1. Learning Probabilistic Relational Models

Lise Getoor[1], Nir Friedman[2], Daphne Koller[1], and Avi Pfeffer[3]

[1] Computer Science Department, Stanford University
Stanford, CA 94305-9010, USA

[2] The School of Computer Science & Engineering, Hebrew University
Jerusalem 91904, Israel

[3] Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138, USA

Abstract

Probabilistic relational models (PRMs) are a language for describing statistical models over typed relational domains. A PRM models the uncertainty over the attributes of objects in the domain and uncertainty over the relations between the objects. The model specifies, for each attribute of an object, its (probabilistic) dependence on other attributes of that object and on attributes of related objects. The dependence model is defined at the level of *classes* of objects. The class dependence model is instantiated for any object in the class, as appropriate to the particular context of the object (i.e., the relations between this objects and others). PRMs can also represent uncertainty over the relational structure itself, e.g., by specifying a (class-level) probability that two objects will be related to each other. PRMs provide a foundation for dealing with the noise and uncertainty encountered in most real-world domains. In this chapter, we show that the compact and natural representation of PRMs allows them to be learned directly from an existing relational database using well-founded statistical techniques. In this chapter, we give an introduction to PRMs and an overview of methods for learning them. We show that PRMs provide a new framework for relational data mining, and offer new challenges for the endeavor of learning relational models for real-world domains.

1.1 Introduction

Relational models are the most common representation of structured data. Enterprise business information, marketing and sales data, medical records, and scientific datasets are all stored in relational databases. Efforts to extract knowledge from partially structured (e.g., XML) or even raw text data also aim to extract relational information. Recently, there has been growing interest in extracting interesting statistical patterns from these huge amounts of data. These patterns provide a deeper understanding of the domain and the relationships in it. In addition, extracted patterns can be used for reaching conclusions about important attributes whose values may be unobserved.

Probabilistic graphical models, and particularly Bayesian networks , have been shown to be a useful way of representing statistical patterns in real-world domains. Recent work [1.5, 1.11] develops techniques for learning these models directly from data, and shows that interesting patterns often emerge in this learning process. However, all of these learning techniques apply only to flat-file representations of the data, and not to the richer relational data encountered in many applications. An obvious solution is to take a relational database and "flatten" it, creating a flat file on which standard Bayesian network learning algorithms can be run. As we discuss in Section 1.7, this approach has several important shortcomings.

Probabilistic relational models (PRMs) are a recent development [1.15, 1.20, 1.23] that extend the standard attribute-based Bayesian network representation to incorporate a much richer relational structure. These models allow the specification of a probability model for *classes* of objects rather than simple attributes; they also allow properties of an entity to depend probabilistically on properties of other *related* entities. The probabilistic class model represents a generic dependence, which is then instantiated for specific circumstances, i.e., for particular sets of entities and relations between them.

We have developed methods for learning PRMs directly from structured data such as relational databases. The two key tasks in the construction of any statistical model are model selection and parameter estimation. We have developed algorithms for each of these tasks. These algorithms are based on the same underlying principles that govern the learning of Bayesian networks.

A learned PRM provides a statistical model that can uncover and discover many interesting probabilistic dependencies that hold in a domain. Unlike a set of (probabilistic) rules for classification, PRMs specify a joint distribution over a relational domain. Thus, like Bayesian networks, they can be used for answering queries about any aspect of the domain given any set of observations. Furthermore, rather than trying to predict one particular attribute, the PRM learning algorithm attempts to tease out the most significant direct dependencies in the data. The resulting model thus provides a high-level, qualitative picture of the structure of the domain, in addition to the quantitative information provided by the probability distribution. Thus, PRMs are ideally suited to exploratory analysis of a domain and relational data mining.

This chapter is structured as follows. We begin by briefly surveying some of the foundations on which PRMs are built: In the next two sections, we provide some basic background on probabilistic models and relational models. In Section 1.4, we define PRMs and give their semantics. In Section 1.5 we describe how to learn a PRM from an existing database. In Section 1.6 we describe experimental results for several real-world domains. We conclude with a discussion of related work and briefly mention some promising directions for future work.

1.2 Probabilistic models

The traditional logic-based approach to representing knowledge is to write down a knowledge base in the form of logical axioms about the domain. The knowledge base restricts the set of possible worlds, or models, to those consistent with the axioms. Additional facts — those that are true in all of these possible worlds — are logically entailed by the knowledge base.

In a standard logical framework, we are restricted to representing only facts that are true absolutely. Thus, this framework is unable to represent and reason with uncertain and noisy information. This is a significant gap in the expressive power of the framework, and a major barrier to its use in many real-world applications. Uncertainty is unavoidable in the real world: our information is often inaccurate and always incomplete, and only a few of the "rules" that we use for reasoning are true in all (or even most) of the possible cases.

This limitation, which is critical in many domains (e.g., medical diagnosis), has led over the last decade to the resurgence of probabilistic reasoning in artificial intelligence. Probability theory models uncertainty by assigning a probability to each of the states of the world that an agent considers possible. Most commonly in probabilistic reasoning, these states are the set of possible assignments of values to a set of *attributes* or *random variables.* Consider, for example, a simple model of the performance of a student in a course. There are six random variables: *Intelligence*, *Difficulty* (of the course), *Good Test Taker*, *Understands Material*, *Exam Grade* and *Homework Grade*. Of these variables, *Intelligence*, *Good Test Taker*, and *Understands Material* are boolean variables, *Difficulty* takes values from {*low, medium, high*}, and *Exam Grade* and *Homework Grade* take values from $\{A, B, C, D, F\}$. The possible worlds are all possible assignments of values to these variables, 600 $(2 \times 2 \times 2 \times 3 \times 5 \times 5)$ in this case.

A probabilistic model specifies a joint distribution over all possible worlds. Thus, it specifies implicitly the probability of any event, such as an assignment of values to some subset of variables. Unlike many models, such as a set of rules used for predicting some particular attribute, a probabilistic model is not limited to conclusions about a prespecified set of attributes, but rather can be used to answer queries about any variable or subset of variables. Nor does it require that the values of all other variables be given; it applies in the presence of any evidence. For example, a probabilistic model of a student's performance can be used to predict the distribution over the student's exam grade given his intelligence. As new evidence is obtained, e.g., about his homework grade, conditioning can be used to update this probability, so that the probability of a high exam grade will go up if we observe good homework grades. The same model is used to do the predictive and the evidential reasoning.

Furthermore, a probabilistic model can perform *explaining away*, a reasoning pattern that is very common in human reasoning, but very difficult to obtain in other formal frameworks. Explaining away uses evidence supporting one cause to decrease the probability in another, not because the two are incompatible, but simply because the one cause explains away the evidence, removing the support for the other cause. For example, if we observe that the student's exam grade is high, our belief that she is intelligent will go up. If we then hear that the class is known to be easy, that fact provides an alternative explanation for the student's high grade, reducing our belief that she is intelligent. The same probabilistic model supports all of these reasoning patterns, allowing it to be used in many different tasks.

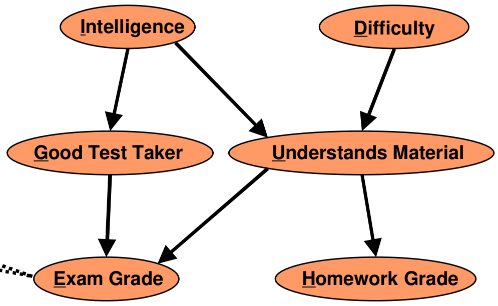
The traditional objection to probabilistic models has been their computational cost. A complete joint probability distribution over a set of random variables must specify a probability for each of the exponentially many different instantiations of the set. Even in our very simple example, we must specify 600 numbers to specify the joint distribution. This type of representation is impractical both from a knowledge engineering perspective, since it is almost impossible for a person to specify an entry in a complex joint distribution, far less to specify an exponential number of them, and from a learning perspective, since the number of samples required to perform robust parameter estimation would be infeasible.

Therefore, a naive representation of the joint distribution is infeasible for all but the simplest domains. *Bayesian networks* [1.21] use the underlying structure of the domain to overcome this problem. The key insight is the locality of influence present in many real-world domains: each variable is directly influenced by only a few others. For example, a student's intelligence induces a better understanding of the material, which in turns leads to a higher homework grade. But the effect of intelligence on homework grade is an indirect one: if the student does not understand the material, her intelligence does not help her get better grades. A Bayesian network captures this insight graphically; it represents the distribution as a directed acyclic graph whose nodes represent the random variables and whose edges represent direct dependencies. Figure 1.1 shows a Bayesian network for our simple student domain.

A Bayesian network has formal semantics in terms of *probabilistic conditional independence*. Formally, the network asserts that each node (or rather the random variable) is conditionally independent of its non-descendants given values for its parents. For example, if we know that the student does not understand the material, our distribution over her grades is no longer influenced by information that we might have about her intelligence.

These conditional independence assumptions allow a very concise representation of the joint probability distribution over these random variables: we associate with each node a *conditional probability distribution* (CPD), which specifies for each node X the probability distribution over the values of X given each combination of values for its parents, denoted $\mathrm{Pa}(X)$. The conditional independence assumptions associated with the Bayesian networks

G	U	P(E | G,U) A	B	C	D	F
t	t	0.7	0.25	0.03	0.01	0.01
t	f	0.3	0.4	0.2	0.05	0.05
f	t	0.4	0.3	0.2	0.08	0.02
f	f	0.05	0.2	0.3	0.3	0.15



$$P(I, D, G, U, E, H) = P(I)P(D)P(G \mid I)P(U \mid I, D)P(E \mid G, U)P(H \mid U)$$

Fig. 1.1. A simple Bayesian network for the student performance domain, the decomposition of the joint distribution into a product of CPDs, and the CPD for one of the nodes in the network

imply that these numbers suffice to uniquely determine the probability distribution over these random variables. More precisely, the joint distribution over all variables can be *factorized* into a product of the CPDs of all the variables via the *Chain Rule for Bayesian Networks*:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathrm{Pa}(X_i)).$$

Consider the Bayesian network for our student performance domain (Figure 1.1). In this model, the student's performance in tests depends on her intelligence. Her understanding of the material depends both on her intelligence and on the difficulty of the class. Her exam grade depends on whether she is a good test taker, and on her understanding of the material, while her homework grade depends on her understanding of the material. The structure of the network encodes a number of conditional independence assertions. For example, the student's exam grade is conditionally independent of her intelligence given her test taking ability and understanding of the material.

These independence assumptions allow us to factor the joint distribution into a product form as shown in Figure 1.1. Each of the conditional probabilities in this product form is a CPD of one of the variables. In this example, the CPD is simply a table, such as the one shown for $P(E \mid G, U)$ in the figure. This CPD shows that if a student is a good test taker and understands the material, then she has probability 0.7 of getting an A on the exam, whereas if the student is a bad test taker and does not understand the material, her probability of getting an A is only 0.05.

Bayesian networks provide a compact representation of complex joint distributions. By modeling an entire joint distribution, Bayesian networks implicitly specify the answer to any probability query, in particular, any query where we want to find the probability distribution over some variables given evidence about any others. In theory, the problem of doing inference in Bayesian networks is NP-hard [1.4]. However, the dependency structure made explicit by the network representation can be exploited by inference algorithms, allowing for efficient inference in practice, even for very large networks.

The semantics and compact representation of Bayesian networks also allow effective statistical learning from data. Standard statistical parameter estimation techniques can be used for learning the parameters of a given network. For learning the structure of the network, typical approaches use a *score* function (which is typically based on Bayesian considerations) to score how different structures "match" the training data. The learning process then reduces to the task of searching for the highest scoring structure [1.11]. These techniques allow a Bayesian network structure to be discovered from data. The learned structure can often give us insight about the nature of the connections between the variables in the domain. Furthermore, the graph structure can sometimes be interpreted causally [1.24], allowing us to induce cause and effect, which can be very useful for understanding our domain, and to reach conclusions about the consequences of intervening (acting) in the domain. Statistical learning techniques are also robust to the presence of missing data and hidden variables. Techniques such as *EM (expectation maximization)* can be used to deal with this issue in the context of parameter estimation [1.16] and have recently been generalized to the harder problem of structure selection [1.7].

Bayesian network learning has been applied successfully to data mining applications. For example, Breese et al. [1.2] show how a Bayesian network can be learned from data describing people's preferences over a variety of items. The learned dependencies correspond to correlations between a person's preference for different items. The resulting Bayesian network can be used for collaborative filtering, and is a better predictor than the standard approaches to this task. In addition to their predictive ability, Bayesian networks have the advantage that they provide a visualization of the most significant direct correlations in the domain, clarifying the domain structure to the user.

1.3 Relational models

Over the last decade, Bayesian networks have been used with great success in a wide variety of real-world and research applications. However, despite their success, Bayesian networks are often inadequate to properly model aspects of complex relational domains. A Bayesian network for a given domain involves a prespecified set of random variables, whose relationship to each other is fixed in advance. Hence, a Bayesian network cannot be used to deal with domains where we might encounter several entities in a variety of configurations. This limitation of Bayesian networks is a direct consequence of the fact that they lack the concept of an "object" (or domain entity). Hence,

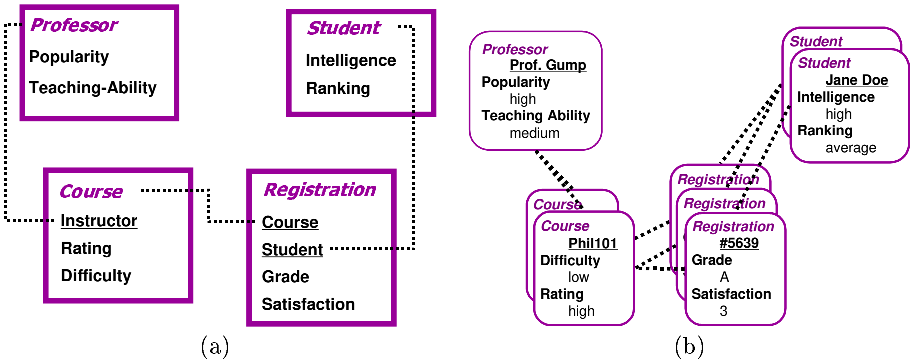


Fig. 1.2. (a) A relational schema for a simple university domain. The underlined attributes are reference slots of the class and the dashed lines indicate the types of objects referenced. (b) An example instance of this schema. Here we do not show the reference slots, we use dashed lines to indicate the relationships that hold between objects.

they cannot represent general principles about multiple similar objects which can then be applied in multiple contexts.

Relational logic, which has traditionally formed the basis for most large-scale knowledge representation systems, addresses these problems. The notions of "individuals", their properties, and the relations between them provide an elegant and expressive framework for reasoning about many diverse domains. The use of quantification allows us to compactly represent general rules, that can be applied in many different situations. For example, when reasoning about genetic transmission of certain properties (e.g., genetically transmitted diseases), we can write down general rules that hold for all people and many properties.[1]

Probabilistic relational models (PRMs) [1.15, 1.22] extend Bayesian networks with the concepts of individuals, their properties, and relations between them. In a way, they are to Bayesian networks as relational logic is to propositional logic. Bayesian networks have a formal semantics in terms of probability distributions over sets of propositional interpretations that are assignments of values to attributes. PRMs have a similar formal semantics in terms of probability distributions over sets of *relational logic* interpretations.

Our relational framework, on which PRMs are based, is derived from the presentation of Friedman et al. [1.8]; it is motivated primarily by the concepts of relational databases, although some of the notation is derived from frame-based and object-oriented systems. However, the framework is a fully general one, and is equivalent to the standard vocabulary and semantics of relational logic.

---

[1] See Chapter 1 and Chapter 3 in this volume for a more extensive discussion of the advantages of relational representations.

A schema for a relational model describes a set of *classes*, $\mathcal{X} = \{X_1, \ldots, X_n\}$. Each class is associated with a set of *descriptive attributes* and a set of *reference slots*. There is a direct mapping between our representation and that of relational databases. Each class corresponds to a single table. Our descriptive attributes correspond to standard attributes in the table, and our reference slots correspond to attributes that are foreign keys (key attributes of another table).

Figure 1.2(a) shows a schema for a simple domain that we will be using as our main running example. The domain is that of a university, and contains professors, students, courses, and course registrations. The classes in the schema are Professor, Student, Course, and Registration.

The set of descriptive attributes of a class X is denoted $\mathcal{A}(X)$. Attribute A of class X is denoted $X.A$, and its space of values is denoted $\mathcal{V}(X.A)$. We assume here that value spaces are finite. For example, the Student class has the descriptive attributes *Intelligence* and *Ranking*. The value space for Student.*Intelligence* might be $\{$*high, low*$\}$.

The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use a similar notation, $X.\rho$, to denote the reference slot ρ of X. Each reference slot ρ is typed, i.e., the schema specifies the range type of object that may be referenced. More formally, for each ρ in X, the *domain type* $\mathrm{Dom}[\rho]$ is X and the *range type* $\mathrm{Range}[\rho]$ is Y for some class Y in $\mathcal{X}$. For example, the class Course has reference slot *Instructor* with range type Professor, and class Registration has reference slots *Course* and *Student*. In Figure 1.2(a) the reference slots are underlined.

For each reference slot ρ, we can define an *inverse slot* ρ^{-1}, which is interpreted as the inverse function of ρ. For example, we can define an inverse slot for the *Student* slot of Registration and call it *Registered-In*. Note that this is not a one-to-one relation, but returns a *set* of Registration objects. Finally, we define the notion of a *slot chain*, which allows us to compose slots, defining functions from objects to other objects to which they are indirectly related. More precisely, we define a *slot chain* $\rho_1, \ldots, \rho_k$ to be a sequence of slots (inverse or otherwise) such that for all i, $\mathrm{Range}[\rho_i] = \mathrm{Dom}[\rho_{i+1}]$. For example, Student.*Registered-In.Course.Instructor* can be used to denote a student's set of instructors.

An *instance* $\mathcal{I}$ of a schema is simply a standard relational logic interpretation of this vocabulary. It specifies: a set of objects x, partitioned into classes; a value for each attribute $x.A$ (in the appropriate domain); and a value for each reference slot $x.\rho$, which is an object in the appropriate range type. We use $\mathcal{A}(x)$ as a shorthand for $\mathcal{A}(X)$, where x is of class X. For each object x in the instance and each of its attributes A, we use $\mathcal{I}_{x.A}$ to denote the value of $x.A$ in $\mathcal{I}$. For example, Figure 1.2(b) shows an instance of the schema from our running example. In this (simple) instance there is one Professor, two Classes, three Registrations, and two Students. The relations between them show that the Professor is the instructor in both classes, and that one student ("Jane Doe") is registered only for one class ("Phil101"), while the other student is registered for both classes.

1.4 Probabilistic relational models

Probabilistic relational models are a new development that integrates the strengths of probabilistic models and relational logic. Several approaches have been proposed, some based on probabilistic logic programming [1.20, 1.23] and others based on a more object-relational framework [1.15]. Our presentation is based on the ideas presented by Koller and Pfeffer [1.15] and follows the presentation of [1.8]. It also accommodates and generalizes the probabilistic logic programming approaches [1.20, 1.23].

1.4.1 Basic language

PRMs provide a language for specifying a probability distribution over a set of relational interpretations. More precisely, a PRM specifies a distribution over a set of instances of a given schema. One might consider PRMs that specify a distribution over all possible instances of the schema, i.e., all possible databases over that schema. This set of databases is infinitely large, as it includes all the possible variations over the number of objects in each class and the possible relations between them. It is clearly very difficult to place a distribution over this type of space, and it is not obvious that such a general-purpose distribution is useful. On the other hand, unlike in the case of Bayesian networks, we want PRMs to be a general model, that can apply to a wide variety of situations. Hence, a PRM is actually a template: given a set of ground objects, a PRM specifies a probability distribution over a set of interpretations involving these objects (and perhaps other objects). We begin with describing the simplest form of PRMs, where the relational structure of the model — the set of objects and the relations between them — is assumed to be part of the input to the template. Only the attributes of the objects participate in the probabilistic model. In Section 1.8 we discuss how to extend this framework to much richer settings.

A *relational skeleton* σ of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects for each class and the relations that hold between the objects. However, it leaves the values of the attributes unspecified. Figure 1.3(a) shows the relational skeleton of the instance shown in Figure 1.2(b). A PRM specifies a probability distributions over *completions* $\mathcal{I}$ of any given skeleton. As we will show, for any skeleton for the schema, the PRM induces a distribution over instances that complete the skeleton.

A PRM specifies the probability distribution using the same underlying principles used in specifying Bayesian networks. The assumption is that each

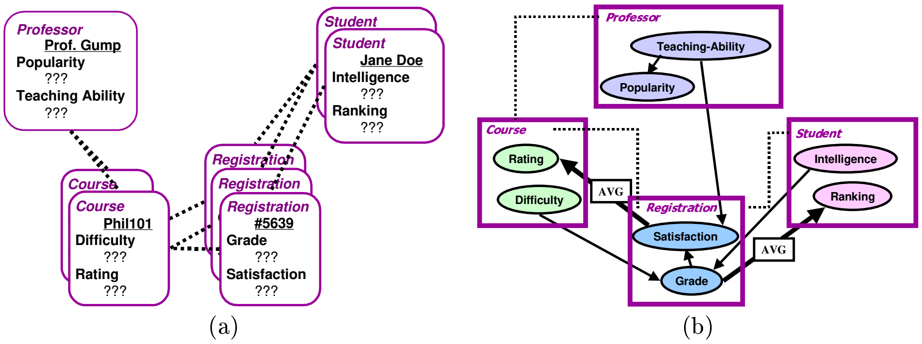


Fig. 1.3. (a) An example relational skeleton for the school domain. Here we know only the objects in our domain and the relationships that hold between them. (b) A PRM structure for the school domain. Edges correspond to probabilistic dependency. Edges from one class to another are routed through slot-chains (chains of references). For clarity these are not explicitly stated in the diagram (see text).

of the random variables in the PRM — in this case the attributes $x.A$ of the individual objects x — is directly influenced by only a few others. The PRM therefore defines for each $x.A$ a set of *parents*, which are the direct influences on it, and a local probabilistic model that specifies the dependence on these parents. However, there are two primary differences between PRMs and Bayesian networks. First, a PRM defines the dependency model at the class level, allowing it to be used for any object in the class. In a sense, the class dependency model is universally quantified and instantiated for every element in the class domain. Second, the PRM explicitly uses the relational structure of the model, in that it allows the probabilistic model of an attribute of an object to depend also on attributes of related objects. The specific set of related objects can vary with the skeleton σ; the PRM specifies the dependency in a generic enough way that it can apply to an arbitrary relational structure.

A PRM consists of two components: the qualitative dependency structure, $\mathcal{S}$, and the set of parameters associated with it, $\theta_{\mathcal{S}}$. Figure 1.3(b) shows an example PRM structure for our school domain. The dependency structure is defined by associating with each attribute $X.A$ a set of *parents* $\mathrm{Pa}(X.A)$. These correspond to *formal* parents; they will be instantiated in different ways for different objects in X. Intuitively, the parents are attributes that are "direct influences" on $X.A$. In Figure 1.3(b), the arrows define the dependency structure.

We distinguish between two types of formal parents. The attribute $X.A$ can depend on another probabilistic attribute B of X. This formal dependence induces a corresponding dependency for individual objects: for any object x in of class X, $x.A$ will depend probabilistically on $x.B$. For example, in Figure 1.3(b), a professor's *Popularity* depends on her *Teaching*

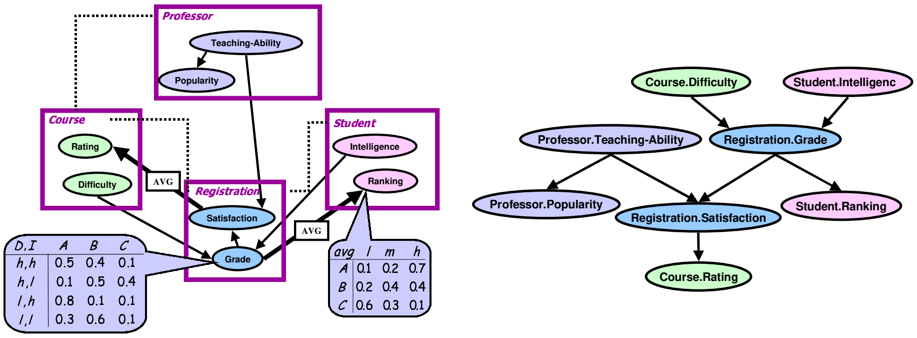
Ability. This dependency model is duplicated for each professor in the skeleton. Thus, we essentially assume that the same probabilistic model applies to all the professors in our domain.

In addition, an attribute $X.A$ can also depend on attributes of related objects $X.\tau.B$, where τ is a slot chain. In Figure 1.3(b), the grade of a student in a course, Registration.*Grade*, depends on Registration.*Student.Intelligence* and Registration.*Course.Difficulty*. The PRM language also allows us to use longer slot chains, for example the dependence of Student.*Satisfaction* on Registration.*Course.Instructor.Teaching-Ability*. Such slot chains are instantiated for each object by following the references that are assigned to it by the skeleton. Thus, for example, for the registration object *#5639*, Registration.*Student.Intelligence* references *Jane-Doe.Intelligence*, and the slot Registration.*Course.Difficulty* references *Phil101.Difficulty*.

Our example PRM also contains a dependence of Student.*Ranking* on Student.*Registered-In.Grade*. Note that a student will typically be registered in several classes; the model specifies a dependence of the student's ranking on the grades that he receives in all of them. In general, $x.\tau$ represents the *set* of objects that are τ-relatives of x. Except in cases where the slot chain is guaranteed to be single-valued, we must specify the probabilistic dependence of $x.A$ on the multiset $\{y.B \ : \ y \in x.\tau\}$. This dependence poses a representational problem, since we need to specify the distribution of $x.A$ given a multiset of values of size 1, 2, 3, and so on. It is clearly impractical to to provide a dependency model for each of the unboundedly many possible multiset sizes.

The notion of *aggregation* from database theory gives us an appropriate tool to address this issue. The dependence of $x.A$ on $x.\tau.B$, is interpreted as a probabilistic dependence of $x.A$ on some (deterministically computed) aggregate property of this multiset. There are many natural and useful notions of aggregation: the mode of the set (most frequently occurring value); mean value of the set (if values are numerical); median, maximum, or minimum (if values are ordered); cardinality of the set; etc. More formally, our language allows a notion of an aggregate γ; γ takes a multiset of values of some ground type, and returns a summary of it. The type of the aggregate can be the same as that of its arguments. However, we allow other types as well, e.g., an aggregate that reports the size of the multiset. More precisely, we allow $X.A$ to have as a parent $\gamma(X.\tau.B)$; the semantics is that for any $x \in X$, $x.A$ will depend on the value of $\gamma(x.\tau.B)$. We define $\mathcal{V}(\gamma(X.\tau.B))$ to be the set of possible values of this aggregate. In our example PRM, there are two aggregate dependencies defined, one that specifies that the ranking of a student depends on the average of his grades and one that specifies that the rating of a course depends on the average satisfaction of students in the course.

As in Bayesian networks, the second component of a PRM is the parameters associated with the qualitative structure. A PRM contains a CPD for



(a) (b)

Fig. 1.4. (a) The CPD for Registration.*Grade* and the CPD for an aggregate dependency of Student.*Rank* on Student.*Registered-In*.*Grade*. (b) The dependency graph for the school PRM.

each attribute of each class. As for the dependencies, we assume that the parameters are shared by each object in the class. We associate with each attribute $X.A$ a CPD $P(X.A \mid \mathrm{Pa}(X.A))$. Figure 1.4(a) shows two CPDs, one for a dependency on a single-valued chain and one for an aggregate dependency. More precisely, let $\mathbf{U}$ be the set of parents $\mathrm{Pa}(X.A)$. Each of these parents U_i — whether a simple attribute or an aggregate — has a set of values $\mathcal{V}(U_i)$ in some ground type. For each tuple of values $\mathbf{u} \in \mathcal{V}(\mathbf{U})$, we specify a distribution $P(X.A \mid \mathbf{u})$ over $\mathcal{V}(X.A)$. We use $\theta_{X.A|\mathbf{u}}$ to denote the parameters of this distribution. The entire set of these parameters, for all $X.A$ and all $\mathbf{u}$, comprises $\theta_{\mathcal{S}}$.

Definition 1: A *probabilistic relational model (PRM)* Π for a relational schema $\mathcal{R}$ is defined as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$, we have:

- a set of *parents* $\mathrm{Pa}(X.A) = \{U_1, \ldots, U_l\}$, where each U_i has the form $X.B$ or $\gamma(X.\tau.B)$, where τ is a slot chain;
- a *conditional probability distribution (CPD)* that represents $P_\Pi(X.A \mid \mathrm{Pa}(X.A))$. ▮

1.4.2 PRM semantics

Given any skeleton, we have a set of random variables of interest: the attributes $x.A$ of the objects in the skeleton. Formally, let $\mathcal{O}^\sigma(X)$ denote the set of objects in skeleton σ whose class in X. The set of random variables for σ is the set of attributes of the form $x.A$ where $x \in \mathcal{O}^\sigma(X_i)$ and $A \in \mathcal{A}(X_i)$ for some class X_i. The PRM specifies a probability distribution over the possible joint assignments of values to these random variables. As with Bayesian networks, the joint distribution over these assignments can be factored. That is, we take the product, over all $x.A$, of the probability in the CPD of the specific value assigned by the instance to the attribute given the values assigned to its parents. Formally, this is written as follows:

$$\begin{aligned} P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) &= \prod_{x \in \sigma} \prod_{A \in \mathcal{A}(x)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \\ &= \prod_{X_i} \prod_{A \in \mathcal{A}(X_i)} \prod_{x \in \mathcal{O}^{\sigma}(X_i)} P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \end{aligned} \tag{1.1}$$

This expression is very similar to the chain rule for Bayesian networks. There are two primary differences. First, our random variables are the attributes of a set of objects. Second, the set of parents of a random variable can vary according to the relational context of the object — the set of objects to which it is related.

As in any definition of this type, we have to take care that the resulting function from instances to numbers does indeed define a *coherent* probability distribution, i.e., where the sum of the probability of all instances is 1 . In Bayesian networks, where the joint probability is also a product of CPDs, this requirement is satisfied if the dependency graph is acyclic: a variable is not an ancestor of itself. A similar condition is sufficient to ensure coherence in PRMs as well. We want to ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. To do so, we can consider the graph of dependencies among attributes of objects in the skeleton. Consider the parents of an attribute $X.A$. When $X.B$ is a parent of $X.A$, we define an edge $x.B \rightarrow_{\sigma} x.A$; when $\gamma(X.\tau.B)$ is a parent of $X.A$ and $y \in x.\tau$, we define an edge $y.B \rightarrow_{\sigma} x.A$. We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to a skeleton σ if the directed graph defined by $\rightarrow_{\sigma}$ over the variables $x.A$ is acyclic. In this case, we are guaranteed that the PRM defines a coherent probabilistic model over complete instantiations $\mathcal{I}$ consistent with σ.

This procedure allows us to check whether a dependency structure $\mathcal{S}$ is acyclic relative to a fixed skeleton σ. However, we often want stronger guarantees: we want to ensure that our dependency structure is acyclic for any skeleton that we are likely to encounter. How do we guarantee this property based only on the class-level PRM? To do so, we consider potential dependencies at the class level. More precisely, we define a *class dependency graph*, which reflects these dependencies [1.15, 1.8]. This class dependency graph has an edge from $Y.B$ to $X.A$ if either: $X = Y$ and $X.B$ is a parent of $X.A$; or $\gamma(X.\tau.B)$ is a parent of $X.A$ and $\mathrm{Range}[X.\tau] = Y$. Figure 1.4(b) shows the dependency graph for our school domain.

The most obvious approach for using the class dependency graph is to simply require that it be acyclic. This requirement is equivalent to assuming a stratification among the attributes of the different classes, and requiring that the parents of an attribute precede it in the stratification ordering. It is clear that if the class dependency graph is acyclic, we can never have that $x.A$

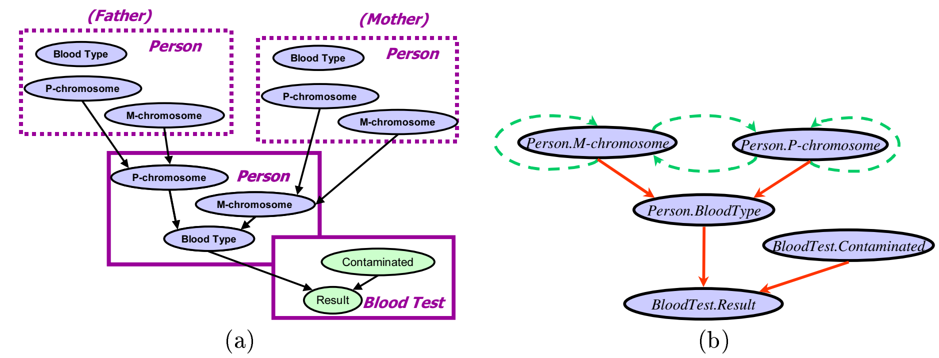


Fig. 1.5. (a) A simple PRM for the genetics domain. (b) the corresponding dependency graph. Dashed edges correspond to "guaranteed acyclic" dependencies.

depends (directly or indirectly) on itself. For example, if we examine the PRM of Figure 1.3(b), we can easily convince ourselves that we cannot create a cycle in any instance. Indeed, as we saw in Figure 1.4(b), the class dependency graph is acyclic. Note, however, that if we introduce additional dependencies we can create cycles. For example, if we make Professor.*Teaching-Ability* depend on the rating of courses she teaches (e.g., if high teaching ratings increase her motivation), then the resulting class dependency graph is cyclic, and there is no stratification order that is consistent with the PRM structure. An inability to stratify the class dependency graph implies that there are skeletons for which the PRM will induce a distribution with cyclic dependencies. In general, however, a cycle in the class dependency graph does not imply that all skeletons induce cyclic dependencies.

While this simple approach clearly ensures acyclicity, it is too limited to cover many important cases. Consider, for example, a simple genetic model of the inheritance of a single gene that determines a person's blood type, shown in Figure 1.5(a). Each person has two copies of the chromosome containing this gene, one inherited from her mother, and one inherited from her father. There is also a possibly contaminated test that attempts to recognize the person's blood type. Our schema contains two classes Person and BloodTest. Class Person has reference slots *Mother* and *Father* and descriptive attributes *Gender*, *P-Chromosome* (the chromosome inherited from the father), and *M-Chromosome* (inherited from the mother). *BloodTest* has a reference slot *Test-Of* that points to the owner of the test, and descriptive attributes *Contaminated* and *Result*.

In our genetic model, the genotype of a person depends on the genotype of his parents; thus, at the class level, we have *Person.P-Chromosome* depending directly on *Person.P-Chromosome*. As we can see in Figure 1.5(b), this dependency results in a cycle that clearly violates the requirements of our simple approach. However, it is clear to us that the dependencies in this model are not actually cyclic for any skeleton that we will encounter in this domain. The reason is that, in "legitimate" skeletons for this schema, a person cannot be his own ancestor, which disallows the situation of the person's genotype depending (directly or indirectly) on itself. In other words, although the model appears to be cyclic at the class level, we know that this cyclicity is always resolved at the level of individual objects.

Our ability to guarantee that the cyclicity is resolved relies on some prior knowledge that we have about the domain. We want to allow the user to give us information such as this, so that we can make stronger guarantees about acyclicity. The user can specify that certain slots are *guaranteed acyclic*. In our genetics example, *Father* and *Mother* are guaranteed acyclic; cycles involving these attributes may in fact be legal. In fact, they are mutually guaranteed acyclic, so that compositions of the slots are also guaranteed acyclic. Figure 1.5(b) shows the class dependency graph for the genetics domain, with guaranteed acyclic edges shown as dashed edges. It turns out that because all of the cycles in this graph contain mutually guaranteed acyclic relations, the structure is legal. In [1.8], we give an algorithm for checking the legality of structures that contain guaranteed acyclic slots and slot chains.

1.5 Learning PRMs

In the previous sections, we defined the PRM language and its semantics. We now move to the task of learning a PRM from data. In the learning problem, our input contains a relational schema, that specifies the basic vocabulary in the domain — the set of classes, the attributes associated with the different classes, and the possible types of relations between objects in the different classes (which simply specifies the mapping between a foreign key in one table and the associated primary key). Our training data consists of a fully specified instance of that schema. We assume that this instance is given in the form of a relational database. Although our approach would also work with other representations (e.g., a set of ground facts completed using the closed world assumption), the efficient querying ability of relational databases is particularly helpful in our framework, and makes it possible to apply our algorithms to large datasets.

There are two variants of the learning task: parameter estimation and structure learning. In the parameter estimation task, we assume that the qualitative dependency structure of the PRM is known; i.e., the input consists of the schema and training database (as above), as well as a qualitative dependency structure $\mathcal{S}$. The learning task is only to fill in the parameters that define the CPDs of the attributes. In the structure learning task, there is no additional required input (although the user can, if available, provide prior knowledge about the structure, e.g., in the form of constraints). The goal is to extract an entire PRM, structure as well as parameters, from the training database alone. We discuss each of these problems in turn.

1.5.1 Parameter Estimation

We begin with the parameter estimation task for a PRM where the dependency structure is known. In other words, we are given the structure $\mathcal{S}$ that determines the set of parents for each attribute, and our task is to learn the parameters $\theta_{\mathcal{S}}$ that define the CPDs for this structure. While this task is relatively straightforward, it is of interest in and of itself. Experience in the setting of Bayesian networks shows that the qualitative dependency structure can be fairly easy to elicit from human experts, in cases where such experts are available. In addition, the parameter estimation task is a crucial component in the structure learning algorithm described in the next section.

The key ingredient in parameter estimation is the *likelihood function*, the probability of the data given the model. This function measures the extent to which the parameters provide a good explanation of the data. Intuitively, the higher the probability of the data given the model, the better the ability of the model to predict the data. The likelihood of a parameter set is defined to be the probability of the data given the model: $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) = P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}})$. As in many cases, it is more convenient to work with the logarithm of this function:

$$\begin{aligned} l(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S}) &= \log P(\mathcal{I} \mid \sigma, \mathcal{S}, \theta_{\mathcal{S}}) \\ &= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \mathcal{O}^{\sigma}(X_i)} \log P(\mathcal{I}_{x.A} \mid \mathcal{I}_{\mathrm{Pa}(x.A)}) \right] . \end{aligned} \tag{1.2}$$

The key insight is that this equation is very similar to the log-likelihood of data given a Bayesian network [1.11]. In fact, it is the likelihood function of the Bayesian network induced by the structure given the skeleton: the network with a random variable for each attribute of each object $x.A$, and the dependency model induced by $\mathcal{S}$ and σ, as discussed in Section 1.4.2. The only difference from standard Bayesian network parameter estimation is that parameters for different nodes in the network — those corresponding to the $x.A$ for different objects x from the same class — are forced to be identical. This similarity allows us to use the well-understood theory of learning from Bayesian networks.

Consider the task of performing *maximum likelihood* parameter estimation. Here, our goal is to find the parameter setting $\theta_{\mathcal{S}}$ that maximizes the likelihood $L(\theta_{\mathcal{S}} \mid \mathcal{I}, \sigma, \mathcal{S})$ for a given $\mathcal{I}$, σ and $\mathcal{S}$. Thus, the maximum likelihood model is the model that best predicts the training data. This estimation is simplified by the *decomposition* of log-likelihood function into a summation of terms corresponding to the various attributes of the different classes. Each of the terms in the square brackets in (1.2) can be maximized independently of the rest. Hence, maximum likelihood estimation reduces to independent maximization problems, one for each CPD. In fact, a little further work reduces Eq. (1.2) even further, to a sum of terms, one for each multinomial distribution $\theta_{X.A|\mathbf{u}}$. Furthermore, there is a closed form solution for the parameter estimates. In addition, while we do not describe the details here, we can take a *Bayesian approach* to parameter estimation by incorporating parameter priors. For an appropriate form of the prior and by making standard assumptions, we can also get a closed form solution for the estimates.

1.5.2 Structure Learning

We now move to the more challenging problem of learning a dependency structure automatically, as opposed to having it given by the user. The main problem here is finding a good dependency structure among the potentially infinitely many possible ones. As in most learning algorithms, there are three important issues that need to be addressed in this setting:

– **hypothesis space:** specifies which structures are candidate hypotheses that our learning algorithm can return;
– **scoring function:** evaluates the "goodness" of different candidate hypotheses relative to the data;
– **search algorithm:** a procedure that searches the hypothesis space for a structure with a high score.

We discuss each of these in turn.

Hypothesis Space. Fundamentally, our hypothesis space is determined by our representation language: a hypothesis specifies a set of parents for each attribute $X.A$. Note that this hypothesis space is infinite. Even in a very simple schema, there may be infinitely many possible structures. In our genetics example, a person's genotype can depend on the genotype of his parents, or of his grandparents, or of his great-grandparents, etc. While we could impose a bound on the maximal length of the slot chain in the model, this solution is quite brittle, and one that is very limiting in domains where we do not have much prior knowledge. Rather, we choose to leave open the possibility of arbitrarily long slot chains, leaving the search algorithm to decide how far to follow each one.

We must, however, restrict our hypothesis space to ensure that the structure we are learning is a legal one. Recall that we are learning our model based on one training database, but would like to apply it in other settings, with potentially very different relational structure. We want to ensure that the structure we are learning will generate a consistent probability model for any skeleton we are likely to see. As we discussed in Section 1.4.2, we can test this condition using the class dependency graph for the candidate PRM. It is straightforward to maintain the graph during learning, and consider only models whose dependency structure passes the appropriate test.

Scoring Structures. The second key component is the ability to evaluate different structures in order to pick one that fits the data well. We adapt

Bayesian *model selection* methods to our framework. Bayesian model selection utilizes a probabilistic scoring function. In line with the Bayesian philosophy, it ascribes a prior probability distribution over any aspect of the model about which we are uncertain. In this case, we have a prior $P(\mathcal{S})$ over structures, and a prior $P(\theta_{\mathcal{S}} \mid \mathcal{S})$ over the parameters given each possible structure. The *Bayesian score* of a structure $\mathcal{S}$ is defined as the *posterior* probability of the structure given the data $\mathcal{I}$. Formally, using Bayes rule, we have that:

$$P(\mathcal{S} \mid \mathcal{I}, \sigma) \propto P(\mathcal{I} \mid \mathcal{S}, \sigma) P(\mathcal{S} \mid \sigma)$$

where the denominator, which is the marginal probability $P(\mathcal{I} \mid \sigma)$ is a normalizing constant that does not change the relative rankings of different structures.

This score is composed of two main parts: the prior probability of the structure, and the probability of the data given that structure. It turns out that the marginal likelihood is a crucial component, which has the effect of penalizing models with a large number of parameters. Thus, this score automatically balances the complexity of the structure with its fit to the data. In the case where $\mathcal{I}$ is a complete assignment, and we make certain reasonable assumptions about the structure prior, there is a closed form solution for the score.

Structure Search. Now that we have a hypothesis space and a scoring function that allows us to evaluate different hypotheses, we need only provide a procedure for finding a high-scoring hypothesis in our space. For Bayesian networks, we know that the task of finding the highest scoring network is NP-hard [1.3]. As PRM learning is at least as hard as Bayesian network learning (a Bayesian network is simply a PRM with one class and no relations), we cannot hope to find an efficient procedure that always finds the highest scoring structure. Thus, we must resort to heuristic search.

The simplest heuristic search algorithm is greedy hill-climbing search, using our score as a metric. We maintain our current candidate structure and iteratively improve it. At each iteration, we consider a set of simple local transformations to that structure, score all of them, and pick the one with highest score. As in the case of Bayesian networks, we restrict attention to simple transformations such as adding or deleting an edge. We can show that, as in Bayesian network learning, each of these local changes requires that we recompute only the contribution to the score for the portion of the structure that has changed in this step; this has a significant impact on the computational efficiency of the search algorithm. We deal with local maxima using random restarts, i.e., when a local maximum is reached in the search, we take a number of random steps, and the,n continue the greedy hill-climbing process.

There are two problems with this simple approach. First, as discussed in the previous section, we have infinitely many possible structures. Second, even the atomic steps of the search are expensive; the process of computing the statistics necessary for parameter estimation requires expensive database operations. Even if we restrict the set of candidate structures at each step of the search, we cannot afford to do all the database operations necessary to evaluate all of them.

We propose a heuristic search algorithm that addresses both these issues. At a high level, the algorithm proceeds in phases. At each phase k, we have a set of potential parents $Pot_k(X.A)$ for each attribute $X.A$. We then do a standard structure search restricted to the space of structures in which the parents of each $X.A$ are in $Pot_k(X.A)$. We structure the phased search so that it first explores dependencies within objects, then between objects that are directly related, then between objects that are two links apart, etc. This approach allows us to gradually explore larger and larger fragments of the infinitely large space, giving priority to dependencies between objects that are more closely related. The second advantage of this approach is that we can precompute the database view corresponding to $X.A, Pot_k(X.A)$; most of the expensive computations — the joins and the aggregation required in the definition of the parents — are precomputed in these views. The sufficient statistics for any subset of potential parents can easily be derived from this view. The above construction, together with the decomposability of the score, allows the steps of the search (say, greedy hill-climbing) to be done very efficiently.

1.6 Experimental results

We have tested our learning algorithm in several domains, both real and synthetic. We now describe experimental results on one synthetic dataset and two real ones.

1.6.1 Genetics Domain

We begin by presenting our results on a synthetic dataset generated by the genetics example used in this chapter. The goal of these experiments is to test the learning algorithm, showing that it can reconstruct the dependency structure if it is clearly present in the distribution.

The datasets here were generated by a PRM that has the structure shown in Figure 1.5(a). We generated various training sets, of size from 200 to 800, with 10 training sets of each size. We also generated an independent test database of size 10,000. A data set of size n consists of a family tree containing n people, with an average of 0.6 blood tests per person. For each training set, we learned a PRM using the algorithm described in the previous section, and then tested how well the learned PRM predicts the test data. For measuring the predictive ability, we used the log-likelihood of the test data, the standard

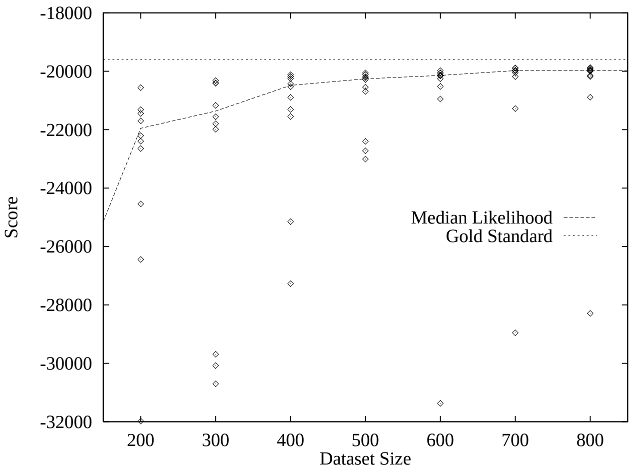


Fig. 1.6. Learning curve showing the generalization performance of PRMs learned in the genetic domain. The x-axis shows the training set size; the y-axis shows log-likelihood of a test set of size 10,000. For each sample size, we show learning experiments on ten different independent training sets of that size. The curve shows median log-likelihood of the models as a function of the sample size.

measure for evaluating density estimation procedures. Figure 1.6 shows the results for the different training sets. The straight line at the top is the log-likelihood of the test data given the "true" model used to generate the data. The data is presented is in the form of a scatter plot, showing the accuracy for each of the training sets, as well as the median log-likelihood of the learned models for each size. We can see that the median is quite reasonable, but there are a few outliers. In most cases, our algorithm learned a model with the correct structure, and scored well; the difference in score is due to the parameter estimation, which is inherently noisy given limited data. However, in a small minority of cases, the algorithm got stuck in local maxima, learning a model with incorrect structure that scored quite poorly.

1.6.2 Tuberculosis Patient Domain

We also applied the algorithm to various real-world domains. The first of these is drawn from a database of epidemiological data for 1300 patients from the San Francisco tuberculosis (TB) clinic, and their 2300 contacts [1.1, 1.26]. For the Patient class, the schema contains demographic attributes such as age, gender, ethnicity, and place of birth, as well as medical attributes such as HIV status, disease site (for TB), X-ray result, etc. In addition, a sputum sample is taken from each patient, and subsequently undergoes genetic marker analysis. This allows us to determine which strain of TB a patient has, and thereby create a Strain class, with a relation between patients and strains. Each patient is also asked for a list of people with

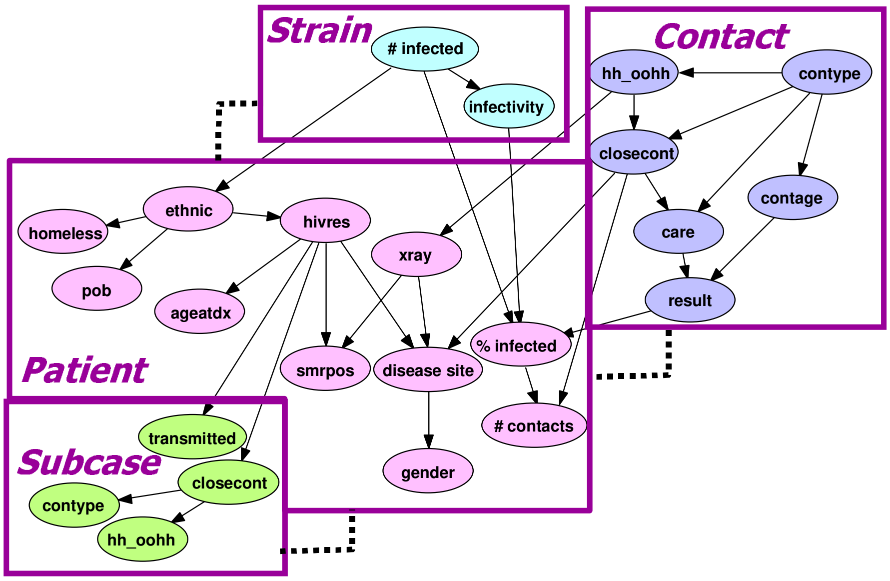


Fig. 1.7. The PRM structure for the TB domain.

whom he has been in contact; the Contact class has attributes that specify the type of contact (sibling, coworker, etc.) contact age, whether the contact is a household member, etc.; in addition, the type of diagnostic procedure that the contact undergoes (*Care*) and the result of the diagnosis (*Result*) are also reported. In cases where the contact later becomes a patient in the clinic, we have additional information. We introduce a new class Subcase to represent contacts that subsequently became patients; in this case, we also have an attribute *Transmitted* which indicates whether the disease was transmitted from one patient to the other, i.e., whether the patient and subcase have the same TB strain.

The structure of the learned PRM is shown in Figure 1.7. We see that we learn a rich dependency structure both within classes and between attributes in different classes. We showed this model to our domain experts who developed the database, and they found the model quite interesting. They found many of the dependencies to be quite reasonable, for example: the dependence of age at diagnosis (*ageatdx*) on HIV status (*hivres*) — typically, HIV-positive patients are younger, and are infected with TB as a result of AIDS; the dependence of the contact's age on the type of contact — contacts who are coworkers are likely to be younger than contacts who are parents and older than those who are school friends; or the dependence of HIV status on ethnicity — Asian patients are rarely HIV positive whereas white patients are much more likely to be HIV positive, as they often get TB as a result of having AIDS. In addition, there were a number of dependencies that they found interesting, and worthy of further investigation. For example, the dependence between close contact (*closecont*) and *disease site* was novel and potentially interesting. There are also dependencies that seem to indicate a bias in the contact investigation procedure or in the treatment of TB; for example, contacts who were screened at the TB clinic were much more likely to be diagnosed with TB and receive treatment than contacts who were screened by their private medical doctor. Our domain experts were quite interested to identify these and use them as a guide to develop better investigation guidelines.

We also discovered dependencies that are clearly relational, and that would have been difficult to detect using a non-relational learning algorithm. For example, there is a dependence between the patient's HIV result and whether he transmits the disease to a contact: HIV positive patients are much more likely to transmit the disease. There are several possible explanations for this dependency: for example, perhaps HIV-positive patients are more likely to be involved with other HIV-positive patients, who are more likely to be infected; alternatively, it is also possible that the subcase is actually the infector, and original HIV-positive patient was infected by the subcase and simply manifested the disease earlier because of his immune-suppressed status. Another interesting relational dependency is the correlation between the ethnicity of the patient and the number of patients infected by the strain. Patients who are Asian are more likely to be infected with a strain which is unique in the population, whereas other ethnicities are more likely to have strains that recur in several patients. The reason is that Asian patients are more often immigrants, who immigrate to the U.S. with a new strain of TB, whereas other ethnicities are often infected locally.

1.6.3 Company Domain

The second domain we present is a dataset of company and company officers obtained from Security and Exchange Commission (SEC) data.[2] The data set includes information, gathered over a five year period, about companies (which were restricted to banks in the dataset we used), corporate officers in the companies, and the role that the person plays in the company. For our tests, we had the following classes and table sizes: Company (20,000), Person (40,000), and Role (120,000). Company has yearly statistics, such as the number of employees, the total assets, the change in total assets between years, the return on earnings ratio, and the change in return on assets. Role describes information about a person's role in the company including their salary, their top position (president, CEO, chairman of the board, etc.), the number of roles they play in the company and whether they retired or were

[2] This dataset was developed by Alphatech Corporation based on Primark banking data, under the support of DARPA's Evidence Extraction and Link Discovery (EELD) project.

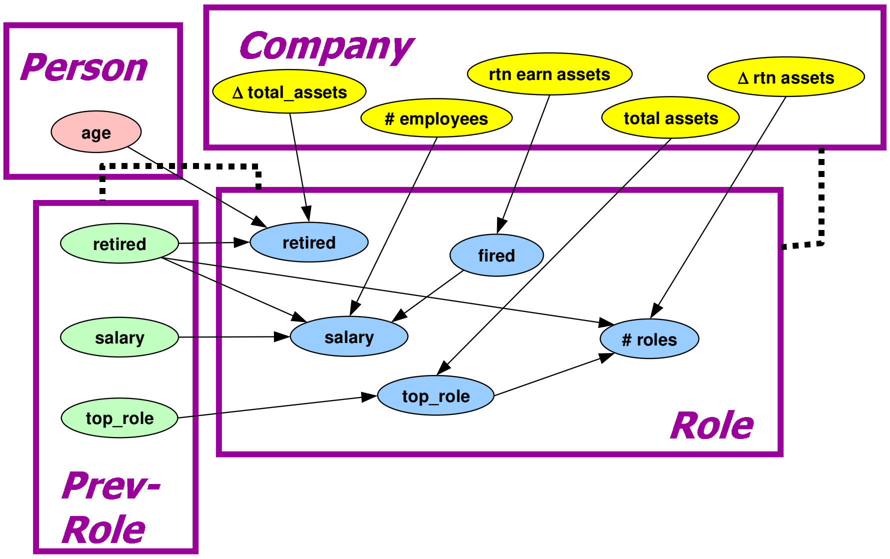


Fig. 1.8. The PRM structure for the Company domain.

fired. Prev-Role indicates a slot whose range type is the same class, relating a person's role in the company in the current year to his role in the company in the previous year.

The structure of the learned PRM is shown in Figure 1.8. We see that we learn some reasonable persistence arcs such as the facts that this year's salary depends on last year's salary and this year's top role depends on last year's top role. There is also the expected dependence between Person.*Age* and Role.*Retired*. A more interesting dependence is between the number of employees in the company, which is a rough measure of company size, and the salary. For example, an employee that receives a salary of \$200K in one year is much more likely to receive a raise to \$300K the following year in a large bank (over 1000 employees) than in a small one. Again, we see interesting correlations between objects in different relations.

1.7 Discussion and related work

There are clearly many other approaches to learning from data. We can categorize them along three main axes: probabilistic versus deterministic; model learning versus classification; and attribute-based versus relational.

Most approaches to machine learning fall into the category of attribute-based classification, with Naive Bayes falling into the probabilistic category, concept learning into the deterministic category, and approaches such as decision trees and neural networks somewhere in between.

Bayesian networks fall into the category of attribute-based model-learning: the result of the learning is a model of the dependencies within the set of attributes, rather than an attempt to predict one particular attribute using the others. However, as we discussed in Section 1.2, Bayesian networks are attribute-based in nature, which substantially limits their ability to represent complex domains involving multiple entities. One might ask, however, whether they can be nevertheless be applied as a discovery tool for relational data. A standard and fairly obvious solution is to flatten the data, creating a single table that contains all the attributes, and then apply Bayesian network learning. While this approach can be useful, it suffers from three major problems.

One key issue relates to the statistical correctness of this approach. Consider our TB dataset, and imagine flattening the data to a table that contains a patient and a contact. In this case, a patient who has two contacts would appear in the table twice (once with each contact), whereas one who has ten contacts would appear in the table ten times. This has the effect of skewing the data substantially, leading to parameter estimates (and correlations) that are highly non-representative of the true distribution. In more technical terms, Bayesian network learning makes the assumption that the data cases are independent (IID), whereas this assumption is clearly violated when our data set is obtained by flattening a relational database.

A second limitation is that we have to determine in advance which are the relevant attributes to put in the table. Consider, for example, the genetics domain, but where we have only phenotype observations (observable features of the person). We might think to join the table to itself, creating a table that contains the person and his parents. However, this would restrict us to discovering correlations that cross generations. For example, male baldness is inherited from a man's maternal grandfather rather than his father; this type of correlation would be lost if we flattened the data in the obvious way.

Finally, even if a Bayesian network model is learned for such a database, it cannot be used to reach conclusions based on relational dependencies. For example, in the TB domain, we might imagine inferring that a patient has a particularly infectious strain by noticing that he transmitted the strain to many of his contacts; we can then infer that a new contact is likely to have the same strain, and therefore that she is also likely to infect many of her contacts. This type of reasoning is only possible using the relational structure that allows us to use information obtained about one patient to reach conclusions about the strain and from that reach conclusions about another patient entirely.

Other than PRMs (and the related approaches of [1.20, 1.23]), the only relational learning approaches are variants of inductive logic programming (ILP). Most ILP approaches are deterministic classification approaches, which do not attempt to model a probability distribution, but rather only to predict (classify) a particular predicate. However, there are two recent developments within the ILP community that are related to PRMs: stochastic logic programs (SLPs)[1.18, 1.6] and Bayesian logic programs (BLPs)[1.13]. The semantics for these two approaches are quite different, with the BLP semantics being the closest to PRMs. An SLP defines a sampling distribution over logic programming proofs; as a consequence, it induces a probability distribution over the possible ground facts for a given predicate. On the other hand, a BLP consists of a set of rules, along with conditional probabilities and a combination rule; following the approach of knowledge-based model construction [1.25], the BLP essentially specifies a propositional Bayesian network. This approach is very similar to the probabilistic logic programs of [1.20, 1.23].

Learning algorithms for these approaches are being developed. Methods for learning SLPs are described in [1.19]. A maximum likelihood approach is taken for parameter estimation for an SLP which is based on maximizing the posterior probability of the program. The task of learning the structure of an SLP is quite different from learning a PRM structure and is based on more traditional ILP approaches. On the other hand, while BLPs are more closely related to PRMs, and methods for learning BLPs have been suggested in [1.13], learning algorithms have not yet been developed. Methods for learning PRMs may be found to be applicable to learning BLPs.

1.8 Extensions

1.8.1 Structural Uncertainty

So far, we have assumed that the skeleton is external to the probabilistic model; in other words, the skeleton is assumed to be part of the input to the PRM. This limitation has two main implications. Most obviously, it implies that the model can only be used in settings where the relational structure is known. Thus, for example, we cannot use it to conclude that a patient is more likely to have a particular strain of TB, based on his demographics and his contacts. A more subtle point is that this restriction can diminish the quality of our model even in cases where the relational structure *is* given, because it ignores interesting correlations between attributes of entities and the relations between them. For example, in a movie domain a "serious" actor is unlikely to appear in many horror movies; hence, we can infer information about an actor's (unknown) attributes based on the Role relation between actors and movies.

The PRM framework can be extended to accommodate uncertainty about the structural relationships between objects as well as about their properties [1.22]. We have extended our learning algorithms to deal with such *structural uncertainty*. We now provide a brief sketch of this extension, which is described in more detail in [1.10].

Suppose we have a simple domain in which we have Movie-Theaters and Movies. Each theater shows some number of movies; this is represented by the class Shows, with reference slots that point to both the theater and the movie. We might know the number of screens that a theater has, but would like to represent a probabilistic model over which movie each theater chooses to show. One way of representing this model is by defining a distribution over the reference slot Shows.Movie: which movie, among the movies currently available, is a theater likely to show. We call this approach to structural uncertainty *Reference Uncertainty.*

Naively, we might think of representing this distribution as a very large multinomial distribution, with a probability for every movie in our domain. This approach is infeasible for two reasons. First, the representation is much too large. Second, our training set will contain one set of movies, but we want to learn a model that can also be applied to other domains, with a different set of movies. In one approach, we can partition the movies into categories, using some set of attributes, either of the movies or of related objects. For example, we might partition on the type of movie (action, horror, comedy), and/or on the origin of the studio which is the source of the movie (Hollywood, independent U.S., or foreign). We then represent the probability that a movie theater shows a movie from a particular category in the partition. As usual, this distribution might depend on some set of parents, e.g., the type of theater (megaplex or art theater). Thus, we might state that an art theater is more likely to play a foreign movie, while a megaplex may be more likely to play a Hollywood movie.

Another approach to structural uncertainty we call *Existence Uncertainty.* Existence uncertainty is a general approach to modeling the probability that a relationship exists between any two entities. To do so, we require that the relationship between the entities is represented by a class; for example, the Registration class in our school domain represents the relationship between a student and a course. We can now represent a probabilistic model that a student will register for a class by introducing a probabilistic model that a given pair (student, class) will appear in the Registration table. More precisely, we specify the probability that such a pair exists in the table given attributes of the student and of the course. For example, it is more likely that a university senior would register for an advanced class, whereas a freshman is more likely to register for an introductory class.

1.8.2 Class Hierarchies

We have also investigated the idea of learning PRMs with class hierarchies. Class hierarchies change the language in two important ways. In the non-hierarchical case, all objects in a class must share the same CPDs. Hierarchical models allow us to specialize the CPDs of attributes in different parts of the hierarchy. In our school example, we might choose to partition the Course class according to their intended year, creating the two subclasses Undergraduate-Course and Graduate-Course. We might then *specialize* the CPD for *Difficulty* in each of the subclasses. More interestingly, a hierarchical model allows us to create dependencies that might seem cyclic in the non-hierarchical case. For example, we can now have a student's grades in a graduate course depend on her grades in the undergraduate courses she has taken. Note that this dependency is apparently cyclic at the level of the Course class, since an attribute of one course depends on the same attribute in another course. However, the division of courses into two subclasses resolves this apparent cycle. A preliminary discussion of these issues can be found in [1.9].

1.9 Conclusions

PRMs provide a new approach to relational data mining that is grounded in a sound statistical framework. Algorithms for learning PRMs build on the recent developments in learning Bayesian networks, and extend learning to this rich class of relational models. Because these models do not focus on a single classification task, they are particularly well suited to exploratory data analysis.

There are several directions for future work. Perhaps the most obvious one is the treatment of missing data and hidden variables. We can extend standard techniques (such as Expectation Maximization for missing data) to this task. (See [1.14] for some preliminary work on related models.) However, the complexity of inference on large databases with many missing values make the cost of a naive application of such algorithms prohibitive. Clearly, this domain calls both for new inference algorithms and for new learning algorithms that avoid repeated calls to inference over these very large problems. Even more interesting is the issue of automated discovery of hidden variables. There are some preliminary answers to this question in the context of Bayesian networks [1.7], in the context of ILP [1.17], and very recently in the context of simple binary relations [1.12]. Combining these ideas and extending them to this more complex framework is a significant and interesting challenge. Ultimately, we would want these techniques to help us automatically discover interesting entities and relationships that hold in the world.

Acknowledgments We thank Benjamin Taskar for his collaboration on several papers that build on this work, and for his help building the software on which our experiments were performed. We also thank Dr. Peter Small and Jeanne Rhee from the Stanford University Medical Center for providing us with the TB data and with their expertise on this topic, and Kendra Moore and Chris White of Alphatech Corporation for providing us with the Company dataset. The work of Lise Getoor, Daphne Koller and Avi Pfeffer was funded by ONR contract N66001-97-C-8554 under DARPA's HPKB program and EELD program, by ONR grant N00014-96-1-0718, and by the generosity of the Sloan Foundation and of the Powell foundation. Nir Friedman was supported through the generosity of the Michael Sacher Trust and the Sherman Senior Lectureship.

References

1.1 M.A. Behr, M.A. Wilson, W.P. Gill, H. Salamon, G.K. Schoolnik, S. Rane, and P.M. Small. Comparative genomics of BCG vaccines by whole genome DNA microarray. *Science*, 284:1520–1523, 1999.

1.2 J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty*, pages 43–52, Madison, WI, 1998. Morgan Kaufman.

1.3 D. M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H.-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pages 121–130. Springer Verlag, 1996.

1.4 G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

1.5 G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

1.6 J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proceedings of the Fifteenth Annual Conference on Uncertainty*, pages 126–133, Stockholm, Sweden, 1999. Morgan Kaufman.

1.7 N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 125–133, Nashville, TN, 1997. Morgan Kaufman.

1.8 N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1300–1307, Stockholm, Sweden, 1999. Morgan Kaufman.

1.9 L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proceedings of the ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries*, pages 25–34, 2000.

1.10 L. Getoor, D. Koller, B. Taskar, and N. Friedman. Learning probabilistic relational models with structural uncertainty. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20. AAAI Press, 2000.

1.11 D. Heckerman. A tutorial on learning with Bayesian networks. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 301 – 354. MIT Press, Cambridge, MA, 1998.

1.12 T. Hofmann, J. Puzicha, and M. Jordan. Learning from dyadic data. In *Advances in Neural Information Processing Systems 11*, pages 466–472, Cambridge, MA, 1998. MIT Press.

1.13 K. Kersting, L. de Raedt, and S. Kramer. Interpreting Bayesian logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 29–35. AAAI Press, 2000.

1.14 D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1316–1321, Nagoya, Japan, 1997. Morgan Kaufman.

1.15 D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 580–587, Madison, WI, 1998. AAAI Press.
1.16 S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis*, 19:191–201, 1995.
1.17 N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
1.18 S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, Amsterdam, 1996.
1.19 S.H. Muggleton. Learning stochastic logic programs. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 36–41. AAAI Press, 2000.
1.20 L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147– 177, 1996.
1.21 J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, 1988.
1.22 A. Pfeffer. *Probabilistic Reasoning for Complex Systems*. PhD thesis, Stanford University, 2000.
1.23 D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64:81–129, 1993.
1.24 P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. Lecture Notes in Statistics. Springer-Verlag, NY, 1993.
1.25 M.P. Wellman, J.S. Breese, and R.P. Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 7(1):35–53, 1992.
1.26 M. Wilson, J.D. DeRisi, H.H. Kristensen, P. Imboden, S. Rane, P.O. Brown, and G.K. Schoolnik. Exploring drug-induced alterations in gene expression in Mycobacterium tuberculosis by microarray hybridization. In *Proceedings of the National Academy of Sciences*, 2000. In press.