From Instances to Classes in Probabilistic Relational Models

Lise Getoor
Computer Science Dept.
Stanford University
Stanford, CA 94305
getoor@cs.stanford.edu

Daphne Koller
Computer Science Dept.
Stanford University
Stanford, CA 94305
koller@cs.stanford.edu

Nir Friedman
School of Computer Sci. & Eng.
Hebrew University
Jerusalem, 91904, Israel
nir@cs.huji.ac.il

Abstract

Probabilistic graphical models, in particular Bayesian networks, are useful models for representing statistical patterns in propositional domains. Recent work develops effective techniques for learning these models directly from data. However these techniques apply only to attribute-value (i.e., flat) representations of the data. *Probabilistic relational models* (PRMs) allow us to represent much richer dependency structures, involving multiple entities and the relations between them; they allow the properties of an entity to depend probabilistically on properties of *related* entities. PRMs represent a generic dependence, which is then instantiated for specific circumstances, i.e., for a particular set of entities and relations between them. Friedman *et al.* showed how to learn PRMs from relational data, and presented techniques for learning both parameters and probabilistic dependency structure for the attributes in a relational model. Here we examine the benefit that class hierarchies can provide PRMs. We show how the introduction of subclasses allows us to use inheritance and specialization to refine our models. We show how to learn PRMs with class hierarchies (PRM-CH) in two settings. In the first, the class hierarchy is provided, as part of the input, in the relational schema for the domain. In the second setting, in addition to learning the PRM, we must learn the class hierarchy. Finally we discuss how PRM-CHs allow us to build models that can represent models for both particular instances in our domain, and classes of objects in our domain, bridging the gap between a class-based model and an attribute-value-based model.

1 Introduction

Probabilistic graphical models, in particular Bayesian networks, are useful models for representing statistical patterns in propositional domains. Recent work (Cooper and Herskovits 1992; Heckerman 1998) develops effective techniques for learning these models directly from data. These techniques are now well-developed, and have been applied to a variety of research and commercial applications. However, they are significantly limited in that they can be applies only to attribute-value, or flat, representations of the data. Any richer relational structure in the domain cannot be modeled.

Consider, the problem of building a model of television viewers and the shows that they enjoy. One approach is to build a Bayesian network that represents the preferences of a single viewer, which has a random variable for each TV program. Each person in the training set has a vector that represents the TV programs that they have watched, and their ratings for that show. From this data set, we can learn a Bayesian network that represents the correlations between their preferences for the different programs. Thus, we could learn that the user's rating of one program, say "Seinfeld", depends on his ratings for the programs that are its parents in the learned network, say "Friends" and "Frasier". Breese *et al.* (1998) compare this approach to other collaborative filtering approaches and show that it is superior in its ability to predict TV-show preferences.

This approach is limited in that it models only the relationships between instances of one class, the TV programs. We cannot model broad dependencies, such as whether a person enjoys sitcom reruns depends on whether they watch prime-time sitcoms. In addition, we cannot model relationships between people. For example, if my roommate watches "LA Law", I am more likely to pull up a chair and watch also.

Probabilistic relational models (PRMs) (Koller and Pfeffer 1998) allow us to represent richer dependency structures, involving multiple entities and the relations between them; they allow the attributes of an entity to depend probabilistically on properties of *related* entities. PRMs model the domain at the *class* level; i.e., all instances in the same class share the same dependency model. This model is then instantiated for particular situations. For example, a person's ratings for a TV program can depend both on the attributes of the person and the attributes of the program. For a given situation, involving some set of people and programs, this dependency model will be used several times. This allows us, for example, to use the properties and ratings of one person to reach conclusions about the properties of a program (e.g., how funny it is), and thereby to reach conclusions about the chances that another viewer would like it.

Friedman *et al.* (1999) showed how to learn PRMs from relational data, and presented techniques for learning both parameters and probabilistic dependency structure for the attributes in a relational model. This learning algorithm exploits the fact that the models are constructed at the class level. Thus, an observation concerning one user and one program is used to refine the class model applied to all users and all programs, hence making much broader use of our data.

However, this class-based approach also has disadvantages: all elements of the same class must use the same model. Thus, for example, we cannot have the rating of a user for documentaries depend on one set of parents, and his ratings for comedies depend on another. In particular, we cannot have the rating for "Seinfeld" depend on the rating for "Friends": The dependendency model for these two ratings must be identical, and we cannot have the rating for "Friends" depending on itself.

In this work, we propose methods for discovering useful refinements of a PRM's dependency model. We begin in Section 3 by defining *Probabilistic Relational Models with Class Hierarchies* (PRMs-CH). PRMs-CH extend PRMs by including class hierarchies over the classes. Subclasses allow us to specialize the probabilistic model for some instances of the class. For example, we might consider subclasses of TV programs, such as documentaries, dramas, newcasts, and sitcoms. The budget of sitcoms (a subclass of TV programs) may depend on their popularity, whereas the budget of newscasts (another subclass of TV programs) may depend on the venue of the associated TV network. Subclassing allows us to model probabilistic dependencies at the appropriate level of detail. For example, we can have the parents of the budget attribute in the sitcom subclass be different than the parents of the same attribute in the documentary subclass. In addition, as we show, subclassing allows additional dependency paths to be represented in the model, that would not be allowed in a PRM that does not support subclasses. For example, whether I watch sitcoms may depend on whether I watch documentaries. PRMs-CH provide a general mechanism that allow us to define a rich set of dependencies. In fact, they provide the basic representational power that will allow us to model dependency models for individuals (as done in Bayesian Networks) and dependency models for categories of individuals (as done in PRMs).

We next turn in Section 4 to some of the practical issues involved in learning PRMs-CH. First, we examine the case where the class hierarchy is given as input, in the relational schema. Our learning task is then simply to choose the appropriate level at which to model the probabilistic dependencies — at the class level, or specialized according to some subclass. We then turn to the case where the class hierarchy is not provided, and in addition to learning the probabilistic model, we must also discover the structure of the class hierarchy.

In Section 5, we present some preliminary experimental results illustrating how we have expanded the space of probabilistic models considered by our learning algorithm, and how this allows us to learn more expressive and more accurate models. We conclude in Section 6 with some discussion and future work.

But, before turning to our new research, which begins in Section 3, the next section reviews some necessary background which includes the definition of a probabilistic relational model.

2 Probabilistic Relational Models

A *probabilistic relational model (PRM)* specifies a template for a probability distribution over a database. The template includes a relational component, that describes the relational schema for our domain, and a probabilistic component, that describes the probabilistic dependencies that hold in our domain. A PRM, together with a particular database of objects and relations, defines a probability distribution over the attributes of the objects and the relations.

2.1 Relational Schema

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* [1].

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 domain of values is denoted $V(X.A)$. We assume here that domains are finite. For example, the Person class might have the descriptive attributes *Sex*, *Age*, *Height*, *Income*, etc. The domain for Person.*Age* might be $\{$child, young-adult, middle-aged, senior$\}$.

The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use 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 of $\mathrm{Dom}[\rho] = X$ and the range type $\mathrm{Range}[\rho] = Y$, where Y is some class in $\mathcal{X}$. A slot ρ denotes a function from $\mathrm{Dom}[\rho] = X$ to $\mathrm{Range}[\rho] = Y$. For example, we might have a class TV-Program with the reference slot *On-Network* whose range is the class Network. For clarity, in some situations it is useful to specify both the name of the reference slot and the class of the object that to which it refers; for example if each network has an owner slot that is a company then we use the notation Network.$\mathit{Owner}_{\mathsf{Company}}$.

It is often useful to distinguish between an *entity* and a *relationship*, as in entity-relationship diagrams. In our language, classes are used to represent both entities and relationships. Thus, entities such as people and TV programs are represented by classes, but a relationship such as Vote, which relates people to TV shows, is also be represented as a class, with reference slots to the class Person and the class TV-Program. This approach, which blurs the distinction between entities and relationships, is common, and allows us to accommodate descriptive attributes that are associated with the relation, such as *Ranking*. We use the generic term *object* to refer both to entities and to relationships.

In addition to the explicit reference slots of a class, we also allow the construction of derived reference slots. One

[1]We note that there is a direct mapping between our notion of class and the tables used in a relational database. 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).

of the most basic derived reference slots is the *inverse*. For each reference slot ρ, we can define an inverse slot ρ^{-1}, which is interpreted as the inverse function of ρ. Note that if ρ is a many-to-one function, then its inverse takes on values that are sets of objects. Another type of derived reference slot is a *slot chain*, which allows us to compose slots, defining functions from objects to other objects to which they are not directly related. More precisely, we define a *slot chain* $\rho_1, \ldots, \rho_k$ be a sequence of slots (inverse or otherwise) such that for all i, $\mathrm{Range}[\rho_i] = \mathrm{Dom}[\rho_{i+1}]$. For example, TV-Program.*Network*.*Owner* can be used to denote the company that owns the network which produces a TV program. And Person.(*Voter*$_{\mathsf{Vote}}{}^{-1}$) can be used to denote the set of votes that a person has made. For notational convenience, we allow the derived reference slots to be have names associated, which can then be used as shorthand. For example, we can replace the unintuitive Person.(*Voter*$_{\mathsf{Vote}}{}^{-1}$) with the shorthand Person.*Votes*.

The semantics of this language is straightforward. In an instantiation $\mathcal{I}$, each X is associated with a set of objects $\mathcal{O}^{\mathcal{I}}(X)$. For each attribute $A \in \mathcal{A}(X)$ and each $x \in \mathcal{O}^{\mathcal{I}}(X)$, $\mathcal{I}$ specifies a value $x.A \in V(X.A)$. For each reference slot $\rho \in \mathcal{R}(X)$, $\mathcal{I}$ specifies a value $x.\rho \in \mathcal{O}^{\mathcal{I}}(\mathrm{Range}[\rho])$. For $y \in \mathcal{O}^{\mathcal{I}}(\mathrm{Range}[\rho])$, we use $y.\rho^{-1}$ to denote the set of entities $\{x \in \mathcal{O}^{\mathcal{I}}(X) \ : \ x.\rho = y\}$. The semantics of a slot chain $\tau = \rho_1.\ldots.\rho_k$ are defined via straightforward composition. For $A \in \mathcal{A}(\mathrm{Range}[\rho_k])$ and $x \in \mathcal{O}^{\mathcal{I}}(X)$, we define $x.\tau.A$ to be the *multiset* of values $y.A$ for y in the set $x.\tau$.

Finally, the *relational skeleton*, σ_r specifies the set of objects in all classes, as well as all the relationships that hold between them. In other words, it specifies $\mathcal{O}^{\sigma}(X)$ for each X, and for each object $x \in \mathcal{O}^{\sigma}(X)$, it specifies the values of all of the reference slots $x.\rho$.

2.2 Probabilistic Model

A probabilistic relational model Π specifies a probability distribution over all instantiations $\mathcal{I}$ of the relational schema. It consists of two components: the qualitative dependency structure, $\mathcal{S}$, and the parameters associated with it, $\theta_{\mathcal{S}}$. The dependency structure is defined by associating with each attribute $X.A$ a set of *parents* Pa$(X.A)$.

A parent of $X.A$ can have the form $X.\tau.B$, for some (possibly empty) slot chain τ. To understand the semantics of this dependence, recall that $x.\tau.A$ is a multiset of values S in $V(X.\tau.A)$. We use the notion of *aggregation* from database theory to define the dependence on a multiset; thus, $x.A$ will depend probabilistically on some aggregate property $\gamma(S)$. There are many natural and useful notions of aggregation, such as *median* or *mode*. We allow $X.A$ to have as a parent $\gamma(X.\tau.B)$; for any $x \in X$, $x.A$ will depend on the value of $\gamma(x.\tau.B)$.

The quantitative part of the PRM specifies the parameterization of the model. Given a set of parents for an attribute, we can define a local probability model by associating with it a *conditional probability distribution (CPD)*. For each attribute we have a CPD that specifies $P(X.A \mid \mathrm{Pa}(X.A))$.

Definition 1: A *probabilistic relational model (PRM)* Π for a relational schema $\mathcal{S}$ 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 $X.\tau.B$, where τ is a slot chain;
- a *conditional probability distribution (CPD)* that represents $P_{\Pi}(X.A \mid \mathrm{Pa}(X.A))$. ▮

Given a relational skeleton σ_r, a PRM Π specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with σ_r:

$$P(\mathcal{I} \mid \sigma_r, \Pi) = \prod_{X \in \mathcal{X}} \prod_{x \in \mathcal{O}^{\sigma_r}(X)} \prod_{A \in \mathcal{A}(X)} P(x.A \mid \mathrm{Pa}(x.A)) \tag{1}$$

For this definition to specify a coherent probability distribution over instantiations, we must ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. To verify acyclicity, we construct an *object dependency graph* G_{σ_r}. Nodes in this graph correspond to descriptive attributes of entities. Let $X.\tau.B$ be a parent of $X.A$ in our probabilistic dependency schema; for each $y \in x.\tau$, we define an edge in G_{σ_r}: $y.B \rightarrow_{\sigma_r} x.A$. We say that a dependency structure $\mathcal{S}$ is *acyclic* relative to a relational skeleton σ_r if the directed graph G_{σ_r} is acyclic. When G_{σ_r} is acyclic, we can use the chain rule to ensure that Eq. (1) defines a legal probability distribution (as done, for example, in Bayesian networks).

The definition of the object dependency graph is specific to the particular skeleton at hand: the existence of an edge from $y.B$ to $x.A$ depends on whether $y \in x.\tau$, which in turn depends on the interpretation of the reference slots. Thus, it allows us to determine the coherence of a PRM only relative to a particular relational skeleton. When we are evaluating different possible PRMs as part of our learning algorithm, we want to ensure that the dependency structure $\mathcal{S}$ we choose results in coherent probability models for *any* skeleton. We provide such a guarantee using a *class dependency graph*, which describes all possible dependencies among attributes. In this graph, we have an (intra-object) edge $X.B \rightarrow X.A$ if $X.B$ is a parent of $X.A$. If $\gamma(X.\tau.B)$ is a parent of $X.A$, and $Y = \mathrm{Range}[\tau]$, we have an (inter-object) edge $Y.B \rightarrow X.A$. A dependency graph is *stratified* if it contains no cycles. If the dependency graph of $\mathcal{S}$ is stratified, then it defines a legal model for any relational skeleton σ_r (Friedman *et al.* 1999).

3 PRMs with Class Hierarchies

In this section, we describe refinements of our probabilistic model using class hierarchies. To motivate our extensions, consider a simple PRM for the TV program domain. Let us restrict attention to the three classes Person, TV-Program, and Vote. We can have the attributes of Vote depending on attributes of the person voting (via the slot Vote.*Voter*) and on attributes of the program (via the slot Vote.*Program*). However, given the attributes of all the people and the programs in the model, the different votes are (conditionally) independent and identically distributed. By contrast, in the BN model for this domain, each program could have a different dependency model; we could even have one depend on the other.

3.1 Class Hierarchies

Our aim is to refine the notion of a class, such as TV-Program, into finer subclasses, such as "sitcoms", "dramas", "documentaries", etc. Moreover, we want to allow recursive refinements of this structure. So that we might refine "dramas" into the subclasses "legal dramas", "medical dramas", and "soap operas".

Formally, we introduce the notion of a probabilistic class hierarchy, similar to that introduced in (Koller and Pfeffer 1997; 1998). We assume that the original set of classes define, at the schema level, the structure of an object (attributes and slots associated with it). Unlike the subclass mechanism in (Koller and Pfeffer 1997; 1998), subclasses do not change this structure.

A hierarchy $H[X]$ for a class X consists of two parts, a finite set of subclasses $\mathcal{C}[X]$ and a partial ordering $\prec$ over $\mathcal{C}[X]$ defining the *class hierarchy*. The set $\mathcal{C}[X]$ specifies the set of subclasses of X. For each value $c \in \mathcal{C}[X]$, we have a subclass X_c. The hierarchy is defined using a partial ordering $\prec$ on $\mathcal{C}[X]$. For $c, d \in \mathcal{C}[X]$, if $c \prec d$, we say that X_c is a *direct subclass* of X_d, and X_d is a *direct superclass* of X_c. We require that $\prec$ define a tree directed to some root $\top$, where $Class_\top$ corresponds to the original class X. We define $\prec^*$ to be the reflexive transitive closure of $\prec$; if $c \prec^* d$, we say that X_c is a subclass of X_d.

For example we may have the class TV-Program and its direct subclasses TV-Program$_{\textit{sitcom}}$, TV-Program$_{\textit{drama}}$, and TV-Program$_{\textit{documentary}}$. The subclass TV-Program$_{\textit{drama}}$ might, in turn, have the direct subclasses TV-Program$_{\textit{legal-drama}}$, TV-Program$_{\textit{medical-drama}}$, and TV-Program$_{\textit{soap-opera}}$. We have that TV-Program$_{\textit{medical-drama}}$ is a direct subclass of TV-Program$_{\textit{drama}}$, and a subclass (but not a direct one) of the root class TV-Program.

We define the leaves of the hierarchy to be the *basic subclasses*, denoted $\textit{basic}(H[X])$. We achieve subclassing for a class X by requiring that there be an additional subclass indicator attribute $X.\textit{Class}$ that determines the basic class to which an object belongs. We note that each object belongs to precisely one basic class. Thus, if c is a subclass, then $\mathcal{O}^{\mathcal{I}}(X_c)$ contains all objects $x \in X$ for which $x.\textit{Class} \prec^* c$, i.e., all objects that are in some basic class which is a subclass of c. In our example, TV-Program has a subclass indicator variable TV-Program.*Class* with the five possible values $\{\textit{sitcom}, \textit{documentary}, \textit{soapopera}, \textit{legal-drama}, \textit{medical-drama}\}$.

Subclasses allow us to make finer distinctions when constructing a probabilistic model. In particular, they allow us to *specialize* CPDs for different subclasses in the hierarchy.

Definition 2: A *probabilistic relational model with subclass hierarchy* is defined as follows. For each class $X \in \mathcal{X}$ we have

- a class hierarchy $H[X] = (\mathcal{C}[X], \prec)$;
- a subclass indicator attribute $X.\textit{Class}$ such that $V(X.\textit{Class}) = \textit{basic}(H[X])$;
- a set of parents and a CPD for $X.\textit{Class}$ (as in Definition 1).
- for each subclass $c \in \mathcal{C}[X]$ and attribute $A \in \mathcal{A}(X)$ we have either
  - a set of parents $\text{Pa}^c(X.A)$ and a CPD that describes $P(X.A \mid \text{Pa}^c(X.A))$; or
  - an *inherited* indicator that specifies that the CPD for $X.A$ in c is inherited from its direct superclass. The root of the hierarchy cannot have the inherited indicator.

We define $P(X.A \mid \text{Pa}^c(X.A))$ to be the CPD associated with A in X_d, where d is the most specialized superclass of c (which may be c itself) such that the CPD of $X.A$ in d is not marked with the inherited indicator. ∎

With the introduction of subclass hierarchies, we can refine our probabilistic dependencies. Before each attribute $X.A$ had an associated CPD. Now, if we like, we can specialize the CPD for an attribute within particular subclass. We can associate a different CPD with the attributes of different subclasses. For example TV-Program$_{\textit{sitcom}}$.*Budget* may have a different conditional distribution from TV-Program$_{\textit{documentary}}$.*Budget*. Further, the distribution for each of the attributes may depend on a completely different set of parents. Continuing our discussion from the introduction, if the budget of sitcoms depends on their popularity, then TV-Program$_{\textit{sitcom}}$.*Budget* would have as parents TV-Program$_{\textit{sitcom}}$.*Popularity*. However, for documentaries, the budget depends on the venue of the broadcasting network — *cable*, *public-broadcast*, or *commercial-broadcast*; then, TV-Program$_{\textit{documentary}}$.*Budget* would have the parent TV-Program$_{\textit{documentary}}$.*On-Network*.*Venue*.

3.2 Refined Slot References

At first glance, the increase in representational power provided by supporting subclasses is deceptively small. It seems that little more than an extra constructed type variable has been added, and that the structure that is exploited by the new sub-classed CPDs could just as easily have been provided using structured CPDs, such as the tree-structured CPDs or decision graphs (Boutilier *et al.* 1996; Chickering *et al.* 1997).

However the representational power has been extended in a very important way. Certain dependency structures that would have been disallowed in the original framework are now allowed. These dependencies appear circular when examined only at the class level; however, when refined and modeled at the subclass level, they are no longer cyclic. One way of understanding this phenomenon is that, once we have refined the class, the subclass information allows us to disentangle and order the dependencies.

Returning to our earlier example, suppose the we have the classes Voter, TV-Program and Vote. Vote has reference slots *Person* and *TV-Program* and an attribute *Ranking* that gives the score that a person has given for a TV program. Suppose we want to model a correlation between a person's votes for documentaries and his votes for soap operas. (This correlation might be a negative one.) In the unrefined model, we do not have a way of referring to a person's votes for some particular subset of programs; we can only consider aggregates over a person's entire set of votes. Furthermore, even if we could introduce such a dependence, the dependency graph would show a dependence of Vote.*Rank* on itself.

The introduction of subclasses of TV programs provides us with a way of isolating a person's votes on some subset of programs. In particular, we can try to introduce a dependence of $\text{Vote}_{\textit{documentary}}.\textit{Rank}$ on $\text{Vote}_{\textit{soap-opera}}.\textit{Rank}$. In order to allow this type dependency, we need a mechanism for constructing slot chains that restrict the types of objects along the path to belong to specific subclasses. Recall that a reference slot ρ is a function from $\mathrm{Dom}[\rho]$ to $\mathrm{Range}[\rho]$, i.e. from X to Y. We can introduce *refinements* of a slot reference by restricting the types of the objects in the domain and range.

Definition 3: Let ρ be a reference slot of X with range Y. Let c and d be particular subclasses of X and Y respectively. A *refined slot reference* $\rho_{\langle c,d\rangle}$ for ρ to c and d is a function from X to Y.

- $x.\rho_{\langle c,d\rangle} = y$ if $x \in X_c$ and $y \in Y_d$ and $x.\rho = y$.
- If $x \notin X_c$, then $x.\rho_{\langle c,d\rangle} = y$ is undefined.
- Likewise, if $y \notin Y_d$, then $y.({\rho_{\langle c,d\rangle}}^{-1})$ is undefined. ∎

Returning to our earlier example suppose that we have subclasses of TV-Program, $\text{TV-Program}_{\textit{documentary}}$ and $\text{TV-Program}_{\textit{soap-opera}}$. In addition, suppose we also have subclasses of Vote $\text{Vote}_{\textit{documentary}}$ and $\text{Vote}_{\textit{soap-opera}}$. To get from a person to their votes, we use the inverse of slot reference Person.*Votes*. Now we can construct refinements of Person.*Votes*, $\textit{Votes}_{\langle \text{Person}, \text{Vote}_{\textit{documentary}}\rangle}$ and $\textit{Votes}_{\langle \text{Person}, \text{Vote}_{\textit{soap-opera}}\rangle}$. Let us name these slots *Documentary-Vote* and *Soap-Votes*. To specify the dependency of votes for documentaries on votes for we can say that $\text{Vote}_{\textit{documentary}}.\textit{Rank}$ has a parent $\gamma(\text{Vote}_{\textit{documentary}}.\textit{Person}.\textit{Soap-Votes}.\textit{Rank})$.

The introduction of subclasses brings the benefit that we can now provide a smooth transition from the PRM, a class-based probabilistic model, to models that are more similar to Bayesian networks. To see this, suppose our subclass hierarchy for TV programs is very "deep" and starts with the general class and ends in the most refined levels with the names of particular TV programs. Thus, at the most refined version of the model we can define the preferences of a person by either class based dependency (the probability of watching sitcoms depends whether the individual watches soap operas) or show based dependency (the probability of watching "Frasier" depends on whether the individual watches "Seinfeld"). The latter model is essentially the same as the Bayesian network models learned by (Breese *et al.* 1998) in the context of collaborative filtering.

In addition, the new flexibility in defining refined slot references allows us to make interesting combinations of these types of dependencies. For example, whether an individual watches a particular show (e.g., "All My Children") can be enough to predict whether she watches a whole other type of shows (e.g., documentaries).

3.3 Probabilistic Dependency Model

At some level, the introduction of a class hierarchy introduces no substantial difficulties — the semantics of the model remain unchanged. Given a relational skeleton σ_r, and subclass information for each object, a PRM-CH Π_{CH} specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with σ_r:

$$P(\mathcal{I} \mid \sigma_r, \Pi) = \prod_X \prod_{x \in \mathcal{O}^{\sigma_r}(X_c)} P(x.\textit{Class} \mid \mathrm{Pa}(x.\textit{Class})) \prod_{A \in \mathcal{A}(X)} P(x.A \mid \mathrm{Pa}^{x.\textit{Class}}(x.A))$$

However, the problem of ensuring that our probabilistic dependency structure is acyclic (and hence defines a coherent probability model) has become a little more complicated. As before, we can build the class dependency graph. Now, however, there is a node in the graph for each attribute of each subclass, $X_c.A$; for each class we have a node $X.\textit{Class}$. As before, we have an edge $X_c.B \to X_c.A$ if $X_c.B$ is a parent of $X_c.A$. In addition, if the CPD for $X.A$ is specialized by c or one of its non-root superclasses, then we have an edge $X.\textit{Class} \to X_c.A$. If $\gamma(X_c.\tau.B)$ is a parent of $X_c.A$, and $Y_d = \mathrm{Range}[\tau]$, we have an edge $Y_d.B \to X_c.A$. In addition, for any refined slot reference $\rho_{\langle e,f`\rangle}$ along the chain τ, we introduce an edge from $Z.\textit{Class}$ to $X_c.A$, where $Z = \mathrm{Dom}[\rho]$.

Once again, we can show that if this dependency graph is *stratified*, it defines a coherent probabilistic model.

Theorem 4: *Let Π_{CH} be a PRM-CH with a stratified dependency graph. Let σ_r be an relational skeleton. Then the PRM and σ_r uniquely define a probability distribution over instantiations $\mathcal{I}$.*

4 Learning PRMs

We start with a brief review of the approach of (Friedman *et al.* 1999) to learning PRMs with attribute uncertainty. We then describe a new algorithm for learning PRMs with class hierarchies. We examine two scenarios: in one case the class hierarchies are given as part of the input and in the other, in addition to learning the PRM, we also must learn the class hierarchy. The learning algorithms use the same criteria for scoring the models, however the search space is significantly different.

4.1 Review

We separate the learning problem into two basic questions: how to evaluate the "goodness" of a candidate structure, and how to search the space of legal candidate structures. We consider each question separately.

For scoring candidate structures, we adapt Bayesian *model selection* (Heckerman 1998). We compute the posterior probability of a structure $\mathcal{S}$ given an instantiation $\mathcal{I}$. Using Bayes rule, we have that $P(\mathcal{S} \mid \mathcal{I}, \sigma) \propto P(\mathcal{I} \mid \mathcal{S}, \sigma) P(\mathcal{S} \mid \sigma)$. This score is composed of two main parts: the prior probability of $\mathcal{S}$, and the probability of the instantiation assuming the structure is $\mathcal{S}$. By making fairly reasonable assumptions about the prior probability of structures and parameters, this second term can be *decomposed* into a product of terms. Each term in the decomposed form measures how well we predict the values of $X.A$ given the values of its parents. Moreover, the term for $P(X.A \mid \mathbf{u})$ depends only on the *sufficient statistics* $C_{X.A}[v, \mathbf{u}]$, that count the number of entities with $x.A = v$ and $\text{Pa}(x.A) = \mathbf{u}$. These sufficient statistics can be computed using standard relational database queries.

To find a high-scoring structure, we use a phased search procedure. At the kth phase of the search, we allow dependency models where parents use slot chains of length at most k. Thus, at phase 0, we allow as parents for $X.A$ only parents within the class X; at phase 1, we allow parents that are of the form $X.\rho.A$; etc. We only expand slot chains in directions that seem to have promising dependencies.

Within each iteration, we use a search procedure that considers operators ω such as adding, deleting, or reversing edges in the dependency model $\mathcal{S}$. The search procedure performs greedy hill-climbing search in this space, using the Bayesian score to evaluate models.

4.2 Class Hierarchies Provided in Schema

We now turn to learning PRMs with class hierarchies. We begin with the simpler scenario, where we assume that the class hierarchy is given as part of input.

As in (Friedman *et al.* 1999), we restrict attention to fully observable data sets. Hence, we assume that, in our training set, the class of each object is given. Without this assumption, the subclass indicator attribute would play the role of a hidden variable, greatly complicating the learning algorithm.

As discussed above, we need a scoring function that allows us to evaluate different candidate structures, and a search procedure that searches over the space of possible structures.

The scoring function remains largely unchanged. For each object x in each class X, we have the basic subclass c to which it belongs. For each attribute A of this object, the probabilistic model then specifies the subclass d of X from which c inherits the CPD of $X.A$. Then $x.A$ contributes only to the sufficient statistics for the CPD of $X_d.A$. With that recomputation of the sufficient statistics, the Bayesian score can now be computed unchanged.

Next we extend our search algorithm to make use of the subclass hierarchy. First, we extend our existing operators to apply to models involving a class hierarchy. Then, we introduce two new sets of operators. The first set allows us to refine and abstract the CPDs of attributes in our model, using our class hierarchy to guide us. The second set, allows us to refine the existing parents of an attribute by refining one of the slots used in the chain.

As in our definition of a PRM-CH, each attribute for each subclass is associated with a CPD. The CPD can either be marked as 'inherited' or 'specialized'. Initially, only the CPD for attributes of $X_\top$ are marked as specialized; all the other CPDs are 'inherited'. Our original search operators — those that add and delete parents — can be applied to attributes at all levels of the class hierarchy. However, we only allow parents to be added and deleted from attributes whose CPDs that have been specialized. Note that any change to the parents of an attribute is propagated to any descendents of the attribute whose CPDs are marked as inherited from this attribute.

Next, we introduce operators $\mathsf{Specialize}$ and $\mathsf{Inherit}$. If $X_c.A$ currently has an inherited CPD, we can apply $\mathsf{Specialize(X_c.A)}$. This has two effects. First, it recomputes the parameters of that CPD to utilize only the sufficient statistics of the subclass c. To understand this point, assume that $X_c.A$ was being inherited from X_d prior to the specialization. The CPD of $X_d.A$ was being computed using all objects in $\mathcal{O}^{\mathcal{I}}(X_d)$. After the change, the CPD will be computed using just the objects in $\mathcal{O}^{\mathcal{I}}(X_c)$. The second effect of the operator is that it makes the CPD modifiable, in that we can now add new parents or delete them. The $\mathsf{Inherit}$ operator has the opposite effect.

The second set of operators that we introduce refine and abstract the parents of an attribute. This relies on the construction of refined slot chains described earlier. Suppose $\tau.B$ is currently a parent of $X.A$. Let ρ be some slot used in τ, and let $\rho_{\langle c,d \rangle}$ be one of its possible refinements. The $\mathsf{Refine\text{-}Parent}$ operator can replace ρ in τ with $\rho_{\langle c,d \rangle}$. We also define the complementary operator $\mathsf{Abstract\text{-}Parent}$, which generalizes a slot in the parent slot chain.

4.3 Learning Subclass Hierarchies

We next examine the case where the subclass hierarchies are not given as part of the input. In this case, we will learn them at the same time we are learning the PRM.

As above, we wish to avoid the problem of learning from partially observable data. Hence, we need to assume that the basic subclasses are observed in the training set. At first glance, this requirement seems incompatible with our task definition: if the class hierarchy is not known, how can we observe subclasses in the training data? We resolve this problem by defining our class hierarchy based on the standard class attributes. For example, TV programs might be associated with an attribute specifying the genre — sitcom, drama, or documentary. If our search algorithm decides that this attribute is a useful basis for forming subclasses, we would define subclasses based in a deterministic way on its values. Another attribute might be the nationality of the network — English, American, or French. The algorithm might choose to refine the class hierarchy by partitioning sitcoms according to the values of this attribute. Note that, in this case, the class hierarchy depends on an attribute of a related class, not the class itself.

We implement this approach by requiring that the subclass indicator attribute be a deterministic function of its parents. These parents are the attributes used to define the subclass hierarchy. In our example, TV-Program.*Class* would have as parents TV-Program.*Genre* and TV-Program.*Network.Nationality*. Note that, as the function defining the subclass indicator variable is required to be deterministic, the subclass is effectively observed in the training data (due to the assumption that all other attributes are observed).

We restrict attention to decision-tree CPDs. The leaves in the decision tree represent the basic subclasses, and the attributes used for splitting the decision tree are the parents of the subclass indicator variable. We can allow binary splits that test whether an attribute has a particular value, or, if we find it necessary, we can allow a split on all possible values of an attribute.

The decision tree gives a simple algorithm for determining the subclass of an object. In order to build the decision tree during our search, we introduce a new operator Split$(X, c, X.\tau.B)$, where c is a leaf in the current decision tree for X.*Class* and $X.\tau.B$ is the attribute on which we will split that subclass.

Note that this step expands the space of models that can be considered, but in isolation does not change the score of the model. Thus, if we continue to use a purely greedy search, we would never take these steps. There are several approaches for addressing this problem. One is to use some lookahead for evaluating the quality of such a step. Another is to use various heuristics for guiding us towards worthwhile splits. For example, if an attribute is the common parent of many other attributes within X_c, it may be a good candidate on which to split.

The other operators, Specialize, Inherit, Refine-Parent and Abstract-Parent remain the same; they simply use the subclasses defined by the decision tree.

5 Preliminary Results

We now present some preliminary results related to our approach. We have not yet implemented the full search algorithm; however we have compared the expressive power of models with subclasses (Π_{CH}) to our standard PRMs that do not support the refinement of class definitions (Π).

We present results for a dataset that we have constructed from information about movies from the Internet Movie Database[2] and information about people's ratings of movies from the Each Movie dataset[3]. We extended the demographic information we had for the people by including census information available for a person's zip-code. The three classes are Movie, Person, and Votes. The training set contained 1467 movies, 5210 people and 243,333 votes.

[2]©1990-2000 Internet Movie Database Limited

[3]http://www.research.digital.com/SRC/EachMovie

We defined a rather simple class hierarchy for Votes, based on the genre of the Movie, Action-Votes, Romance-Votes, Comedy-Votes and Other-Votes. We learned two different models, one that made use of the class hierarchy (Figure 2) and one that did not (Figure 1). We then evaluated the models on five different test sets. Note that, in relational data, different test sets have markedly different structure, so trying the model on different test sets might result in very different answers. Each test set had 1000 votes, and approximately 100 movies and 115 people. The average log-likelihood of the test set for Π was -12079 with a standard deviation of 475.68. The model with class hierarchies, Π_{CH}, performed much better, with average log-likelihood of -10558 and a standard deviation of 433.10. Using a standard t-test, we obtain that Π_{CH} is better than Π with well over 99% confidence interval.

Looking more closely at the qualitative difference in structure between the two models, we see that the PRM-CH is a much richer model. For example the dependency model for Vote.*Rank* cannot be represented without making use of the class hierarchy to both refine the attributes and refine the allowable slot chains. For example, we learn a dependence of $\text{Vote}_{\textit{Romance}}$.*Rank* on $\text{Vote}_{\textit{Romance}}$.*Person*.*Comedy-Votes*.*Rank*, whereas $\text{Vote}_{\textit{Action}}$.*Rank* depends on $\text{Vote}_{\textit{Action}}$.*Person*.*Gender*. Not only are these two dependency models different, but they would be cyclic if interpreted as a standard PRM. Note that in the PRM shown in Figure 1, there is no dependency between Vote.*Rank* and attributes of Person, so the PRM that uses class hierarchies allows us to discover dependencies on properties of Person that we could not uncover before.

6 Conclusions

In the paper, we have proposed a method for making use of class hierarchies while learning PRMs. Class hierarchies give us additional leverage for refining our probabilistic models. They allow us to automatically disentangle our dependency model, allowing us to construct acyclic dependencies between elements within the same class. They also allow us to span the spectrum between class level and instance level dependency models.

However, using class hierarchies significantly expands an already complex search algorithm. The search space for PRMs-CH is much, much larger. In this paper, we describe a general search algorithm. However, a key to the success of the algorithm is the discovery of useful heuristics to guide the search. In future work, we intend to explore the space of possible heuristics, and to test empirically which heuristics work well on real-world problems.

References

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. UAI*, pages 115–123, August 1996.

J. Breese, D. Heckerman, and C. Kadie. Empirical analy-

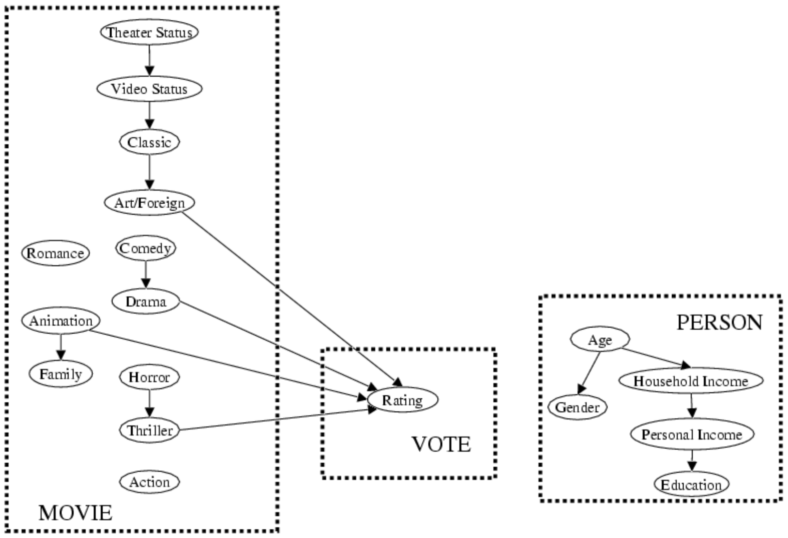

Figure 1: A PRM for the Movie domain.

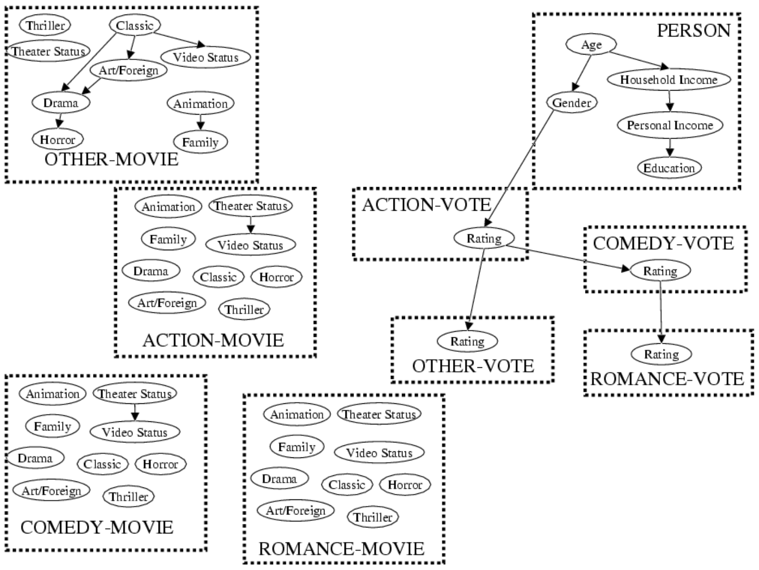

Figure 2: Π_{CH}. The links between vote rankings follows a slot chain, from a person's ranking on one class of movies to the person's ranking on another class of movies.

sis of predictive algorithms for collaborative filtering. In *Proc. UAI*, 1998.

D. M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proc. UAI*, pages 80–89, 1997.

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

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. IJCAI*, 1999.

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

D. Koller and A. Pfeffer. Object-oriented Bayesian networks. In *Proc. UAI*, 1997.

D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. AAAI*, 1998.