition* domains. Both models are based on the hidden Markov model framework, and their structure can be represented relationally as a *Logical hidden Markov model* (LoHMM) (Kersting *et al.*, 2006). However, the specialization to a particular application domain leads to systems which are much more computationally efficient than general LoHMMs. We have proposed *domain-specific learning and inference* algorithms, such as a structure learning algorithm in the haplotype domain to discover *conserved haplotype fragments* (Chapter 7), and an algorithm for hidden state inference in the context of *interleaved activities* (Chapter 8). The resulting systems were shown to be competitive with established approaches for the respective domain.

To summarize, this thesis has contributed a number of SRL approaches that are focused on computational efficiency. Such approaches should extend the practical applicability of SRL in domains that are computationally too challenging for existing systems. Efficiency was achieved by *restricting* approaches to a particular problem setting or application domain, and developing *special-purpose algorithms* that exploit these restrictions. The resulting systems thus occupy an intermediate position in the expressivity-efficiency trade-off inherent in machine learning systems.

The idea of restricting expressivity was taken furthest in the *embedded* approaches that are tailored to a particular application problem. The resulting models are basically propositional, but their structure is motivated by SRL principles such as the abstraction from ground to first-order states. Such abstractions entail constraints, for instance, on the parameterization of the local probability distributions in the model. The methodology employed here is to see statistical relational learning as a general framework for thinking about domain structure, rather than the science of building ever more powerful systems that can be applied off-the-shelf in any domain. At the same time, general systems are still valuable for demonstrating the general principles underlying a solution, and rapid prototyping. Once the domain structure and how it can modeled is well-understood, the model can—and often should—be specialized to a particular application domain.

As an example for this methodology, consider the haplotyping problem discussed in Chapter 7. We initially modeled this domain using the Logical Hidden Markov Model described in Appendix A.1, before later translating the LoHMM to a propositional model which is orders of magnitude more efficient. Another example is given by graph-mining, tree-mining, and sequence-mining approaches, which specialize more general first-order mining systems to increasingly problem-specific and efficient techniques (Yan and Han, 2002; Zaki, 2002; Li *et al.*, 1998). This is a worthwhile direction because graphs, trees and sequences are (nested) subclasses of general first-order representations that occur in many application areas.

9.2 Future Work

The statistical relational learning systems presented throughout this thesis have mostly been kept deliberately simple in order to emphasize their underlying general principles. Some of the presented methods could be extended by using more advanced combinations of statistical and relational learning techniques, to further increase their effectiveness and efficiency. For instance, the dynamic propositionalization algorithms we have discussed are based on the simple FOIL algorithm. While this has worked remarkably well in our experiments, a more effective search could be obtained in some cases by using advanced ILP search techniques such as bottom-up search (Muggleton and Feng, 1990). More advanced statistical modeling techniques could also be employed. For instance, priors could be incorporated to learn the parameters and/or structure of the tree augmented naïve Bayes model in TFOIL (Chapter 3), and Gaussian or exponential kernels could be used within KFOIL (Chapter 4).

Another natural direction for future work is to consider more general learning settings. For example, for the probabilistic relational sequence models discussed in Part II we assumed that data is fully observable. While this assumption makes inference and learning much easier, it will not be realistic in some domains. It would be interesting to explore how learning from partially observable data can be performed without sacrificing computational efficiency. For the models presented in Part II we also did not discuss the structure learning problem: for r-grams, all grams within a human-defined language bias were employed, and for CPT-L the model structure needed to be pre-defined by the user. How learning can be performed if no pre-defined structure is available is another interesting question to be addressed in future work. Furthermore, we are currently investigating an extension of the CPT-L system in which constraints can be placed on valid interpretations. Note that by declaring certain interpretations invalid, some probability mass is lost, which requires the computation of a normalizing factor. However, this normalization can also be computed efficiently from the binary decision diagram structures used for inference and learning.

For the application-specific models discussed in Part III, an interesting direction for future work is to investigate similarly structured domains in which these models could also be applied. This is particularly appealing for the activity recognition model: it should be applicable whenever the final sequence of observations is generated from several interleaved hidden processes. Such interleaving patterns also appear in a number of other application areas. For instance, in intrusion detection (Lee *et al.*, 1999) and user disambiguation problems (Girolami and Kabán, 2003) data consists of network traffic from different sources that is interleaved in time. Another example is gene finding in bioinformatics, where coding and non-coding regions are interleaved in a DNA sequence (Korf, 2004).

More generally, we plan to further explore the presented methodology of upgrading and downgrading SRL approaches according to the requirements of particular application domains. In this thesis, we have developed relatively specific models for several scenarios. More ambitiously, one could try to develop a general framework for building such specific models: a framework that makes it easy to incorporate, and algorithmically exploit, restrictions that are known to hold in a particular domain. This would require a flexible language for formalizing such restrictions, such that appropriate models and algorithms can be automatically synthesized from a domain description given by the human user. As a less ambitious alternative, one can consider creating a library or toolkit of probabilistic and relational modeling, inference, and learning techniques, which can be used as building blocks to quickly create domain-specific SRL systems.

Finally, much work still needs to be done on applying SRL systems to challenging, and, in particular, large-scale application problems. Only the application of SRL techniques in realistic environments can further clarify how much expressivity and how much efficiency is really needed to effectively solve real-world problems, and thus help to better understand the trade-off between expressivity and efficiency that is at the heart of this thesis.

Appendix

Appendix A

LoHMM Representation of Application-specific Models

This appendix shows how the haplotyping model introduced in the Section 7.2 and the activity recognition model introduced in Section 8.2 can be represented as a logical hidden Markov model.

A.1 LoHMM Representation of Haplotyping Model

The haplotyping model can be represented as follows. In the domain-specific model presented in Figure 7.2, a genotype is sampled by sampling two haplotypes *independently* and *from the same distribution*, cf. Equation (7.3) and Equation (7.4) in Chapter 7. This is the crucial property that needs to be expressed also in the LoHMM model. As the propositional model, the LoHMM is organized as a left-to-right model with layers $t = 1, \ldots, m$. At every layer t, one component of the model encodes the distribution $P(h[t+1] \mid h[t])$. This component is traversed twice for sampling the two new alleles $h^1[t+1], h^2[t+1]$ based on their respective histories $h^1[t], h^2[t]$. Afterwards, the unordered pair corresponding to the new allele pair is emitted.

More specifically, Figure A.1 shows a single layer (at marker t) of the LoHMM model, in the standard automaton notation used for LoHMMs defined by Kersting *et al.* (2006). For sampling two new markers $h^1[t+1], h^2[t+1]$ at position $t+1$ based on the markers $h^1[t], h^2[t]$ at position t, we start at state $m_t(X, Y)$ with $h^1[t], h^2[t]$ bound to X and Y. The model then transitions to the state $s_t(X, Y, x)$ to sample the first new marker $h^1[t+1]$. The multiple transitions from state s_t to state s'_t encode the distribution $P(h[t+1] \mid h[t])$. In $s'_t(A', B, x)$, the new marker $h^1[t+1]$ has been sampled and is bound to A'. Afterwards, the same path is traversed again to sample the second marker, with arguments in state s_t swapped. This effectively samples the new marker $h^2[t+1]$ based on

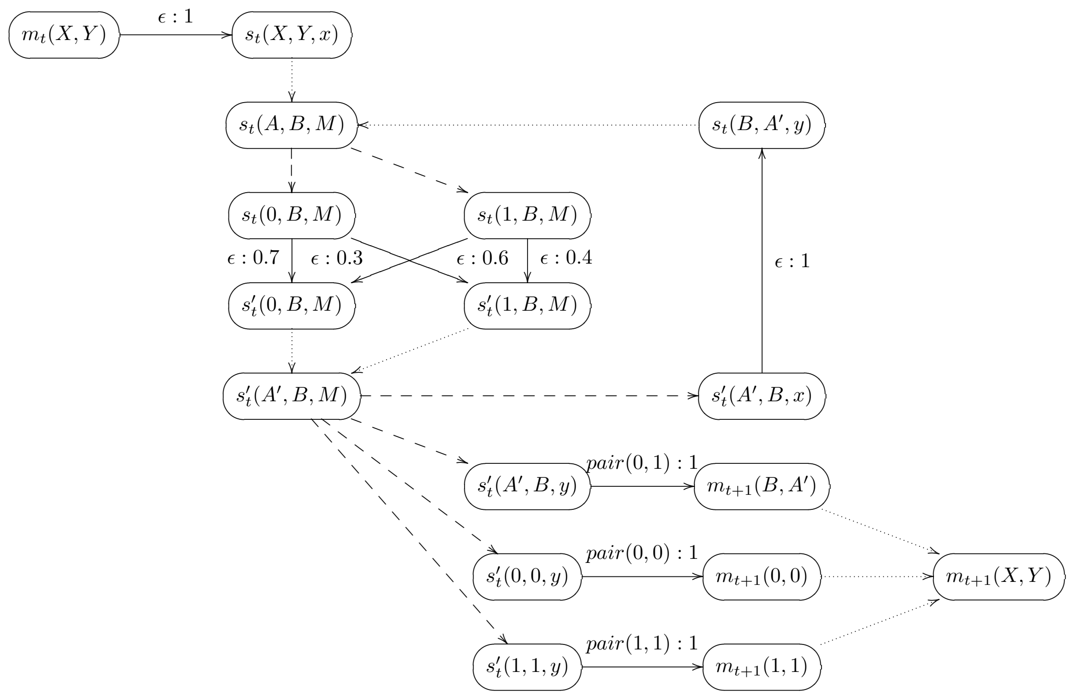


Figure A.1: A LoHMM model for haplotype reconstruction. One layer at marker position t is shown. The standard syntax for visualizing LoHMMs is used: solid arrows represent abstract transitions, dashed arrows the "more general than" relation, and dotted arrows "must follow" links. For a more detailed description of LoHMMs, confer Kersting *et al.* (2006).

$h^2[t]$ independently and from the same distribution as desired. Finally, the unordered pair corresponding to the two new markers is emitted in the transition from s'_t to m_{t+1}. This is can be easily accomplished using the logical generality ordering on abstract states in LoHMMs: if the more specific abstract states for homozygous markers match the ground state a homozygous pair is emitted, otherwise, an (unordered) heterozygous pair. Note that this model only has 2 free parameters per layer, in contrast to a naive first-order HMM model on the the joint state of the two haplotypes, which would have 12 free parameters per layer.

A.2 LoHMM Representation of Activity Recognition Model

The structured probabilistic model for activity recognition presented in Section 8.2 can also be represented using the logical hidden Markov model framework. The model has abstract states of the form $p/(M+1), p_1/(M+1), ..., p_{M+1}/(M+1)$, and $out/(M+2)$. Figure A.2 shows a graphical representation of the model structure for the case $M = 3, K = 2$ in the standard automaton notation used for LoHMMs (for more details see Kersting *et al.*, 2006).

The state $p(S_t^{(1)}, ..., S_t^{(M)}, Z_t)$ represents the current state of the constituent processes and the switching process at time t. From $p(S_t^{(1)}, ..., S_t^{(M)}, Z_t)$ at time t, the model first transitions to a state $p_1(S_t^{(1)}, ..., S_t^{(M)}, Z_{t+1})$ sampling a new state Z_{t+1} of the switching chain based on its history Z_t. Afterwards, in the states $p_2(S_{t+1}^{(1)}, S_t^{(2)}, S_t^{(3)}, Z_{t+1})$, $p_3(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_t^{(3)}, Z_{t+1})$, and $p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_{t+1}^{(3)}, Z_{t+1})$ new states for the constituent processes are sampled. Note that only the chain $a \in \{1, 2, 3\}$ that is active ($Z_{t+1} = a$) actually changes state. Finally, in a transition to the state $out(O_{t+1}, S_{t+1}^{(1)}, ..., S_{t+1}^{(M)}, Z_{t+1})$ the model outputs an observation O_{t+1} based on the currently active process. For ease of exposition, emissions are assumed to be deterministic, but it is straightforward to generalize this part of the model to probabilistic transitions.

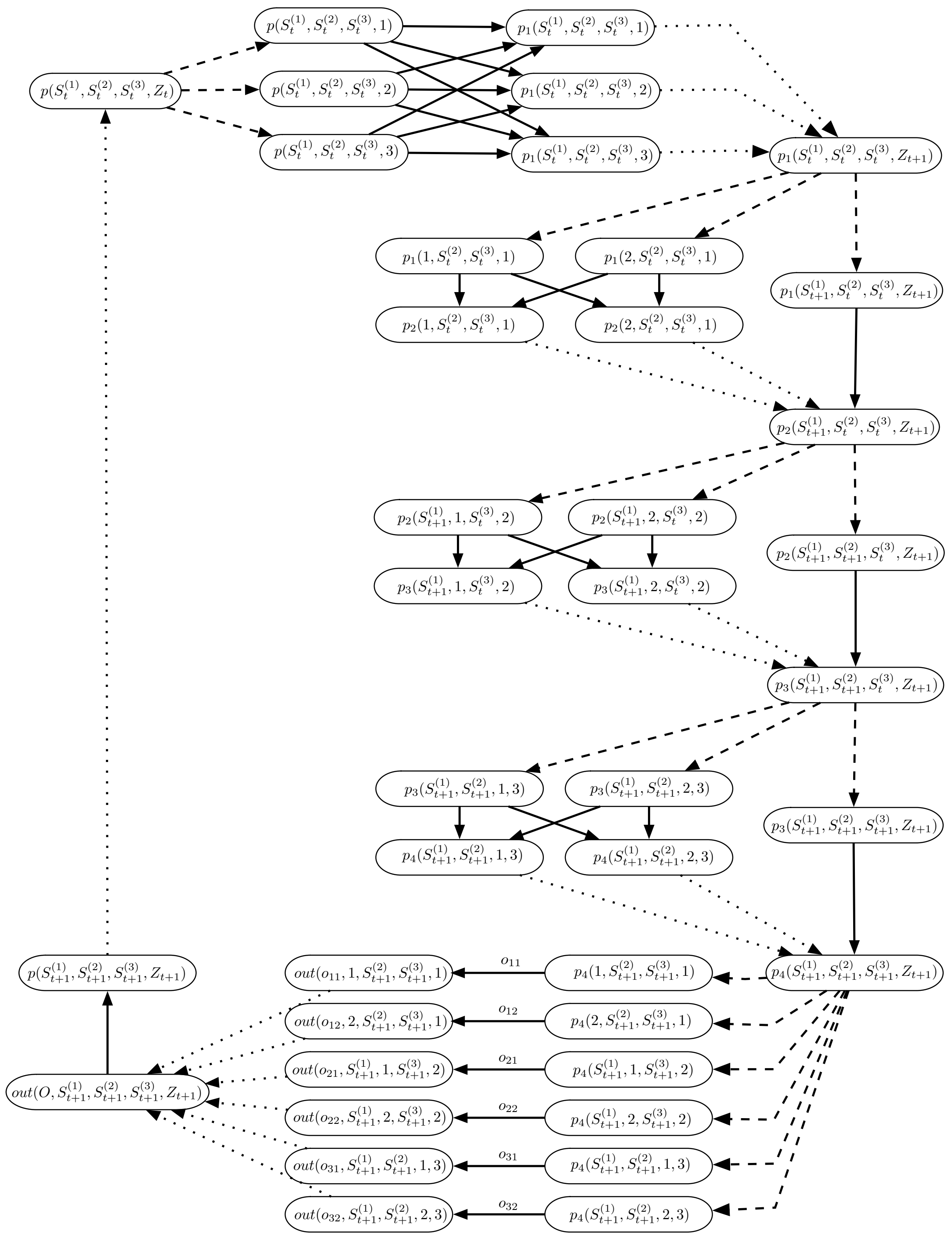

$p(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 1)$
$p(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, Z_t)$
$p(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 2)$
$p(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 3)$
$p_1(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 1)$
$p_1(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 2)$
$p_1(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, 3)$
$p_1(S_t^{(1)}, S_t^{(2)}, S_t^{(3)}, Z_{t+1})$
$p_1(1, S_t^{(2)}, S_t^{(3)}, 1)$
$p_1(2, S_t^{(2)}, S_t^{(3)}, 1)$
$p_1(S_{t+1}^{(1)}, S_t^{(2)}, S_t^{(3)}, Z_{t+1})$
$p_2(1, S_t^{(2)}, S_t^{(3)}, 1)$
$p_2(2, S_t^{(2)}, S_t^{(3)}, 1)$
$p_2(S_{t+1}^{(1)}, S_t^{(2)}, S_t^{(3)}, Z_{t+1})$
$p_2(S_{t+1}^{(1)}, 1, S_t^{(3)}, 2)$
$p_2(S_{t+1}^{(1)}, 2, S_t^{(3)}, 2)$
$p_2(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_t^{(3)}, Z_{t+1})$
$p_3(S_{t+1}^{(1)}, 1, S_t^{(3)}, 2)$
$p_3(S_{t+1}^{(1)}, 2, S_t^{(3)}, 2)$
$p_3(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_t^{(3)}, Z_{t+1})$
$p_3(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 1, 3)$
$p_3(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 2, 3)$
$p_3(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_{t+1}^{(3)}, Z_{t+1})$
$p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 1, 3)$
$p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 2, 3)$
$p(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_{t+1}^{(3)}, Z_{t+1})$
$out(o_{11}, 1, S_{t+1}^{(2)}, S_{t+1}^{(3)}, 1)$
o_{11}
$p_4(1, S_{t+1}^{(2)}, S_{t+1}^{(3)}, 1)$
$p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_{t+1}^{(3)}, Z_{t+1})$
$out(o_{12}, 2, S_{t+1}^{(2)}, S_{t+1}^{(3)}, 1)$
o_{12}
$p_4(2, S_{t+1}^{(2)}, S_{t+1}^{(3)}, 1)$
$out(O, S_{t+1}^{(1)}, S_{t+1}^{(2)}, S_{t+1}^{(3)}, Z_{t+1})$
$out(o_{21}, S_{t+1}^{(1)}, 1, S_{t+1}^{(3)}, 2)$
o_{21}
$p_4(S_{t+1}^{(1)}, 1, S_{t+1}^{(3)}, 2)$
$out(o_{22}, S_{t+1}^{(1)}, 2, S_{t+1}^{(3)}, 2)$
o_{22}
$p_4(S_{t+1}^{(1)}, 2, S_{t+1}^{(3)}, 2)$
$out(o_{31}, S_{t+1}^{(1)}, S_{t+1}^{(2)}, 1, 3)$
o_{31}
$p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 1, 3)$
$out(o_{32}, S_{t+1}^{(1)}, S_{t+1}^{(2)}, 2, 3)$
o_{32}
$p_4(S_{t+1}^{(1)}, S_{t+1}^{(2)}, 2, 3)$

Figure A.2: Logical hidden Markov model representing an interleaved mixture of hidden Markov models (for $M = 3, K = 2$). To simplify notation, one (deterministic) output symbol o_{mi} for state i of process m is assumed. It is straightforward to generalize the model to probabilistic emissions.

Bibliography

Adomavicius, G. and Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, **17**(6), 734–749.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. (1996). Fast Discovery of Association Rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press.

Altman, R. M. (2007). Mixed hidden Markov models: An extension of the hidden Markov model to the longitudinal data setting. *Journal of the American Statistical Association*, **102**, 201–210.

Anderson, C., Domingos, P., and Weld, D. (2002). Relational Markov Models and their Application to Adaptive Web Navigation. In D. Hand, D. Keim, O. Zaïne, and R. Goebel, editors, *Proceedings of the 8th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 143–152. ACM Press.

Argyriou, A., Evgeniou, T., and Pontil, M. (2007). Multi-Task Feature Learning. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 41–48. MIT Press, Cambridge, MA.

Bahr, R. (2004). *Hidden Markov Models: Applications to Financial Econometrics*. Advanced Studies in Theoretical and Applied Econometrics. Springer.

Bakir, G. H., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. V. N. (2007). *Predicting Structured Data*. The MIT Press.

Bao, L. and Intille, S. S. (2004). Activity Recognition from User-Annotated Acceleration Data. In A. Ferscha and F. Mattern, editors, *Pervasive Computing, Second International Conference, 2004, Vienna, Austria, April 21-23, 2004, Proceedings*, volume 3001 of *Lecture Notes in Computer Science*. Springer.

Batu, T., Guha, S., and Kannan, S. (2004). Inferring Mixtures of Markov Chains. In *Proceedings of the 17th Annual Conference on Learning Theory (COLT-2004)*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blockeel, H. and Raedt, L. D. (1998). Top-down Induction of First Order Logical Decision Trees. *Artificial Intelligence*, **101**(1-2), 285–297.

Blockeel, H., De Raedt, L., and Ramon, J. (1998). Top-down Induction of Clustering Trees. In *Proceeding of the 15th International Conference on Machine Learning (ICML-1998)*, Madison, Wisconsin, USA.

Blockeel, H., Dzeroski, S., Kompare, B., Kramer, S., Pfahringer, B., and Laer, W. (2004). Experiments In Predicting Biodegradability. *Applied Artificial Intelligence*, **18**(2), 157–181.

Bordes, A., Ertekin, S., Weston, J., and Bottou, L. (2005). Fast Kernel Classifiers with Online and Active Learning. *Journal of Machine Learning Research*, **6**, 1579–1619.

Brand, M. (1997). Coupled hidden Markov models for modeling interactive processes. Technical Report 405, MIT Media Lab.

Bratko, I. (1990). *Prolog Programming for Artificial Intelligence*. Addison-Wesley. 2nd Edition.

Bratko, I. and Muggleton, S. (1995). Applications of Inductive Logic Programming. *Communications of the ACM*, **38**(11), 65–70.

Brefeld, U. and Scheffer, T. (2005). AUC maximizing support vector learning. In *Workshop on "ROC Analysis in Machine Learning" at the 22nd International Conference on Machine Learning*.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, **21**(4), 543–565.

Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, **35**(8), 677–691.

Caponnetto, A., Micchelli, C., Pontil, M., and Ying, Y. (2008). Universal kernels for multi-task learning. *Journal of Machine Learning Research*, **9**, 1615–1646.

Carter, C. and Catlett, J. (1987). Assesing Credit Card Applications Using Machine Learning. *IEEE Expert*, **2**(3), 71–79.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, **28**(1), 41–75.

Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, **46**(1–3), 131–159.

Chen, H. (1995). Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, **46**, 194–216.

Cortes, C. and Vapnik, V. (1995). Support Vector Networks. *Machine Learning*, **20**, 1–25.

Cothey, V., Aguillo, I., and Arroyo, N. (2006). Operationalising ”Websites”: lexically, semantically or topologically? *International Journal of Scientometrics, Infometrics and Bibliometrics*, **10**.

Crammer, K., Singer, Y., Cristianini, N., Shawe-taylor, J., and Williamson, B. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, **2**, 265–292.

Craven, M. and Slattery, S. (2001). Relational Learning with Statistical Predicate Invention: Better Models for Hypertext. *Machine Learning*, **43**(1–2), 97–119.

Cristianini, N., Shawe-Taylor, J., Elisseef, A., and Kandola, J. (2002). On Kernel-Target Alignment. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*. MIT Press, Cambridge, MA.

Cumby, C. M. and Roth, D. (2003). On Kernel Methods for Relational Learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 107–114, Washington, DC, USA.

Dagum, P. and Chavez, R. (1993). Approximating probabilistic inference in Bayesian belief networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(3), 246–255.

Daly, M., Rioux, J., Schaffner, S., Hudson, T., and Lander, E. (2001). High-Resolution Haplotype Structure in the Human Genome. *Nature Genetics*, **29**, 229–232.

Das, S. K. (1992). *Deductive Databases and Logic Programming*. Addison-Wesley.

Datta, P. and Kibler, D. F. (1993). Concept Sharing: A Means to Improve Multi-Concept Learning. In *Proceedings of the 10th International Conference on Machine Learning*, pages 89–96.

Davis, J., Vítor Santos Costa, I. O., Page, D., and Dutra, I. (2004). Using Bayesian Classifiers to Combine Rules. In *Working Notes of the Third Workshop on Multi-Relational Data Mining (MRDM-2004) in conjunction with the Tenth ACM SIGKDD International Conference on Knowlege Discovery and Data Mining (KDD-2004)*, Seattle, Washington, USA.

Davis, J., Burnside, E., de Castro Dutra, I., Page, D., and Costa, V. S. (2005a). An Integrated Approach to Learning Bayesian Networks of Rules. In J. Gama, R. Camacho,

P. Brazdil, A. Jorge, and L. Torgo, editors, *Proceedings of the Sixteenth European Conference on Machine Learning (ECML-2005)*, volume 3720 of *Lecture Notes in Computer Science*, pages 84–95. Springer.

Davis, J., Burnside, E., Dutra, I., Page, D., Ramakrishnan, R., Costa, V. S., and Shavlik, J. (2005b). View Learning for Statistical Relational Learning: With an Application to Mammography. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*.

Davis, J., Costa, V. S., Ray, S., and Page, D. (2007a). An Integrated Approach to Feature Construction and Model Building for Drug Activity Prediction. In *Proceedings of the 24th International Conference on Machine Learning*.

Davis, J., Ong, I., Struyf, J., Burnside, E., Page, D., and Costa, V. S. (2007b). Change of Representation for Statistical Relational Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer-Verlag.

De Raedt, L. and Bruynooghe, M. (1994). Interactive Theory Revision. In R. Michalski and G. Tecuci, editors, *Machine Learning: a Multistrategy Approach, Volume IV*. Morgan Kaufmann.

De Raedt, L. and Kersting, K. (2003). Probabilistic Logic Learning. *SIGKDD Explorations*, **5**(1), 31–48.

De Raedt, L. and Kersting, K. (2004). Probabilistic Inductive Logic Programming. In S. Ben-David, J. Case, and A. Maruoka, editors, *Proceedings of the Fifteenth International Conference on Algorithmic Learning Theory (ALT-2004)*, volume 3244 of *Lecture Notes in Computer Science*, pages 19–36. Springer.

De Raedt, L. and Ramon, J. (2004). Condensed Representations for Inductive Logic Programming. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning.*

De Raedt, L., Lavrac, N., and Dzeroski, S. (1993). Multiple Predicate Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 1037–1043.

De Raedt, L., Kimmig, A., and Toivonen, H. (2007). ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2462–2467.

De Raedt, L., Frasconi, P., Kersting, K., and Muggleton, S., editors (2008). *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*. Springer.

De Salvo Braz, R., Amir, E., and Roth, D. (2007). Lifted First-Order Probabilistic Inference. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. The MIT press.

Dehaspe, L. (1997). Maximum Entropy Modeling with Clausal Constraints. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming (ILP-1997)*, volume 1297 of *Lecture Notes in Computer Science*, pages 109–124. Springer.

Dehaspe, L., Toivonen, H., and King, R. (1998). Finding Frequent Substructures in Chemical Compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-1998)*, New York City, New York, USA. AAAI Press.

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

Deshpande, A., Milch, B., Zettlemoyer, L., and Kaelbling, L. (2007). Learning Probabilistic Relational Dynamics for Multiple Tasks. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 83–92.

Dietterich, T., Getoor, L., and Murphy, K., editors (2004). *Working Notes of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields (SRL-04)*.

Domingos, P. and Richardson, M. (2004). Markov Logic: A Unifying Framework for Statistical Relational Learning. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields*, pages 49–54.

Duong, T., Phung, D., Bui, H., and Venkatesh, S. (2005). Efficient Coxian Duration Modelling for Activity Recognition in Smart Environments with the Hidden semi-Markov Model. In *Proceedings of the Second Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia*, pages 277–282.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Džeroski, S., Schulze-Kremer, S., Heidtke, K., Siems, K., Wettschereck, D., and Blockeel, H. (1998). Diterpene Structure Elucidation from^{13}C NMR Spectra with Inductive Logic Programming. *Applied Artificial Intelligence, Special Issue on First-Order Knowledge Discovery in Databases*, **12**, 363–383.

Eronen, L., Geerts, F., and Toivonen, H. (2004). A Markov Chain Approach to Reconstruction of Long Haplotypes. In *Pacific Symposium on Biocomputing (PSB-2004)*, pages 104–115.

Eronen, L., Geerts, F., and Toivonen, H. (2006). HaploRec: Efficient and Accurate Large-Scale Reconstruction of Haplotypes. *BMC Bioinformatics*, **7**, 542.

Evgeniou, T., Micchelli, C. A., and Pontil, M. (2005). Learning Multiple Tasks with Kernel Methods. *Journal of Machine Learning Research*, **6**, 615–637.

Fang, H., Tong, W., Shi, L., Blair, R., Perkins, R., Branham, W., Hass, B., Xie, Q., Dial, S., Moland, C., and Sheehan, D. (2001). Structure-Activity Relationships for a Large Diverse Set of Natural, Synthetic, and Environmental Estrogens. *Chemical Research in Toxicology*, **14**(3), 280–294.

Fawcett, T. (2003). ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Technical Report HPL-2003-4, Hewlett Packard Laboratories.

Fawcett, T. and Provost, F. (1997). Adaptive Fraud Detection. *Data Mining and Knowledge Discovery*, **1**(3), 291–316.

Fayyad, U. and Uthurusamy, R. (2002). Evolving data into mining solutions for insights. *Commun. ACM*, **45**(8), 28–31.

Fern, A. (2005). A Simple-Transition Model for Relational Sequences. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK*, pages 696–701.

Fern, A., Getoor, L., and Milch, B., editors (2006). *ICML Workshop on Open Problems in Statistical Relational Learning*, Pittsburgh, PA, USA.

Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden Markov model: analysis and applications. *Machine Learning*, **32**, 41–62.

Flach, P. (1994). *Simply Logical: Intelligent Reasoning by Example*. John Wiley.

Flach, P. and Lachiche, N. (2004). Naive Bayesian classification of structured data. *Machine Learning*, **57**(3), 233–269.

Frasconi, P., Passerini, A., Muggleton, S., and Lodhi, H. (2005). Declarative Kernels. In S. Kramer and B. Pfahringer, editors, *Inductive Logic Programming, 15th International Conference, ILP 2 005, Late-Breaking Papers*, pages 17–19.

Frasconi, P., Jaeger, M., and Passerini, A. (2008). Feature Discovery with Type Extension Trees. In *Proceedings of the 18th International Conference on Inductive Logic Programming*.

Friedman, N. and Goldszmidt, M. (1996). Building Classifiers Using Bayesian Networks. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-1996), Vol. 2*, pages 1277–1284, Portland, Oregon, USA. AAAI Press / The MIT Press.

Garey, M. R. and Johnson, D. S. (1975). Complexity Results for Multiprocessor Scheduling under Resource Constraints. *SIAM Journal Comp.*, **4**(4), 397–411.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, **5**(1), 49–58.

Gärtner, T., Lloyd, J., and Flach, P. (2004). Kernels and Distances for Structured Data. *Machine Learning*, **57**(3), 205–232.

Getoor, L. (2003). Link mining: a new data mining challenge. *SIGKDD Explorations*, **5**(1), 84 – 89.

Getoor, L. and Jensen, D., editors (2000). *Working Notes of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data (SRL-00).*

Getoor, L. and Jensen, D., editors (2003). *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03).*

Getoor, L. and Taskar, B., editors (2007). *Statistical Relational Learning*. MIT press.

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (2001). Learning Probabilistic Relational Models. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*. Springer.

Ghahramani, Z. and Hinton, G. E. (1998). Switching State-Space Models. Technical report, Department of Computer Science, University of Toronto.

Ghahramani, Z. and Jordan, M. I. (1997). Factorial Hidden Markov Models. *Machine Learning*, **29**(2-3), 245–273.

Girolami, M. and Kabán, A. (2003). Simplicial Mixtures of Markov Chains: Distributed Modelling of Dynamic User Profiles. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems*.

Gray, A. and Haahr, M. (2004). Personalised, collaborative spam filtering. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS-2004).*

Greenberg, S. (1988). Using Unix: Collected Traces of 168 Users. Technical report, Department of Computer Science, University of Calgary, Alberta.

Grossman, D. and Domingos, P. (2004). Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-2004)*, pages 361–368, Banff, Canada. ACM Press.

Halldórsson, B., Bafna, V., Edwards, N., Lippert, R., Yooseph, S., and Istrail, S. (2004). A Survey of Computational Methods for Determining Haplotypes. In *Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *Lecture Notes in Computer Science*, pages 26–47.

Han, J. and Kamber, M. (2001). *Data Mining: Concepts and Techniques*. Academic Press.

Harville, M. and Li, D. (2004). Fast, Integrated Person Tracking and Activity Recognition with Plan-View Templates from a Single Stereo Camera. In *Proceedings of the 2nd IEEE Computer Society Conference on Computer VIsion and Pattern Recognition*.

Hubbard, T., Murzin, A., Brenner, S., and Chotia, C. (1997). *SCOP*: a Structural Classification of Proteins Database. *Nucleic Acids Res.*, **27**(1), 236–239.

Hudson, R. (2002). Generating Samples Under a Wright-Fisher Neutral Model of Genetic Variation. *Bioinformatics*, **18**, 337–338.

Huynh, T. and Mooney, R. (2008). Discriminative structure and parameter learning for Markov logic networks. In *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 416–423.

Jacobs, N. and Blockeel, H. (2003). User Modeling with Sequential Data. In *Proceedings of the 10th Conference on Human Computer Interaction*, pages 557–561.

Jaeger, M. (1997). Relational Bayesian Networks. In D. Geiger and P. Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 266–273, Providence, Rhode Island, USA. Morgan Kaufmann.

Jebara, T. (2004). Multi-task feature and kernel selection for SVMs. In *Proceedings of the twenty-first international conference on Machine learning*, page 55, New York, NY, USA. ACM.

Joachims, T. (1999). Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning*, pages 169–184.

Jordan, M. I. (1999). An Introduction to Variational Methods for Graphical Models. *Machine Learning*, **37**, 183–233.

Jordan, M. I. and Weiss, Y. (2002). Graphical models: probabilistic inference. In M. Arbib, editor, *Handbook of neural networks and brain theory*. MIT Press.

Jordan, M. I., Ghahramani, Z., and Saul, L. K. (1996). Hidden Markov Decision Trees. In *Proceedings of the 9th Conference on Advances in Neural Information Processing Systems*.

Kääriäinen, M., Landwehr, N., Lappalainen, S., and Mielikäinen, T. (2007). Combining haplotypers. Technical Report Technical Report C-2007-57, University of Helsinki, Department of Computer Science.

Karwath, A., Kersting, K., and Landwehr, N. (2008). Boosting relational sequence alignments (short paper). In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008)*.

Kaufman, L. (1999). Solving the quadratic programming problem arising in support vector classification. In *Advances in kernel methods: support vector learning*, pages 147–167. MIT Press.

Kersting, K. (2006). An inductive logic programming approach to statistical relational learning: Thesis. *AI Commun.*, **19**(4), 389–390.

Kersting, K. and De Raedt, L. (2001). Bayesian Logic Programs. Technical Report 151, University of Freiburg, Institute for Computer Science.

Kersting, K. and Gärtner, T. (2004). Fisher Kernels for Logical Sequences. In *Proceedings of the 15th ECML*, volume 3201 of *Lecture Notes in Computer Science*, pages 205–216. Springer.

Kersting, K. and Raedt, L. D. (2007). Bayesian Logic Programming: Theory and Tool. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. The MIT Press.

Kersting, K., De Raedt, L., and Raiko, T. (2006). Logical Hidden Markov Models. *Journal of Artificial Intelligence Research*, **25**, 425–456.

Khan, K., Muggleton, S., and Parson, R. (1998). Repeat Learning Using Predicate Invention. In *Proceedings of the 8th International Workshop on Inductive Logic Programming, Madison, Wisconsin, USA*, volume 1446 of *Lecture Notes in Computer Science*, pages 165–174.

Kimmel, G. and Shamir, R. (2005). A Block-Free Hidden Markov Model for Genotypes and Its Applications to Disease Association. *Journal of Computational Biology*, **12**(10), 1243–1259.

King, R., Srinivasan, A., and Sternberg, M. (1995). Relating Chemical Activity to Structure: an Examination of ILP Successes. *New Generation Computing*, **13**(2,4), 411–433.

Kirsten, M., Wrobel, S., and Horváth, T. (2001). Distance based approaches to relational learning and clustering. In *Relational Data Mining*, pages 213–230. Springer.

Kok, S. and Domingos, P. (2005). Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448, New York, NY, USA. ACM.

Koller, D. (1999). Probabilistic Relational Models. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 3–13. Springer-Verlag. Invited paper.

Korf, I. (2004). Gene finding in novel genomes. *BMC Bioinformatics*, **5**(59), 1471–2105.

Kramer, S. and De Raedt, L. (2001). Feature Construction with Version Spaces for Biochemical Applications. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 258–265. Morgan Kaufmann Publishers Inc.

Kramer, S., Lavrac, N., and Flach, P. (2001). Propositionalization Approaches to Relational Data Mining. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag.

Krogel, M. and Wrobel, S. (2001). Transformation-based Learning Using Multirelational Aggregation. In C. Rouveirol and M. Sebag, editors, *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, volume 2157 of *Lecture Notes in Computer Science*. Springer.

Lachiche, N. and Flach, P. A. (2003). 1BC2: a true first-order Bayesian classifier. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 133–148. Springer-Verlag.

Laird, J. E. and van Lent, M. (2000). Human-Level AI's Killer Application: Interactive Computer Games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*.

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., and Jordan, M. I. (2004). Learning the Kernel Matrix with Semidefinite Programming. *Journal of Machine Learning Research*, **5**, 27–72.

Landwehr, N. (2008). Modeling interleaved hidden processes. In *Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 520–527. ACM.

Landwehr, N. and De Raedt, L. (2007). r-grams: Relational Grams. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007), Hyderabad, India, January 6-12, 2007*, pages 907–912.

Landwehr, N. and Mielikäinen, T. (2008). Probabilistic Logic Learning from Haplotype Data. In L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Computer Science*, pages 263–286. Springer.

Landwehr, N., Hall, M., and Frank, E. (2003). Logistic Model Trees. In *Proceedings of the 14th European Conference on Machine Learning (ECML-03)*.

Landwehr, N., Hall, M., and Frank, E. (2005a). Logistic Model Trees. *Machine Learning*, **59**(1-2), 161–205.

Landwehr, N., Kersting, K., and De Raedt, L. (2005b). nFOIL: Integrating Naive Bayes and FOIL. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 795–800.

Landwehr, N., Mielikäinen, T., Landwehr, L. N., Mielikäinen, T., Eronen, L., Toivonen, H., and Mannila, H. (2006a). Constrained Hidden Markov Models for Population-based Haplotyping (Extended Abstract). In *Proceedings of the Workshop on Probabilistic Modeling and Machine Learning in Structural and Systems Biology (PMSB)*.

Landwehr, N., Passerini, A., De Raedt, L., and Frasconi, P. (2006b). kFOIL: Learning Simple Relational Kernels. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence July 16-20, 2006, Boston, Massachusetts, USA*.

Landwehr, N., Mielikäinen, T., Eronen, L., Toivonen, H., and Mannila, H. (2007a). Constrained Hidden Markov Models for Population-based Haplotyping. *BMC Bioinformatics*, **8 (Suppl 2)**.

Landwehr, N., Kersting, K., and De Raedt, L. (2007b). Integrating Naïve Bayes and FOIL. *Journal of Machine Learning Research*, **8**, 481–507.

Landwehr, N., Gutmann, B., Thon, I., Philipose, M., and De Raedt, L. (2007c). Relational Transformation-based Tagging for Human Activity Recognition. In *Working Notes of the 6th Workshop on Multi-Relational Data Mining (MRDM-07) at ECML/PKDD-07*.

Landwehr, N., Gutmann, B., Thon, I., Philipose, M., and De Raedt, L. (2008). Relational Transformation-based Tagging for Activity Recognition. *Fundamenta Informaticae*, **89**, 111–129.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society*, **50**(2), 157–224.

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

Lavrač, N. and Džeroski, S. (1992). Background knowledge and declarative bias in inductive concept learning. In K. Jantke, editor, *Proceedings 3rd International Workshop on Analogical and Inductive Inference*, pages 51–71. Springer-Verlag. (Invited paper).

Lavrač, N., Džeroski, S., and Grobelnik, M. (1991). Learning Nonrecursive Definitions of Relations with LINUS. In Y. Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 265–281. Springer-Verlag.

Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, **8**(1), 98–113.

Lee, S. and De Raedt, L. (2003). Mining Logical Sequences Using SeqLog. In *Database Technology for Data Mining*, volume 2682 of *Lecture Notes in Computer Science*. Springer.

Lee, W., Stolfo, S., and Mok, K. (1999). A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132.

Leslie, C. S., Eskin, E., and Noble, W. S. (2002). The Spectrum Kernel: A String Kernel for SVM Protein Classification. In *Pacific Symposium on Biocomputing, Lihue, Hawaii, USA*, pages 566–575.

Li, C., Yu, P., and Castelli, V. (1998). MALM: a framework for mining sequence database at multiple abstraction levels. In *Proceedings of the 7th International Conference on Information and Knowledge Management (ICIKM-1998)*, pages 267–272.

Liang, P. and Jordan, M. I. (2008). An Asymptotic Analysis of Generative, Discriminative, and Pseudolikelihood Estimators. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*.

Liao, L., Fox, D., and Kautz, H. (2005a). Hierarchical conditional random fields for GPS-based activity recognition. In *Proceeding of the 12th International Symposium of Robotics Research (ISRR 2005). Springer Verlag*. Springer.

Liao, L., Fox, D., and Kautz, H. (2005b). Location-based Activity Recognition using Relational Markov Networks. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 773–778.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *The Journal of Machine Learning Research*, **2**, 419–444.

Macskassy, S. and Provost, F. (2007). Classification in Networked Data: A Toolkit and a Univariate Case Study. *Journal of Machine Learning Research*, **8**, 935–983.

Magoulas, G. D. and Prentza, A. (2001). Machine Learning in Medical Applications. In *Machine Learning and Its Applications*, volume 2049 of *Lecture Notes in Computer Science*, pages 300–307. Springer.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.

Micchelli, C., Xu, Y., and Zhang, H. (2006). Universal Kernels. *The Journal of Machine Learning Research*, **6**, 2651–2667.

Micchelli, C. A. and Pontil, M. (2005). Learning the Kernel Function via Regularization. *Journal of Machine Learning Research*, **6**, 1099–1125.

Minato, S. (1993). Zero-suppressed BDDs for set manipulation in combinatorial problems. In *DAC '93: Proceedings of the 30th international conference on Design automation*, pages 272–277, New York, NY, USA. ACM.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Moore, A. (1990). Efficient Memory-based Learning for Robot Control. Technical report, Computer Laboratory, University of Cambridge, Cambridge, UK.

Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing, Special Issue on Inductive Logic Programming*, **13**, 245–286.

Muggleton, S. (1996). Stochastic logic programs. In L. D. Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press.

Muggleton, S. (2000). Learning stochastic logic programs. *Electronic Transactions in Artificial Intelligence*, **4**(041).

Muggleton, S. and De Raedt, L. (1994). Inductive Logic Programming : Theory and Methods. *Journal of Logic Programming*, **19**, 629–679.

Muggleton, S. and Feng, C. (1990). Efficient Induction of Logic Programs. In *Proceedings of the First Conference on Algorithmic Learning Theory (ALT-1990)*, pages 368–381, Tokyo, Japan. Springer.

Muggleton, S., Amini, A., and Sternberg, M. (2005). Support Vector Inductive Logic Programming. In *Proceedings of the 8th International Conference on Discovery Science*, pages 163–175.

Nallapati, R. (2004). Discriminative models for information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 64–71. ACM.

Natarajan, S., Bui, H. H., Tadepalli, P., and Kersting, K. (2008). Logical hierarchical hidden markov models for modeling user activities. In *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-2008)*.

Ng, A. Y. and Jordan, M. I. (2001). On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems 14*.

Obozinski, G., Taskar, B., and Jordan, M. (June, 2006). Multi-task feature selection. Technical report, Dept. of Statistics, UC Berkeley.

Ong, C. S., Smola, A. J., and Williamson, R. C. (2005). Learning the Kernel with Hyperkernels. *Journal of Machine Learning Research*, **6**, 1043–1071.

Orr, J., R., Abowd, and D., G. (2000). The smart floor: a mechanism for natural user identification and tracking. In *Proceedings of the 2nd ACM CHI 2000 Conference on Human Factors in Computing Systems*, pages 275–276.

Passerini, A., Frasconi, P., and De Raedt, L. (2006). Kernels on Prolog Proof Trees: Statistical Learning in the ILP Setting. *Journal of Machine Learning Research*, **7**, 307–342.

Patterson, D., Fox, D., Kautz, H., and Philipose, M. (2005). Fine-Grained Activity Recognition by Aggregating Abstract Object Usage. In *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, Osaka.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Perlich, C. and Provost, F. (2006). Distribution-based Aggregation for Relational Learning with Identifier Attributes. *Machine Learning*, **62**, 65–105.

Platt, J. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical Report MSR-TR-98-14, Microsoft Research.

Platt, J. (1999). Probabilities for SV Machines. *Advances in Neural Information Processing Systems 12*, pages 61–74.

Pollack, M. E. (2005). Intelligent Technology for an Aging Population: The Use of AI to Assist Elders with Cognitive Impairment. *AI Magazine*, **26**(2), 9–24.

Pomerleau, D. A. (1989). Alvinn: An Autonomous Land Vehicle in a Neural Network. Technical Report CMU-CS-89-107, Pittsburgh, PA: Carnegie Mellon University.

Pompe, U. and Kononenko, I. (1995). Naive Bayesian Classifier within ILP-R. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming (ILP-1995)*, pages 417–436, Tokyo, Japan.

Pompe, U. and Kononenko, I. (1997). Probabilistic First-Order Classification. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming (ILP-1997)*, volume 1297 of *Lecture Notes in Computer Science*, pages 235–242. Springer.

Poole, D. (1997). The Independent Choice Logic for Modelling Multiple Agents Under Uncertainty. *Artificial Intelligence*, **94**(1-2), 7–56.

Popescul, A., Ungar, L., Lawrence, S., and Pennock, D. (2003). Statistical Relational Learning for Document Mining. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM-2003)*, pages 275–282, Melbourne, Florida, USA. IEEE Computer Society.

Provost, F., Fawcett, T., and Kohavi, R. (1998). The Case Against Accuracy Estimation for Comparing Induction Algorithms. In *Proceeding of the Fifteenth International Conference on Machine Learning (ICML-1998)*, Madison, Wisconsin, USA. Morgan Kaufmann.

Quinlan, J. (1990). Learning Logical Definitions from Relations. *Machine Learning*, **5**, 239–266.

Rabiner, L. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, **77**(2), 257–286.

Raento, M., Oulasvirta, A., Petit, R., and Toivonen, H. (2006). ContextPhone - A Prototyping Platform for Context-aware Mobile Applications . *IEEE Pervasive Computing*, **4**(2), 51–59.

Rakotomamonjy, A. (2004). Optimizing area under ROC curve with SVMs. In *First Workshop on ROC Analysis in AI, at the 16th European Conference on Artificial Intelligence*.

Ramon, J. (2002). *Clustering and instance based learning in first order logic*. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

Ramon, J. and Bruynooghe, M. (1998). A Framework for Defining Distances Between First-Order Logic Objects. In *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 271–280.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Rastas, P., Koivisto, M., Mannila, H., and Ukkonen, E. (2005). A Hidden Markov Technique for Haplotype Reconstruction. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI-2005), Mallorca, Spain*, pages 140–151.

Reid, M. D. (2004). Improving Rule Evaluation Using Multitask Learning. In *Proceedings of the 14th International Conference on Inductive Logic Programming, Porto, Portugal*, volume 3194. Springer.

Richardson, M. and Domingos, P. (2004). Markov Logic Networks. Technical report, Dept. Computer Science and Engineering, University of Washington, Seattle.

Richardson, M. and Domingos, P. (2006). Markov Logic Networks. *Machine Learning*, **62**, 107–136.

Riesenhuber, M. and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, **2**, 1019–1025.

Rissanen, J. (1978). Modeling by Shortest Data Description. *Automatica*, **14**, 465–471.

Rückert, U. and Kramer, S. (2007). Margin-Based First-Order Rule Learning. *Machine Learning*, **70**(2-3), 189–206.

Saigo, H., Nowozin, S., Kadowaki, T., Kudo, T., and Tsuda, K. (2009). gBoost: a mathematical programming approach to graph classification and regression. *Machine Learning (to appear).*

Salem, R., Wessel, J., and Schork, N. (2005). A Comprehensive Literature Review of Haplotyping Software and Methods for Use with Unrelated Individuals. *Human Genomics*, **2**, 39–66.

Sato, T. and Kameya, Y. (1997). PRISM: A symbolic-statistical modeling language. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1330–1339. Morgan Kaufmann.

Saul, L. K. and Jordan, M. I. (1999). Mixed Memory Markov Models: Decomposing Complex Stochastic Processes as Mixtures of Simpler Ones. *Machine Learning*, **37**, 75–87.

Scheet, P. and Stephens, M. (2006). A Fast and Flexible Statistical Model for Large-Scale Population Genotype Data: Applications to Inferring Missing Genotypes and Haplotypic Phase. *The American Journal of Human Genetics*, **78**, 629–644.

Semeraro, G., Esposito, F., and Malerba, D. (1995). Ideal Refinement of Datalog Programs. In *Proceedings of the 5th LOPSTR*, volume 1048 of *Lecture Notes in Computer Science*. Springer.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. *Proceedings of the 24th International Conference on Machine Learning, Corvallis, Oregon, USA*, pages 807–814.

Slaney, J. and Thiébaux, S. (2001). Blocks World revisited. *Artificial Intelligence*, **125**(1-2), 119–153.

Slattery, S. and Craven, M. (1998). Combining Statistical and Relational Methods for Learning in Hypertext Domains. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*.

Srinivasan, A., Muggleton, S., King, R., and Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, pages 217–232.

Srinivasan, A., Muggleton, S., King, R., and Sternberg, M. (1996). Theories for Mutagenicity: a Study of First-Order and Feature-Based Induction. *Artificial Intelligence*, **85**, 277–299.

Srinivasan, A., King, R., and Bristol, D. (1999). An Assessment of ILP-Assisted Models for Toxicology and the PTE-3 Experiment. In S. Dzeroski and P. Flach, editors, *Proceedings of the Ninth Internatinal Workshop on Inductive Logic Programming (ILP-1999)*, volume 1634 of *Lecture Notes in Computer Science*. Springer.

Steck, H. (2007). Hinge Rank Loss and the Area Under the ROC Curve. In *Proceedings of the 18th European Conference on Machine Learning, Warsaw, Poland*.

Stephens, M. and Scheet, P. (2005). Accounting for Decay of Linkage Disequilibrium in Haplotype Inference and Missing-Data Imputation. *The American Journal of Human Genetics*, **76**, 449–462.

Swamidass, S. J., Chen, J., Bruand, J., Phung, P., Ralaivola, L., and Baldi, P. (2005). Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, **21**(1), 359–368.

The International HapMap Consortium (2005). A Haplotype Map of the Human Genome. *Nature*, **437**, 1299–1320.

Thon, I., Landwehr, N., and De Raedt, L. (2008). A Simple Model for Sequences of Relational State Descriptions. In *Proceedings of the 19th European Conference on Machine Learning (ECML-2008)*.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer.

Vennekens, J., Denecker, M., and Bruynooghe, M. (2006). Representing Causal Information About a Probabilistic Process. In *Logics In Artificial Intelligence*, volume 4160 of *Lecture Notes in Computer Science*, pages 452–464.

Wachman, G. and Khardon, R. (2007). Learning from interpretations: a rooted kernel for ordered hypergraphs. In *Proceedings of the 24th International Conference on Machine Learning, Corvalis, Oregon*, pages 943–950.

Wang, S., Pentney, W., Popescu, A.-M., Choudhury, T., and Philipose, M. (2007). Common Sense Based Joint Training of Human Activity Recognizers. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*.

Wang, W., Barratt, B., Clayton, D., and Todd, J. (2005). Genome-wide Association Studies: Theoretical and Practical Concerns. *Nature Reviews Genetics*, **6**, 109–118.

Wellmann, M. (1992). From Knowledge Bases to Decision Models. *Knowledge Engineering Review*, **7**, 35–53.

Wheeler, D. L. L., Barrett, T., Benson, D. A. A., Bryant, S. H. H., Canese, K., Chetvernin, V., Church, D. M. M., Dicuccio, M., Edgar, R., Federhen, S., Feolo, M., Geer, L. Y. Y., Helmberg, W., Kapustin, Y., Khovayko, O., Landsman, D., Lipman, D. J. J., Madden, T. L. L., Maglott, D. R. R., Miller, V., Ostell, J., Pruitt, K. D. D., Schuler, G. D. D., Shumway, M., Sequeira, E., Sherry, S. T. T., Sirotkin, K., Souvorov, A., Starchenko, G., Tatusov, R. L. L., Tatusova, T. A. A., Wagner, L., and Yaschenko, E. (2008). Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res*, **33**, 39–45.

Winters-Hilt, S. (2006). Hidden Markov Model Variants and their Application. *BMC Bioinformatics*, **7 (Suppl. 2)**.

Witten, I. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implemenations*. Morgan Kaufmann.

Wu, H. D. D. and Vapnik, V. (1999). Support Vector Machines for Spam Categorization. *IEEE Transactions on Neural Networks*, **10**(5), 1048–1054.

Yan, X. and Han, J. (2002). gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM-2002)*.

Zaki, M. (2002). Efficiently mining frequent trees in a forest. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pages 71–80.

Zettlemoyer, L. S., Pasula, H., and Kaelbling, L. P. (2005). Learning Planning Rules in Noisy Stochastic Worlds. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 911–918.

Zhang, W., Chen, F., Xu, W., and Cao, Z. (2007). Decomposition in hidden Markov models for activity recognition. In *Multimedia Content Analysis and Mining*, volume 4577 of *Lecture Notes in Computer Science*.

Publication List

Journal Articles

- N. Landwehr, B. Gutmann, I. Thon, M. Philipose, and L. De Raedt. *Relational Transformation-based Tagging for Activity Recognition*. Fundamenta Informaticae 89, pp. 111-129, 2008.

- N. Landwehr, T. Mielikäinen, L. Eronen, H. Toivonen, and H. Mannila. *Constrained Hidden Markov Models for Population-based Haplotyping*. BMC Bioinformatics 8 (Suppl. 2) : S9, 2007.

- N. Landwehr, K. Kersting, and L. De Raedt. *Integrating Naïve Bayes and FOIL*. Journal of Machine Learning Research 8, pp. 481-507, 2007.

- N. Landwehr, M. Hall, and E. Frank. *Logistic Model Trees*. Machine Learning 59 (1-2), pp. 161-205, 2005.

Conferences and Workshops, Published in Proceedings

- A. Karwath, K. Kersting and N. Landwehr. *Boosting Relational Sequence Alignments (Short Paper)*. In Proceedings of the 8th IEEE International Conference on Data Mining (ICDM-2008), Pisa, Italy, 2008.

- I. Thon, N. Landwehr and L. De Raedt. *A Simple Model for Sequences of Relational State Descriptions*. In Proceedings of the 19th European Conference on Machine Learning (ECML-2008), Antwerp, Belgium, 2008, volume 5212 of Lecture Notes in Computer Science, pp. 506–521, Springer.
  Shorter versions of this paper also appeared at the 6th International Workshop on Mining and Learning with Graphs (MLG-2008), and in the Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-2008, Late Breaking Papers).

- N. Landwehr. *Modeling Interleaved Hidden Processes*. In Proceedings of the 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland, 2008.
- N. Landwehr, B. Gutmann, I. Thon, M. Philipose, and L. De Raedt. *Relational Transformation-based Tagging for Human Activity Recognition*. In Proceedings of the 6th Workshop on Multi-Relational Data Mining (MRDM-2007) at the 18th European Conference on Machine Learning (ECML-2007), Warsaw, Poland.
  A different version of this paper also appeared in the Proceedings of the International Workshop on Knowledge Discovery from Ubiquitous Data Streams (IWK-DUDS) at ECML-2007.
- N. Landwehr and L. De Raedt. *r-grams: Relational Grams*. In Proceedings of the Twentieth Joint International Conference on Artificial Intelligence (IJCAI-2007), Hyderabad, India, 2007.
- N. Landwehr, A. Passerini, L. De Raedt and P. Frasconi. *kFOIL: Learning Simple Relational Kernels*. In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006), Boston, Massachusetts, USA, 2006.
  A shorter version of this paper also appeared in the Proceedings of the 16th International Conference on Inductive Logic Programming (ILP-2006, Short Papers).
- N. Landwehr, T. Mielikäinen, L. Eronen, H. Toivonen, and H. Mannila. *Constrained Hidden Markov Models for Population-based Haplotyping (Extended Abstract)*. In Proceedings of the Workshop on Probabilistic Modeling and Machine Learning in Structural and Systems Biology, Tuusula, Finland, 2006.
- N. Landwehr, K. Kersting, and L. De Raedt. *nFOIL: Integrating naïve Bayes and FOIL*. In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005), Pittsburgh, Pennsylvania, USA, 2005.
  A shorter version of this paper also appeared in the Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-2005, Late Breaking Papers).
- N. Landwehr, M. Hall, and E. Frank. *Logistic Model Trees*. In Proceedings of the 14th European Conference on Machine Learning (ECML-2003), Dubrovnik, Croatia, 2003, volume 2837 of Lecture Notes in Computer Science, pp. 241–252, Springer.

Book Chapters

- N. Landwehr and T. Mielikäinen. *Probabilistic Logic Learning from Haplotype Data*. In L. De Raedt, P. Frasconi, K. Kersting and S. Muggleton (Eds.): Probabilistic Inductive Logic Programming, Springer, 2008.

- K. Kersting, L. De Raedt, B. Gutmann, A. Karwath and N. Landwehr. *Relational Sequence Learning*. In L. De Raedt, P. Frasconi, K. Kersting and S. Muggleton (Eds.): Probabilistic Inductive Logic Programming, Springer, 2008.

- K. Kersting and N. Landwehr. *Scaled Conjugate Gradients for Maximum Likelihood: An Empirical Comparison with the EM Algorithm.* In J. A. Gmez, S. Moral and A. Salmern (Eds.): Advances in Learning Bayesian Networks, Springer, 2004.

Technical Reports

- M. Kääriänen, N. Landwehr, S. Lappalainen and T. Mielikäinen. *Combining Haplotypers*, Technical Report C-2007-57, University of Helsinki, Department of Computer Science, 2007.

Biography

Niels Landwehr was born on February 22, 1977 in Kiel, Germany. He went to school at the Freie Waldorfschule in Kiel, from where he graduated (with "Abitur") in July 1996. After studying mathematics at the Christian-Albrechts-University in Kiel for one semester, and one year of compulsory civil service, he started studying computer science at the Albert-Ludwigs-University in Freiburg, Germany, in October 1998. In October 2000 he obtained a "Vordiplom", and in July 2003 a "Diplom" in computer science from this university. Afterwards, he spent 6 months as an intern in a software company in Buenos Aires, supported by a grant from "Capacity Building International", Germany, a human resources development organization. In June 2004 he joined the machine learning group at the Albert-Ludwigs-University Freiburg and started work on a Ph.D. in the area of statistical relational learning, supervised by Luc De Raedt. In April 2007 he moved with Luc De Raedt to the DTAI (Declarative Talen en Artificiële Intelligentie) group at the Katholieke Universiteit Leuven, Belgium. In February 2009 he will defend his Ph.D. thesis on "Trading Expressivity for Efficiency in Statistical Relational Learning" at the Katholieke Universiteit Leuven.