University of Alberta

Hierarchical Probabilistic Relational Models for Recommender Systems

by

Jack Newton 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science.**

Department of Computing Science

Edmonton, Alberta
Spring 2005

Only those who risk going too far can possibly find out how far one can go.
T.S. Eliot

To my family.

Abstract

In this thesis we describe an approach to the recommender system problem based on the Probabilistic Relational Model framework.

Traditionally, recommender systems have fallen into two broad categories: *content-based-* and *collaborative-filtering-* based recommender systems, each of which has a distinct set of strengths and weaknesses. We present a sound statistical framework for integrating both of the above approaches, which allows the strengths of one system to help mitigate the weaknesses of the other.

To accomplish this goal we apply the framework of Probabilistic Relational Models (PRMs) to the recommender system problem domain. We begin by applying standard PRMs (sPRMs) to the *EachMovie* recommender system dataset, which uncovers several severe limitations of the sPRM framework. We then apply an extension of PRMs called Hierarchical PRMs (hPRMs) to the recommender problem, which from a theoretical perspective should address several of the limitations of sPRMs. We show through empirical results that hPRMs do, in fact, achieve superior results on the *EachMovie* dataset.

Acknowledgements

First, I would like to thank Russ Greiner for his support as a supervisor, and for allowing me to pursue a thesis topic that was new (and, I think, challenging) to both of us. Russ' steady encouragement and keen intellect helped me through some of the tougher parts of this work.

Rob Holte has also been a big influence; working for Rob as a Research Assistant taught me many things about running and analyzing experiments. Rob also help put me in touch with Lise Getoor, who was an invaluable resource when I was originally learning about PRMs. Without Lise's help, the many complexities and nuances of implementing a PRM system likely would have been overwhelming; her willingness to co-operate and her encouragement to pursue these experiments helped make this thesis topic a reality.

Finally, I would like to thank my wife Tonia and my family for their support throughout the process of writing this thesis.

Contents

Chapter 1

Introduction

Personalized recommender systems, which recommend specific products (e.g., books, movies) to individuals, have become very prevalent — see the success of widely used systems like Amazon.com's book recommender and Yahoo!'s LAUNCHcast music recommender system. The main challenge faced by these system is predicting what each individual will want.

Recommender systems can be broadly broken up into two categories: *content-based-* and *collaborative-filtering-* based recommender systems.

A pure *content-based* recommender will base this on only facts about the products themselves and about the individual (potential) purchaser. This enables us to express each possible purchase as a simple vector of attributes, some about the product and others about the person. If we also know who previously liked what, we can view this as a standard labelled data sample, and use standard machine learning techniques [20] to learn a classifier, which we can later use to determine whether a person will like some (novel) item.

To make this concrete, consider a movie recommendation system that tries to determine whether a specified person will like a specified movie — e.g., will John like Star Wars (`SW`)? A content-based system could use a large `Person×Movie×Vote` database, where each tuple lists facts about a person (e.g., age, gender), then facts about a movie (e.g., genre, box office gross), together with a vote of that person, on a specific movie (e.g., a number between 0 and 5). We could use this dataset to learn a classifier that predicts this vote, based on facts about a person and movie — here about John and about `SW`. There have been a number of such systems based on clustering [2] and Bayesian Models [3], among other technologies. Notice this prediction does not explicitly consider other people (especially people "similar" to John) or other movies (similar to `SW`).

The other main class of recommender system, *collaborative filtering*, addresses this deficiency by using associations: If person P1 appears similar to person P2 (perhaps based on their previous "liked movies"), and P2 liked X, then perhaps P1 will like X as well. A pure collaborative filtering-based system would use only a matrix, whose $\langle i, j \rangle$ element is the vote that person i gives to movie j, which could be unknown. The challenge, then, is using this matrix effectively, to acquire the patterns that will help us predict future votes. While there are a number of other techniques that have proven effective here, such as clustering, PCA, and K-nearest-neighbor [29, 28], notice classical Machine Learning techniques do not work here, as there is no simple way to map this matrix into a simple fixed-size vector of attributes.

Of course, we would like to use *both* content and collaborative information — i.e., include,

as training data, facts about the people, facts about the movies, and a set of ⟨P, M, V⟩ records, which each specifies that person P gave movie M the vote of V. This would not only provide a more complete view of a user's preferences, but would also mitigate the inevitable failure of a purely vote-based system in the case where a user has voted on zero (or very few) items [13]. The challenge is how to use all of this information to predict how John will vote on SW. Here, we want to not only use facts about John and about SW, but also to find and exploit collaborative properties, that deal with people similar to John (in terms of liking similar movies), and movies similar to SW (in terms of being liked by similar people).

Stepping back, the challenge here is learning a distribution over a set of *relations*, describing *sets* of people and *sets* of products, as well as their votes (for those familiar with relational database theory, the concept of *relations* maps nicely to the idea of *tables* in a relational databases, and the tables are linked together in various kinds of relationships). This is quite different from the classical machine learning challenge of learning distributions over tuples (i.e., individual rows of a single relational database), which are iid. That is, while standard techniques seek relationships *within* a row, (e.g., relating a.Vote to a.PersonGender and a.MovieType), our collaborative system needs to reason *across* rows — e.g., to decide that John (described in one row) is sufficiently like George (described in another row) that we use facts about George to make inferences about John. Another natural inter-row application is based on *sets* of rows: e.g., we might use the fact that the *set* of people with some characteristic (e.g., Age=teenage, Gender=male) typically like members of a *set* of movies with some other characteristic (e.g., Genre=action). Furthermore, we want to model relationships between relations; for example, a person's age may influence his/her vote on an action movie.

Probabilistic Relational Models (PRMs) [15] provide a cohesive statistical framework designed to address exactly this type of relational learning and inference problem. This dissertation shows that PRMs can be successfully applied to this learning scenario, in the context of the Recommendation task. We examine the effectiveness of standard PRMs applied to the recommendation task on the EachMovie [8] dataset, then evaluate the effectiveness of an extended version of PRMs called "hierarchical PRMs" (hPRMs) [11]. Our empirical results show that standard PRMs can achieve competitive results on the recommendation task, and then that hPRMs can outperform standard PRMs here.

Section 2 provides an overview of the probabilistic models we use in this thesis. We begin by introducing Bayesian Networks, and describe how we can learn and perform inference on Bayesian Networks. We then proceed to describe standard PRMs, and how learning

and inference apply in this context. Finally, we introduce an extension to standard PRMs, hierarchical PRMs, which can enrich the expressiveness and effectiveness of PRMs through the use of learned class hierarchies.

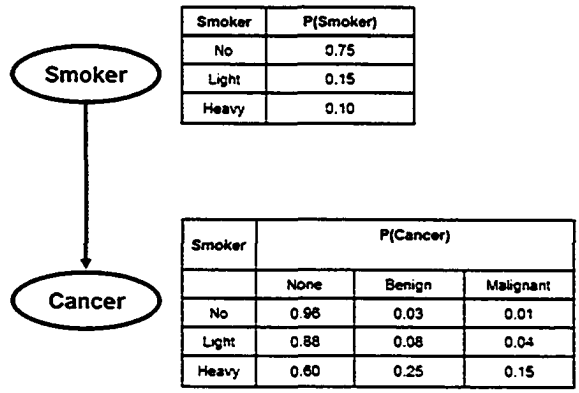
In section 3, we describe our implementation of an (h)-PRM learning and inference system, which we call *tadpole*. This dissertation presents the first implementation and experiments with hierarchical PRMs. Furthermore, in this section we describe how we apply standard and hierarchical PRMs to the recommender system problem, and in particular describe how we map these models to a specific instance of a recommender system dataset, the *EachMovie* dataset.

Section 4 presents the results of applying our framework to the *EachMovie* dataset, and compares standard and hierarchical PRMs with one another, as also presents results comparing these algorithms against other recommender system algorithms.

Finally, in section 5, we present an overview of other approaches that have been taking in the recommender system domain, and compare and contrast these methods with our own.

Chapter 2

Framework



Smoker	P(Smoker)
No	0.75
Light	0.15
Heavy	0.10

Smoker	P(Cancer)		
	None	Benign	Malignant
No	0.96	0.03	0.01
Light	0.88	0.08	0.04
Heavy	0.60	0.25	0.15

Figure 2.1: A simple Bayesian Network

2.1 Overview

In this section we will describe the theoretical framework for our work.

We will begin by describing Bayesian Networks in section 2.2, since PRMs can be viewed as a relational extension of Bayesian Networks. We will also describe the Bayesian Network inference task as well as several well-known inference algorithms, as the algorithms are also used to perform inference on models generated by PRMs.

In section 2.3 we will define standard PRMs (sPRMs), and describe how they extend the Bayesian Network framework.

Finally, in section 2.4 we will describe Hierarchical PRMs, and show how they improve the expressiveness and utility of sPRMs.

2.2 Bayesian Networks

Bayesian Networks (BNs) [23] are a graphical representation of a joint probability distribution. Each BN encodes a compact description of a joint probability distribution by leveraging the conditional independence structure of many problem domains. That is, by taking advantage of the fact that a given variable is generally affected by only a handful of other variables, we are able to encode a vastly reduced representation of the dependencies described by a joint probability distribution.

Figure 2.1 shows an example of a simple BN.

In general, a BN $\mathcal{B} = \{\mathcal{G}, \theta\}$ consists of two components. The first component is a directed acyclic graph (DAG) $\mathcal{G}$, representing the dependency structure of the BN. The nodes in $\mathcal{G}$ correspond to the attributes $A_1, \ldots, A_n$, which can each be viewed as a random

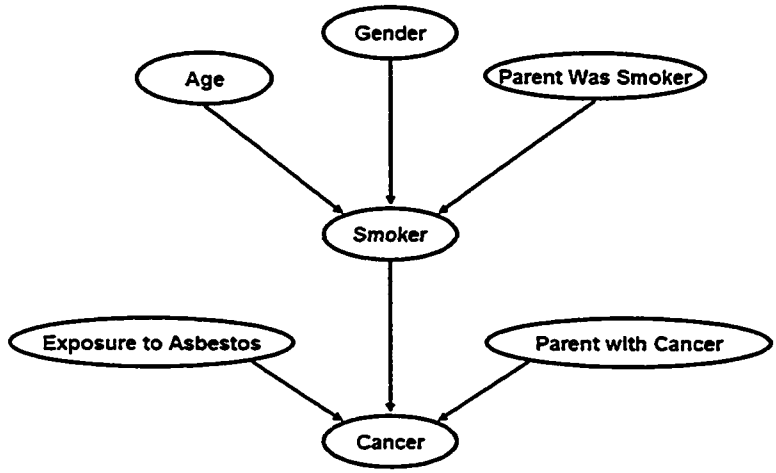


Figure 2.2: A More Complex Bayesian Network

variable.

In our simple example, we have two attributes: *Smoker*, and *Cancer*. Additionally, each attribute has a fixed space of possible values, which we denote $\mathcal{V}(A_i)$. In our example, $\mathcal{V}(Smoker) = \{No, Light, Heavy\}$, and $\mathcal{V}(Cancer) = \{None, Benign, Malignant\}$

The edges in $\mathcal{G}$ represent a direct dependence of A_i on its parents $Pa(A_i)$. In Figure 2.2 the *Cancer* attribute depends directly on three variables: *Exposure to Asbestos*, *Smoker*, and *Parent with Cancer*.

The second component of a BN is the set of parameters θ that associates a *Conditional Probability Distribution (CPD)* with each node in $\mathcal{G}$. The CPD for a node A_i specifies a probability distribution over the values of A_i for every possible assignment to its parents $Pa(A_i)$. Using standard probability notation, we can denote the CPD for a node A_i as $P(A_i|Pa(A_i))$.

A CPD can be represented in a variety of ways. A relatively simple encoding of a CPD is a *conditional probability table (CPT)*, as shown in Figure 2.1. In the case of *Smoker*, which has no parent node, the CPT simply encodes the prior probability of the various values. For the *Cancer* node, the probability of every possible outcome enumerated for every possible combination of the parent node's values. For example, in Figure 2.1 $P(Cancer = Benign|Smoker = Heavy) = 0.25$. Alternative encodings for CPDs are possible; for example, *Conditional Probability Trees* encode the probability distribution as a tree, which in some cases can reduce storage requirements for the CPD [1].

Typically, Bayesian Networks are significantly more complex than the network depicted in Figure 2.1; a BN that extends this model is shown in Figure 2.2 (we have excluded the

CPDs from this example for simplicity).

The main characteristic of a Bayesian Network that allows it to efficiently encode a join probability distribution is the conditional independence assumptions that the graphical structure $\mathcal{G}$ encodes: every node A_i is conditionally independent of its non-descendants given its parents. For example, in Figure 2.2, the probability of *Cancer* is conditionally independent of *Age*, *Gender*, and *Education* given the value of *Smoker*.

Together with the parameters $\mathcal{S}$, the conditional independence assumptions encoded by $\mathcal{G}$ defines a complete joint distribution by using the chain rule:

$$P_{\mathcal{B}}(A_1, \ldots, A_n) = \prod_{i=1}^{n} P_{\mathcal{B}}(A_i | Pa(A_i)) \tag{2.1}$$

2.2.1 Bayesian Network Learning

Learning Bayesian Networks can be looked at as two distinct tasks: learning "optimal" parameters (i.e. CPDs) for a fixed network structure, and learning an "optimal" structure from data.

Learning Parameters

In a generative framework, learning Bayesian Network parameters, namely the conditional probability distributions, for a fixed structure, is a matter of maximizing the likelihood of the data (which contains M cases) for each of the n nodes:

$$L = \frac{1}{M} \sum_{i=1}^{n} \sum_{m=1}^{M} log(P(X_i | Pa(Xi), D_m)) \tag{2.2}$$

Given complete data (i.e., no data is missing from the training set of data), parameter estimation is a fairly straightforward task (in this thesis, we only consider situations where we have complete data; incomplete data could be handled using techniques such as Expectation Maximization) [7].

A nice property of the log-likelihood function is that it decomposes in such a way that we are able to maximize the log-likelihood of each node independently. The maximum likelihood estimate for a given CPT also has the nice property that the maximum likelihood parameters are simply the frequency counts for a given value and its parents' values in the data. These frequency counts are also called *sufficient statistics*.

In some cases the sufficient statistics alone are not informative; for example, consider a dataset where a potential combination of a child/parent values does not exist - this would have a count of 0, and thus create a probability entry of "0.0" in the CPT for that variable.

However, we need to consider the implication of this: we are saying the given variable combination will *never* occur. Given a limited amount of training data, it could be quite likely that a *possible* event has simply never been observed, and should thus have a small (non-zero) probability that becomes vanishingly small as we see more observations that prove the event is unlikely.

Dirichlet priors give us a mechanism to handle this uncertainty by defining a prior for each variable of interest. We can set up "imaginary" observations for an event, which is our prior prediction of the likelihood of an event; as the number of real observations increase, the effect of this prior gets "washed out". For a more detailed explanation of Dirichlet distributions, see [10].

Learning Structure

Learning a Bayesian Network structure is a more challenging proposition than simply finding the maximum likelihood parameters for a fixed network structure.

There is no way of efficiently finding the best model for a given dataset; in general, we must resort to searching for a model using a scoring function that informs us how good this model is. A common search heuristic is to start with a model with no links between nodes, and to begin adding parents with fitted parameters to nodes, and scoring the resulting network [26]. Other than just adding arcs to the graph, we could also consider reversing or deleting existing arcs. Certain constraints must be placed on this search process, as we must avoid introducing cycles into the graph (otherwise the Bayesian Network will not represent a well-formed probability distribution).

While it may seem we want to generate a network with maximum likelihood, optimizing against this sole metric will result in a fully connected network [26]; instead, we must somehow balance the likelihood of the model against the model's complexity. A wide variety of metrics that attempt to balance a model's likelihood against the model's complexity have been used in practice; for a good overview of these methods see [12] or [30].

2.2.2 Bayesian Network Inference

Given that a Bayesian Network models a joint probability distribution, one of the frequent uses for a Bayesian Network is to determine probabilities for a given variable (or variables) given the value(s) of other variables. Consider our running *Smoker* example; a typical question we may want to ask of this network is "given that a male individual is a heavy smoker, has been exposed to asbestos, and has no parents that smoke, what is the probability that he will develop cancer?"

The processes of arriving at an answer to this kind of question using a Bayesian Network is called *Bayesian Network Inference.* More precisely, Bayesian Network inference calculates the posterior probability distribution for a set of *query variables* (*Cancer* in this case) given assignments to a set of *evidence variables.*

For small networks, a wide variety of exact inference algorithms exist [12]. The most elementary exact inference algorithm would compute sums of products of conditional probabilities from the network [26]. This basic approach is amenable to many optimizations, such as the *bucket elimination* algorithm, which eliminates variables that are irrelevant to the query in order to reduce the overall number of calculations required to perform inference [6].

Unfortunately, even with the optimizations that result from leveraging the conditional independence encoded by a Bayesian Network, exact inference in an arbitrary Bayesian Network for discrete variables is NP-hard [4]. Although certain restricted forms of Bayesian Network structures, such as polytrees, are amenable to efficient inference, exact inference in large, multiply-connected networks is intractable.

In practice, inference on large, multiply-connected networks is performed using *approximate inference* methods. These methods are not guaranteed to give the correct answer for a given query, but often return values that are close to the true values [26]. One of the most important approximation algorithms is the *loopy belief propagation* algorithm originally introduced by Pearl [24]. This message-passing algorithm has recently been shown to be both fast and accurate on very large and highly connected networks [22].

2.3 Probabilistic Relational Models

A PRM can encode *class-level* dependencies that can subsequently be used to make inferences about a particular instance of a class. For example, we might connect (the class of) teenage boys to (the class of) action movies, then use that to infer that the teenage boy John will like the action movie SW. Of course, we could do something like that in a standard Belief Network, by first transforming this relational information into a non-structured, propositionalized form.

By performing this propositionalization we lose the rich relational structure and introduce statistical skews [11].

To see how this skew is introduced, consider the diagram in Figure 2.3. Although *Jim* only appears once as an object in the relational representation, taking the cross-product over all the tables in the relational representation to produce the propositionalized (flat-file)

Relational Representation

Person

Name	Age
Jim	20
Sara	29
Sandy	58
Dave	20

Purchase

Name	ProdID
Jim	1
Jim	1
Jim	1
Sandy	2
Dave	3
Dave	4

Product

ID	Description
1	Gloves
2	Jacket
3	Socks
4	Jeans

Flat-File Representation

Name	Age	Product
Jim	20	Gloves
Jim	20	Gloves
Jim	20	Gloves
Sandy	29	Jacket
Dave	20	Socks
Dave	20	Jeans

Figure 2.3: Propositionalizing relational data

representation results in three occurrences of *Jim* in the propositionalized representation. If a simple Bayesian Network were to be learned on this propositionalized dataset, the concept of *Jim* as a single object would be lost; instead, the learning algorithm would likely learn a model that indicates 20 year-olds like gloves; however, this isn't really the case - *Jim* is the only 20 year-old that has bought gloves in our database! However, by propositionalizing the data we have skewed the data in favour of individuals that have made multiple purchases. With a relational model, we are able to account for the fact that *Jim* is a single *object* in our database; since we never need to produce a propositionalized view of our data when learning a PRM, we avoid introducing these skews.

Since PRM can be learned directly on a relational database, we are able to retain and leverage the rich structure contained therein.

Following [11], we view PRM as a pair $\Pi = \langle \mathcal{S}, \theta_{\mathcal{S}} \rangle$ defined over a Relational Schema $\Omega = \langle \mathcal{X}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the qualitative dependency structure of the PRM and $\theta_{\mathcal{S}}$ is the set of associated parameters. Here, $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of *classes* and, for each X, $\mathcal{R}(X) = \{\rho_i\}$ is a set of *reference slots* that define the relationships between class X and other classes.

Each class $X \in \mathcal{X}$ is composed of a set of *descriptive attributes* $\mathcal{A}(X) \in \mathcal{A}$, which in turn

take on a range of values $V(X.A)$. For example, consider a schema that models movies, people, and the votes that people provide for movies. This schema has three classes: `Vote`, `Person`, and `Movie`. For the `Vote` class, the single descriptive attribute is `Score` with values $\{0,\ldots,5\}$; i.e., $\mathcal{A}(Vote) = \texttt{Score}$, and $\mathcal{V}(Score) = \{0,\ldots,5\}$; for `Person` two sample descriptive attributes are `Age` and `Gender`, which take on values {`young`, `middle-aged`, `old`} and {`Male`, `Female`} respectively; and for `Movie`, one descriptive attribute is `Rating`, which takes on values {`G`, `PG`, `M`, `R`}. Furthermore, a class can be associated with a set of *reference slots*, $\mathcal{R}(X) = \{\rho_1, \ldots, \rho_k\}$. The reference slot $X.\rho$ defines how objects of class X are related to objects in other classes in the relational schema (in relational database terminology, a reference slot is analogous to a foreign key). Continuing our example, the `Vote` class would be associated with two reference slots: `Vote.ofPerson`, which defines a link from a `Vote` object to a specific `Person` object; and `Vote.ofMovie`, which defines a link from a `Vote` object to a specific `Movie` object. A sequence of one or more reference slots can composed to form a *reference slot chain*, $\tau = \rho_1 \circ \cdots \circ \rho_\ell$, and attributes of related objects can be denoted by using the shorthand $X.\tau.A$, where A is a descriptive attribute of the related class. For example, `Vote.ofPerson.Gender` refers to the gender attribute of the `Person` associated with a given `Vote` object. To make this concrete, consider a `Vote` object with identifier number 254 that describes *John*'s vote on the movie *StarWars*. In this case, $\texttt{Vote}_{254}.ofPerson = John$, and $\texttt{Vote}_{254}.ofMovie = StarWars$

For each instance x (of type X), and each $A \in \mathcal{A}(X)$, we view $x.A$ as a random variable, whose distribution is determined by the PRM's dependency structure $\mathcal{S}$ and parameters $\theta_{\mathcal{S}}$. This dependency structure $\mathcal{S}$, specifies the parents $Pa(X.A)$ for each attribute $X.A$, where each parent is a descriptive attribute, which can be within the class `X`, or within another class `Y` that is reachable through a reference slot chain. For example, Figure 2.4(a) shows `Person.Education` has the single parent `Person.Age`, and `Vote.Score` has many parents, including `Vote.ofPerson.Gender` (recall `Vote.ofPerson` refers to the person associated with a particular vote object).

The PRM $\Pi = \langle \mathcal{S}, \theta_{\mathcal{S}} \rangle$ also associates a *conditional probability distribution (CPD)* with each attribute $X.A$, which specifies the conditional probability of this variable given each assignment to its parents $Pa(X.A)$ — i.e., $P_\Pi(X.A|Pa(X.A))$. Returning to Figure 2.4(a), this might state that $P_\Pi(\texttt{Person.Age} = \texttt{child} \,|\, \texttt{Person.Education} = \texttt{PhD}) = 0.0$, and $P_\Pi(\texttt{Person.Age} = \texttt{teenager} \,|\, \texttt{Person.Education} = \texttt{highSchool}) = 0.3$, etc. (To understand this, suppose all we know about (the `Person`) Fred is that he has `Fred.Education=PhD`. We can then infer $P_\Pi(\texttt{Fred.Age} = \texttt{child} \,|\, \texttt{Fred.Education} = \texttt{PhD}) = 0.0$.) In general, we can

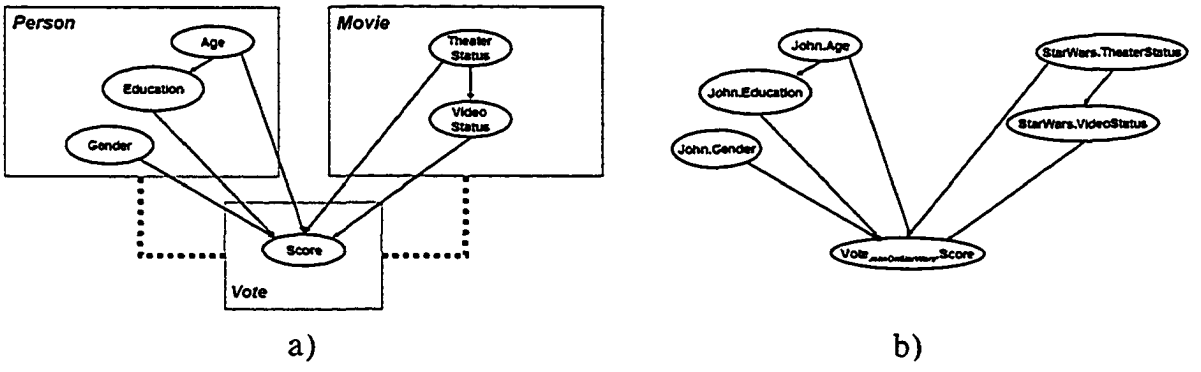


Figure 2.4: (*a*) Standard PRM learned on EachMovie dataset (*b*) Ground Bayesian Network for one `Vote` object

view the structure as a Bayesian Net; we will see that the same basic inference process applies (see [23]).

One final comment: In many cases, the parent of a given attribute will take on a *multiset* of values S in $V(X.\tau.A)$. For example, there could be a dependency between a `Person`'s age on his/her rating of movies in the `Children` genre. However, we cannot directly model this dependency since the user's ratings on `Children`'s movies is a *multiset* of values: e.g., perhaps the user has seen 5 children movies, and given them the (respective) scores of {4, 5, 3, 5, 4}. To address this issue, we borrow the notion of a *aggregate* operator from relational database theory [17]. Examples of such operators are `Median` and `Average`, which would reduce the vector of values above to 4 and 4.2, respectively. In this dissertation, we will always reduce S to a single value using `Median`. (So we would use 4 in this case.)

2.3.1 Learning PRMs

The algorithms for learning PRMs closely resemble the methods used for learning Bayesian Networks, as laid out in 2.2.1. As with Bayesian Networks, there are two fundamental elements involved in learning PRMs: learning parameters, and learning structure.

Parameter Estimation

Like Bayesian Networks, we consider learning PRM parameters where the dependency structure S is known. The input to the parameters estimation algorithm is a dependency structure, S, along with a training dataset, $\mathcal{I}$; the output is the parameter set θ_S.

Just as was the case with Bayesian Networks, we attempt to maximize the likelihood of the data given the parameters. For a PRM the likelihood of the data is:

$$L = \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} log(P(\mathcal{I}_{x.A} | \mathcal{I}_{Pa(x.A)})) \right] \tag{2.3}$$

Where $\sigma(X_i)$ are the objects that are instantiations of X_i in the database.

This equation, like its Bayesian Network analogue, decomposes into a summation of terms:

$$L = \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \left[\sum_{x \in \sigma(X_i)} log(P(\mathcal{I}_{x.A} | \mathcal{I}_{Pa(x.A)})) \right] \tag{2.4}$$

$$= \sum_{X_i} \sum_{A \in \mathcal{A}(X_i)} \sum_{v \in \mathcal{V}(X.A)} \sum_{\mathbf{u} \in \mathcal{V}(Pa(X.A))} C_{X.A}[v, \mathbf{u}] \cdot log(\theta_{v|\mathbf{u}}) \tag{2.5}$$

where $C_{X.A}[v, \mathbf{u}]$ is the count of the number of times we observe $\mathcal{I}_{x.A} = v$ and $\mathcal{I}_{Pa(x.A)} = \mathbf{u}$.

The set of these counts for the entire data set make up the *sufficient statistics* required to perform maximum likelihood estimation. Again, just as with Bayesian Networks, the values of the parameters that maximize the likelihood function are their frequencies in the data. (In section 3 we describe how we are able to use relational database queries to efficiently generate counts required for this calculation.)

Just as with Bayesian Network, we have the problem where $C_{X.A}[v, \mathbf{u}]$ will be zero for any set of values that do not appear in the training set. Again, we can use a Dirichlet prior to help mitigate this effect; we denote the hyperparameters associated with the Dirichlet as $\alpha_{X.A}[v, \mathbf{u}]$.

Structure Learning

As with Bayesian Networks, learning the structure for a PRM is much more challenging than learning parameters.

Recall from section 2.2.1 that any Bayesian Network we learn must be a directed acyclic graph (DAG) in order for the BN to represent a well- formed probability distribution. In section 2.3.2 we will describe how PRM inference is performed by generating a *ground Bayesian Network* from the PRM. Since the instance-level ground Bayesian Network inherits the same structure as its parent, class-level PRM, it is necessary that the PRM dependency structure is a DAG in order to guarantee that the ground Bayesian Network will also be a DAG.

For this thesis we assume there are no *guaranteed acyclic reference slots* [11], which are a special type of reference slot that only appear in specialized domains (for example, a

grandfather relation linking one *Person* class to another would be an example of a guaranteed acyclic reference slot). Allowing guaranteed acyclic reference slot complicates the structure learning procedure, so for brevity we will only consider the (typical) case where all reference slots can potentially introduce cycles. For a complete discussion of guaranteed acyclic reference slots see [11].

We designed a PRM structure learning algorithm that closely resembles the Bayesian Network learning algorithms described in 2.2.1. That is, we take an initially empty PRM structure, and progressively add links between nodes that maximize a regularized likelihood function. However, unlike a Bayesian Network, links cannot exist between two arbitrary nodes (i.e. attributes) in a PRM; the two nodes must either be a) in the same class, or b) reachable through a reference slot chain. Finally, just as with Bayesian Networks, we need a scoring metric to measure the "goodness" of a given model; in this thesis we will use a simple extension of the Bayesian score for Bayesian Networks [12]:

$$P(\mathcal{I}|S,\sigma) = \prod_i \prod_{A \in \mathcal{A}(X_i)} \left[\prod_{\mathbf{u} \in \mathcal{V}(Pa(X_i.A))} DM(\{C_{X_i.A}[v,\mathbf{u}]\}, \{\alpha_{X_i.A}[v,\mathbf{u}]\}) \right] \tag{2.6}$$

where the Dirichlet marginal

$$DM(\{C[v]\}, \{\alpha[v]\}) = \frac{\Gamma(\sum_v \alpha[v])}{\Gamma(\sum_v (\alpha[v] + C[v]))} \prod_v \frac{\Gamma(\alpha[v] + C[v])}{\Gamma(\alpha[v])} \tag{2.7}$$

and $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ is the *Gamma* function.

We adapt the phased learning algorithm presented in [11] for our learning algorithm. We use a straightforward greedy hill-climbing method where we consider all valid operations for a given structure (i.e., add an edge, remove an edge, or delete an edge, subject to DAG constraints), score all those operations, and select the highest-scoring operation. We refer to this algorithm as a *phased* algorithm because we first evaluate all candidate operations with a reference slot chain length of 0 (i.e., all links are intra-class links), then all candidate operations with a reference slot chain length of 1, *etc*, until we reach a predefined reference slot chain length limit (in this thesis we have used a maximum reference slot chain length of 3). Since long reference slot chain length are essentially a form of graph complexity, this phased approach has the desirable side-effect of preferring short reference slot chains over long ones.

2.3.2 PRM Inference

For a given relational schema Ω, an instance $\mathcal{I}$ of that schema is a collection of objects $\mathcal{I}(\mathcal{X})$ that are instances of the classes (X) defined in Ω.

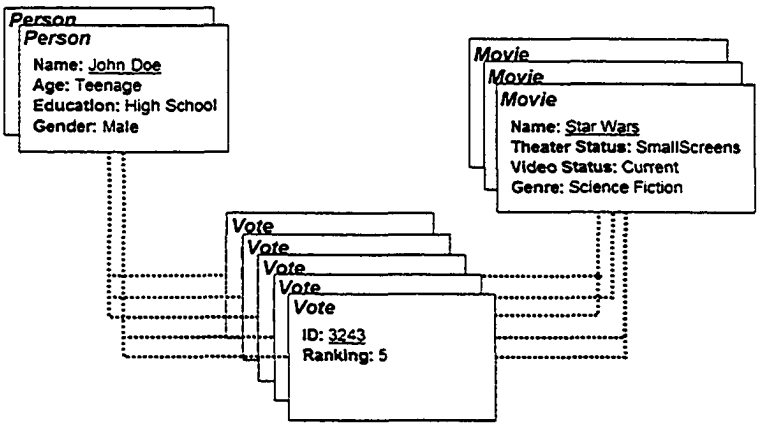


Figure 2.5: A sample PRM instance

We denote an instance of class $\mathcal{X}$ as x; for each attribute in $\mathcal{A}(\mathcal{X})$ we have a corresponding $x.A$ and for every reference slot $X.\rho$ we have a corresponding $x.\rho$.

For each object x in the instance $\mathcal{I}$ we use $\mathcal{I}_x.A$ to denote the value of $x.A$ in $\mathcal{I}$.

For example, consider the example instance as depicted in Figure 2.5. In this example we have two people that are instances of the *Person* class, who have cast five votes (instances of the *Vote* class) on three distinct movies (instances of the *Movie* class). Each of these objects has its own set of attributes values and links to other objects.

To perform PRM inference, we take the PRM structure and parameters, Π, together with the instance for that schema, $\mathcal{I}$, and generate a *ground Bayesian Network*, and perform inference on this ground Bayesian Network.

This process of generating a ground Bayesian Network, sometimes called *unrolling a PRM*, is relatively simple. The class-level parameters and dependencies defined by the PRM are simply copied down to their instance level peers; i.e., an instance x of class $\mathcal{X}$ inherits the parameters defined for $\mathcal{X}$, and has connectivity to other objects in the graph as defined by the PRM.

With this ground Bayesian Network defined, we are able to use standard Bayesian Network inference algorithms to answer any query.

2.4 Hierarchical Probabilistic Relational Models

The collaborative filtering problem reveals two major limitations of PRMs, which in turn motivate hPRMs. First, in the above model, `Vote.Score` can depend on attributes of related objects, such as `Person.Age`, but it is not possible to have `Vote.Score` depend on itself in any

way. For example, we may want to have a user's vote on *Action* movies somehow influence our predicted vote on *Thriller* movies for that user. This is because the class-level PRM's dependency structure must be a directed acyclic graph (DAG) in order to guarantee that the instance-level ground Bayesian Network forms a DAG [9], and thus qualify as a well-formed probability distribution. Without the ability to have `Vote.Score` depend probabilistically on itself, we lose the ability to have a user's rating of an item depend on his rating of other items or on other user's ratings on this movie, which is critical for a collaborative-filtering-based recommender system. For example, we may wish to have the user's ratings of `Comedies` influence his rating of `Action` movies, or his rating of a specific `Comedy` movie influence his ratings of other `Comedy` movies. Second, in the above model, we are restricted to one dependency graph for `Vote.Score`; however, depending on the type of object the rating is for, we may wish to have a specialized dependency graph to better model the dependencies. For example, the dependency graph for an `Action` movie may have `Vote.Score` depend on `Vote.ofPerson.Gender`, whereas a `Documentary` may depend on `Vote.ofPerson.Age`.

2.4.1 Overview

To address the problems described above, we introduce a class hierarchy that applies to our dataset, and modify the PRM learning procedure to leverage this class hierarchy in making predictions. In general, the class hierarchy can either be provided as input, or can be learned directly from the data. We refer to the class hierarchy for class X as $H[X]$. Figure 2.6 shows a sample class hierarchy for the EachMovie domain. $H[X]$ is a DAG that defines an IS-A hierarchy using the subclass relation $\prec$ over a finite set of subclasses $C[X]$ [11]. For a given $c, d \in C[X]$, $c \prec d$ indicates that X_c is a *direct subclass* of X_d (and X_d is a *direct superclass* of X_c). The leaf nodes of $H[X]$ represent the *basic subclasses* of the hierarchy, denoted $basic(H[X])$.

In this dissertation we assume all objects are members of a basic subclass, although this is not a fundamental restriction of hPRMs. Each object of class X has a subclass indicator $X.Class \in basic(H[X])$, which can either be specified manually or learned automatically by a supplementary algorithm. By defining a hierarchy for a class X in a PRM, we also implicitly specialize the classes that are reachable from X via one or more reference slots. For example, if we specialize the `Movie` class, we implicitly specialize the related `Vote` table into a hierarchy as well. For example, in Figure 2.7, the `Vote` class is refined into four leaf classes, each associated with one of the hierarchy elements in $basic(H[X])$. Note we have achieved our goal of having one pattern of voting influence other votes: in the above example a predicted vote for a *Thriller* movie depends on the user's average vote on an

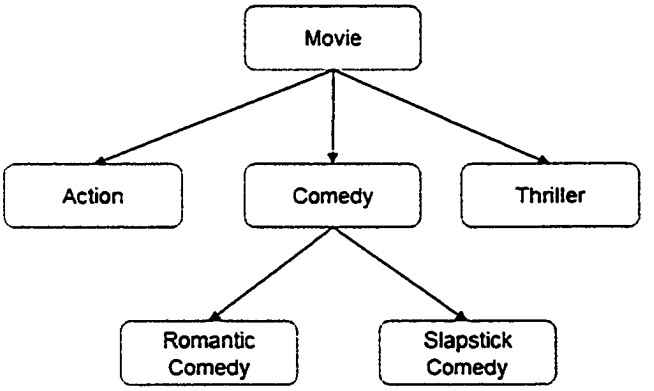


Figure 2.6: Sample class hierarchy

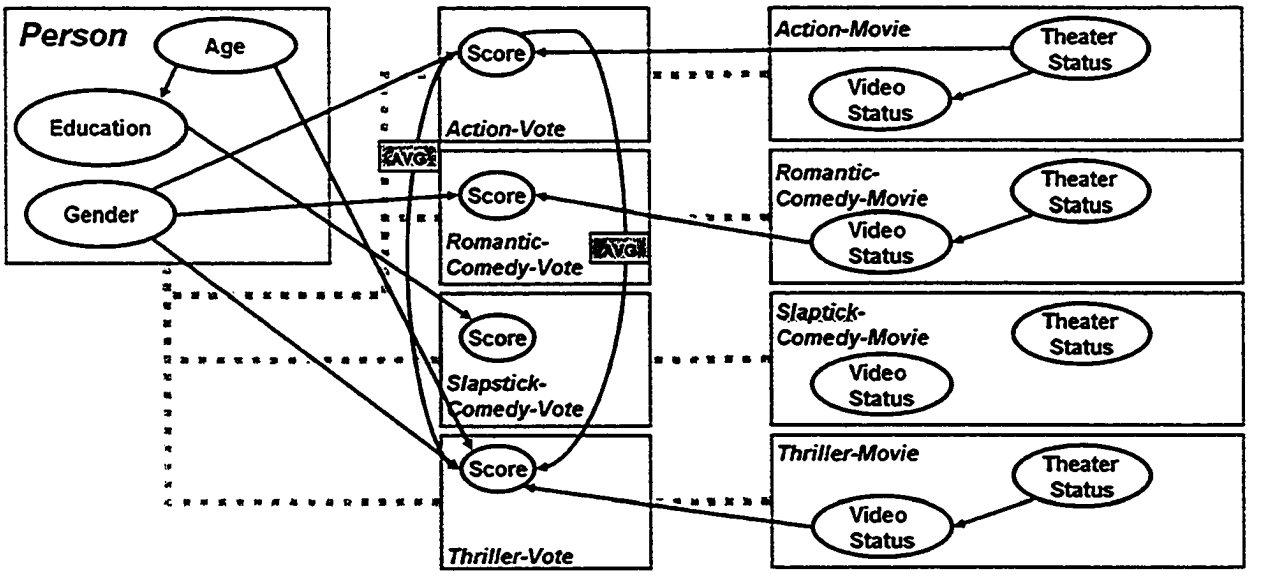


Figure 2.7: Example hPRM for EachMovie dataset

Action movie.

More formally, a hierarchical Probabilistic Model is:

Definition 1 *The components of a* Hierarchical Probabilistic Relational Model (hPRM) Π_H *are:*

- *A class hierarchy* $H[X] = (C[X], \prec)$
- *A set of basic, leaf-node elements* $basic(H[X]) \subset H[X]$
- *A subclass indicator attribute* $X.Class \in basic(H[X])$
- *For each subclass* $c \in C[X]$ *and attribute* $A \in \mathcal{A}(X)$ *we have a specialized CPD for* c *denoted* $P(X^c.A|Pa^c(X.A))$*;* Pa^c *represents the parent nodes for* $X.A$ *in subclass* c*.*
- *For every class* Y *reachable via a reference slot chain from* X*, we have a specialized CPD for* c *denoted* $P(Y^c.A|Pa^c(Y.A))$

2.4.2 Learning hPRMs

The algorithm for learning an hPRM is very similar to the algorithm for learning a standard PRM. Instead of dealing with the standard set of classes $\mathcal{X}$ when evaluating structure quality and estimating parameters, our hPRM algorithm dynamically partitions the dataset into the subclasses defined by $H[X]$. We accomplish this by dynamically modifying the SQL queries and views constructed of the database to reflect information that applies to the hierarchy element of interest.

2.5 hPRM Inference

Inference for hPRMs is generally performed in the same way that standard PRM inference, except that for hPRMs, objects in the instance $\mathcal{I}$ must dynamically be mapped to the appropriate hierarchy element $basic(H[X])$ and the corresponding dependency structure and parameter set. For any given instance i of a class, i's place in the hierarchy is flagged through `X.Class`; using this flag it is possible to associate the proper CPD with a given class instance.

Chapter 3

Applying PRMs to the Recommender System Domain

3.1 Overview

In this section we describe our implementation of the framework described in the previous section, and show how we apply the framework to the recommender system problem domain.

Unlike Bayesian Networks, PRMs are a relatively new research area, and the tools for conducting research into PRMs and their relatives are not well-developed. Our search for a software package that would allow us to learn PRMs and hPRMs from data identified only one contender: the *PHROG* system developed at Daphne Koller's group at Stanford [5]. While Lise Getoor was kind enough to give us the source code for this package, we found it to be extremely specialized in its applicability, and we realized that modifying the *PHROG* system to conduct the kind of research we were interested in would involve a significant amount of work.

Thus, rather than risk spending a large amount of time modifying a code base that we understood poorly (and having no guarantee that this modification would be successful), we implemented a PRM and hPRM system called *Tadpole*. This system is, to the best of our knowledge, the first implemented hierarchical PRM learning system

3.1.1 Database Connectivity

We designed *Tadpole* system so that its structure learning and parameter estimation algorithms can operate directly on a relational database. This addresses one of the traditional weaknesses of machine learning algorithms, which is the impedance mismatch between the format most real-world data is stored in and input format required my most machine learning algorithms. While most real-world data is stored in relational database management systems (such as Microsoft SQL Server or Oracle), virtually all machine learning algorithms require a flat-file, propositionalized representation of that data as in input. More specifically, most machine learning algorithms require that each "object" be represented as a single tuple in a data file, where each tuple consists of a fixed set of attribute values.

The task of transforming relational data to a propositionalized format is one that is often downplayed as a "preprocessing" step in many machine learning papers. However, transforming data into a propositionalized format not only often represents a large amount of effort on the part of the user, but also introduces two potentially serious issues.

First, by flattening the relational domain into a propositionalized format, a large amount of valuable structural information is lost from the problem domain. We therefore cannot model any statistical relationships that may exist between "linked" or "related" objects.

Second, by flattening the relational model, we may introduce significant statistical skews into our propositionalized representation. For example, consider a simple relational database

consisting of three tables: people, products, and a table enumerating the purchases people have made. When propositionalizing this data, a person who makes 100 purchases will have 100 tuples in the propositionalized representation, and a person who only makes 1 purchases will have one entry. If we are trying to model, for example, which attributes of a person influence what objects they buy, the person who purchased 100 items will be over-represented, and we will discover spurious dependencies in the dataset.

The fundamental issue here is that by propositionalizing the data, we lose the concept of an "object" and relationships that exist among those objects. In our *Tadpole* system, the input to our algorithm is a database connection, and we learn directly on the relational representation of the data, where objects and relationships are first-class citizens.

We implemented *Tadpole* in the *Java* programming language, which provides us with a rich substrate of APIs to build upon. In particular, we make extensive use of the Java Database Connectivity (JDBC) API to allow our system to connect directly to a relational database that provides a JDBC driver, such as MySQL, Oracle, or Microsoft SQL Server. The user simply provides *Tadpole* with the relevant connection information, such as an IP address, username, and password, and *Tadpole* will automatically construct appropriate internal data structures and will dynamically create appropriate SQL queries for the given database.

3.1.2 Schema Metadata

In addition to the database JDBC connection information, *Tadpole* requires some additional meta-information about the database. This is because the database connection in isolation does not provide enough information for our *Tadpole* to learn a PRM; other types of information, such as the relationships among the tables in the database, need to be provided.

Again, to make our system as extensible as possible, we implemented the schema metadata facility as an XML document, which can be easily extended to incorporate additional types of information. In its current form, the schema metadata contains the following elements:

- *Class*: Represents a PRM-level Class item, and provides mapping information between the PRM class name and the database name .
  - *Attribute*: represents the PRM-level attribute items that are contained within a Class, defines whether they are a descriptive attribute or part of the skeleton, and provides a mapping between the PRM Attribute name and the database name.

- *ReferenceSlot*: Represents a PRM-level Reference Slot, which defines how the various classes of the PRM are related to each other.
  - *JoinElement*: Related PRM classes map to specific database tables, and every ReferenceSlot has one or more *JoinElement*, which defines which of the attributes within the related classes must "match" in order for two objects of the related classes to be linked. In database terminology this metadata item defines the "join" between two tables (and thus two classes).

A sample XML metadata file is show below.

```
<prm>
  <schema name="eachMovie" dbName="eachMovie" description="EachMovie Domain">
    <class name="Person" dbName="person" description="A person">
      <attribute name="id" dbName="person_id" primaryKey="true" foreignKey="false"
        type="Integer" dbType="INT" description="ID number"/>
      <attribute name="gender" dbName="gender" primaryKey="false" foreignKey="false"
        type="String" dbType="String" description="Gender"/>
      <attribute name="zipCode" dbName="zipcode" primaryKey="false" foreignKey="false"
        type="Integer" dbType="INT" description="ZipCode"/>
      <attribute name="age" dbName="age_int" primaryKey="false" foreignKey="false" type="Integer"
        dbType="Integer" description="Age"/>
      <attribute name="education" dbName="education" primaryKey="false" foreignKey="false"
         type="String" dbType="String" description="Education level"/>
    </class>
    <class name="Vote" dbName="vote" description="A person's vote on a movie">
      <attribute name="id" dbName="vote_id" primaryKey="true" foreignKey="false"
        type="Integer" dbType="INT" description="Vote ID"/>
      <attribute name="person" dbName="person" primaryKey="false" foreignKey="true"
        type="Integer" dbType="INT" description="Person"/>
      <attribute name="movie" dbName="movie" primaryKey="false" foreignKey="true"
        type="Integer" dbType="INT" description="Movie"/>
      <attribute name="score" dbName="score" primaryKey="false" foreignKey="false"
        type="Float" dbType="Float" description="Ranking of movie"/>
    </class>
    <class name="Movie" dbName="movie" description="A movie">
      <attribute name="id" dbName="movie_id" primaryKey="true" foreignKey="false"
        type="Integer" dbType="INT" description="Movie ID"/>
      <attribute name="theaterStatus" dbName="theater_status" primaryKey="false"
        foreignKey="false" type="String" dbType="String" description="Theater status"/>
      <attribute name="videoStatus" dbName="video_status" primaryKey="false" foreignKey="false"
        type="String" dbType="String" description="Video status"/>
    </class>
    <referenceSlot name="voteOfPerson" inverseName="vote" fromClass="Vote" toClass="Person"
      guaranteedAcyclic="false" type="ManyToOne">
      <joinElement fromAttribute="person" toAttribute="id"/>
    </referenceSlot>
    <referenceSlot name="voteOnMovie" inverseName="vote" fromClass="Vote" toClass="Movie"
      guaranteedAcyclic="false" type="ManyToOne">
      <joinElement fromAttribute="movie" toAttribute="id"/>
    </referenceSlot>
```

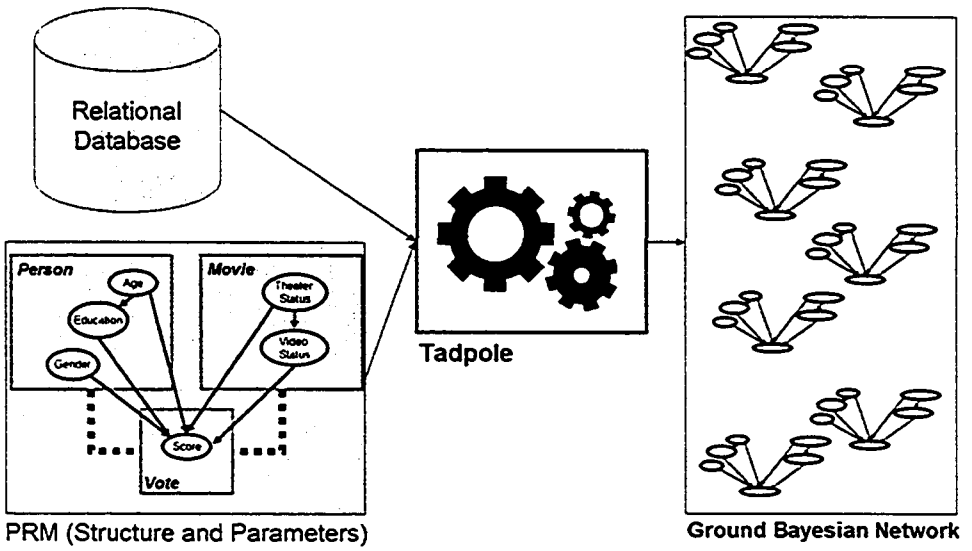


Figure 3.1: Generating a ground Bayesian Network with the *Tadpole* PRM/hPRM inference system.

```
  </schema>
</prm>
```

3.2 Inference

In some cases, learning a PRM from data, and elucidating characteristics of the data from the resulting dependency structure, may be the end-point of the *Tadpole* system. However, many cases, one may wish to use the learned PRM to perform some kind of inference. In this scenario, one would learn a PRM on a given set of data, then apply the learned PRM to a new set of objects contained in another database to generate a ground Bayesian Network. With this ground Bayesian Network in hand, one can then apply the wide variety of standard Bayesian Network inference algorithms available to infer the posterior probability of certain value assignments to nodes in the ground Bayesian Network (for more details see 2.2.2).

We have to perform this kind of inference in many of our experiments, and as such have made an easy-to-use mechanism to generate a ground Bayesian Network from an input PRM and an input relational database. The output Bayesian Network can be encoded in a variety of formats, such as XBN [31] and used in a wide variety of Bayesian Network inference engines. Figure 3.1 outlines the workflow used to generate a ground Bayesian Network.

3.3 Applying Standard PRMs to the EachMovie Dataset

PRMs provide an ideal framework for capturing the kinds of dependencies a recommender system needs to exploit. In general, recommender systems try to capture high-level patterns in data that provide some amount of predictive accuracy. For example, in the EachMovie dataset, one may want to capture the pattern that teenage males tend to rate action movies quite highly, and subsequently use this dependency to make inferences about unknown votes. PRMs are able to model such patterns as class-level dependencies, which can subsequently be used at an instance level to make predictions on unknown ratings — i.e., how will John vote on `SW`.

In order to use a PRM to make predictions about an unknown rating, we must first learn the PRM from data. In our experiments we use the PRM learning procedure described in [9], together with the hierarchical learning extensions outlined in [11], which provide a algorithms for both learning a legal structure for an (h)PRM and estimating the parameters associated with that (h)PRM. Figure 2.4(a) shows a sample PRM structure learned from the EachMovie dataset.

We can then use the learned PRM to infer a new, previously unseen `Vote.score`. (That is, we want to know the distribution of the variable x.`score`, based on the properties of x.`ofPerson` and x.`ofMovie` (i.e., the values of x.`ofPerson`.y and x.`ofMovie`.z, for various attributes y and z) as well as other facts — depending on the information specified and the structure of the PRM.

To accomplish this task, we leverage the *ground Bayesian Network* [11] induced by a PRM. Briefly, a Bayesian Network is constructed from a database using the link structure of the associated PRM's dependency graph, together with the parameters that are associated with that dependency graph. For example, for the PRM in Figure 2.4(a), if we needed to infer the `score` value for a new `Vote` object, we simply construct a ground Bayesian Network using the appropriate attributes retrieved from the associated `Person` and `Movie` objects; see Figure 2.4(b). The PRM's class-level parameters for the various attributes are then tied to the ground Bayesian Network's parameters, and standard Bayesian Network inference procedures can be used on the resulting network [11].

3.4 Applying hPRMs to the EachMovie Dataset

Applying the hPRM framework to the EachMovie dataset first requires a hierarchy to be defined, which is then used to build an hPRM that is ultimately used to make predictions for unknown votes.

In our experiments we automatically learn a hierarchy to be used in the learning procedure. In the EachMovie database, a movie can belong to zero or more of the following genre categories: { `action`, `animation`, `art_foreign`, `classic`, `comedy`, `drama`, `family`, `horror`, `romance`, `thriller` }.

We let $G(\chi)$ denote the set of genres that the movie χ belongs to. For example, $G(\texttt{WhenHarryMetSally}) = \{\texttt{comedy}, \texttt{drama}, \texttt{romance}\}$. To build our hierarchy dynamically, we first enumerate all combinations of genre-sets that actually appear in the EachMovie database. Of course, this set (denoted $\mathcal{G}$) is significantly smaller than the entire 2^ℓ power-set of all possible subsets of the ℓ genres. We also store the number of movies associated with each element of $\mathcal{G}$. We then proceed to greedily partition $\mathcal{G}$ (i.e., we create as large partitions as possible) based on this quantity, until reaching a predefined limit of k partitions. (Here, we used $k = 11$.) We define one additional partition that is used for movies that do not fall into one of the predefined partitions. This partition, together with the other k partitions, are used to create a $k+1$-element hierarchy. In our experiments, this algorithm created partitions that reflect common pairings of movie genres (such as romantic comedies).

Given this hierarchy, the hPRM learning algorithm is applied to the EachMovie dataset, using the same algorithm used for learning standard PRMs (Section 3.3), with the exception that the learning procedure is modified as outlined above.

3.5 The *Tadpole* System

The *Tadpole* system is capable of doing structure and parameter learning for a PRM/hPRM directly from data. We wanted to implement *Tadpole* in the most general way possible, so that the system could continue to be used on new problem domains and extended in new ways beyond the completion of this thesis.

Figure 3.2 shows the overall inputs and outputs of the *Tadpole* system.

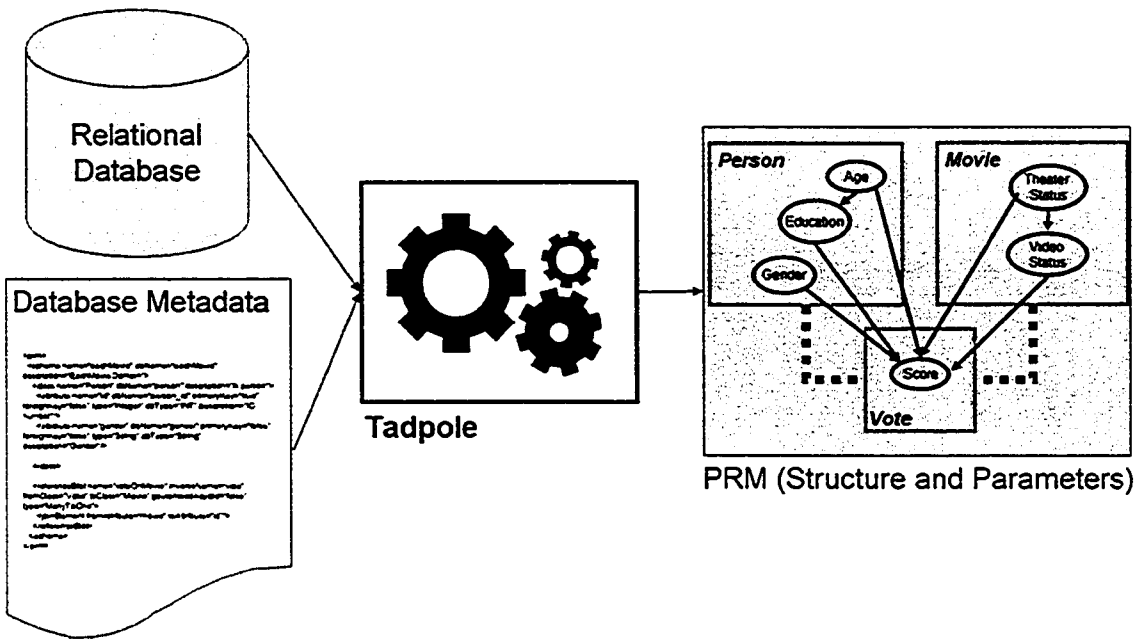


Figure 3.2: The inputs and outputs of the *Tadpole* PRM/hPRM learning system.

Chapter 4

Empirical Results

This section outlines our results in applying both standard PRMs and hPRMs to the recommendation task for the EachMovie dataset. We also compare our results to other recommendation algorithms.

4.1 Experimental Design

One of the main challenges in designing an experiment to test the predictive accuracy of a PRM model is in avoiding resubstitution error. If a PRM is learned on the entire EachMovie database, and subsequently used to make predictions on objects from the same database, we are using the same data for testing as we used for training. By using the same data for testing and training, we are skewing our estimates of what our generalization error will be; a standard tool in machine learning to help mitigate this effect is *cross-validation*. The standard n-fold cross validation algorithm first splits the data into n subsets, trains on $(n-1)/n$ of the data, and test of the remaining $1/n$ subset. This simple approach does not apply here, as it is impossible to divide the data into "independent" compartments, as the movies, people, and votes are intertwined. Hence, the relational learning environment is significantly different from the standard, non-relational scenario where each data row is independently and identically distributed (iid).

We address this issue by applying a modified cross-validation procedure to the dataset. While the traditional method of dividing data into cross-validation folds cannot be applied directly to a relational database, we extend the basic idea to a relational setting as follows. For n-fold cross validation, we first create n new datasets $\{D_1, \ldots, D_n\}$ with the EachMovie data schema. We then iterate over all the objects in the `Person` table, and randomly allocate the individual to one of $D_i \in \{D_1 \ldots D_n\}$. Finally, we add all the `Vote` objects linked to that individual, and all the `Movie` objects linked to those `Vote` objects, to D_i. This procedure, when complete, creates n datasets with roughly balanced properties, in terms of number of individuals, number of votes per person, etc. In our experiments we use $n = 5$-fold cross validation.

4.2 Evaluation Criteria

In this dissertation we adopt the *Absolute Deviation* metric [2, 19] to assess the quality of our recommendation algorithms. We divide the data into a training and test set using the method described above, and, for each fold, build a PRM (resp., hPRM) using the training data. We then iterate over each user in the test set, allowing each user to become the

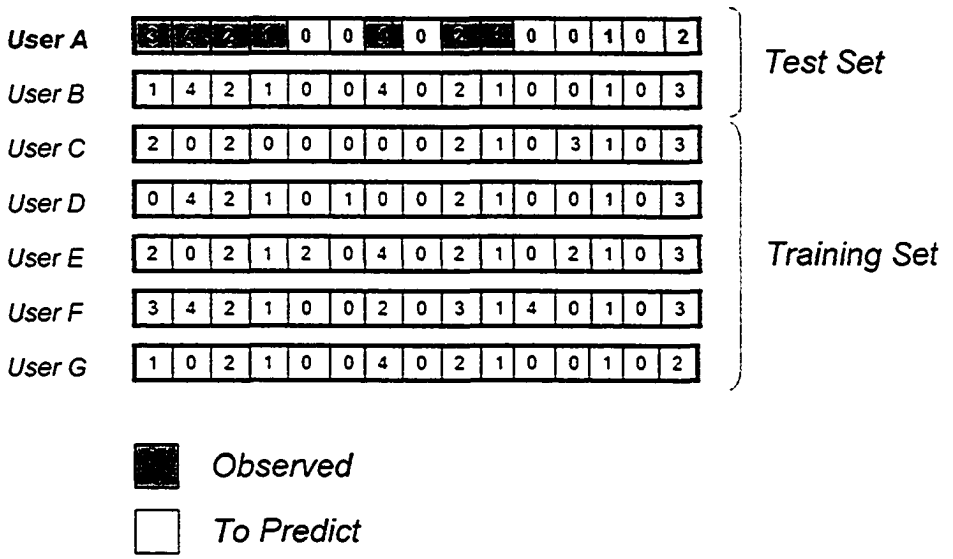


Figure 4.1: Cross-validation architecture for recommender systems.

active user. For the active user we then iterate over his set of votes, P_a, allowing each vote to become the *active vote*; each of the remaining votes are used in the PRM model.

To make this concrete, consider the example depicted in Figure 4.1. In this example *User A* and *User B* constitute the *Test* fold; the rest of the users are part of the *Training* set that the model will be learned upon. The dark-shaded votes constitute our observed votes for the active user, which in this case is *User A*. The light-shaded votes are the votes that our recommender system must predict. The recommender system's performance on this prediction is how we will compare various recommender system implementations.

We let $p_{a,j}$ denote the predicted vote for the active user a on movie j, and $v_{a,j}$ denote the actual vote. The average absolute deviation, over the m_a vote predictions made, is:

$$S_a = \frac{1}{m_a} \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \tag{4.1}$$

The absolute deviation for the dataset as a whole is the average of this score over all the users in the test set of users.

We compare our results using the data and metrics described in [2]. A recent paper, [18], achieves an Absolute Deviation (described below) score of 0.962, but the experimental setup used varies significantly from that described in [2] (for example, only users who have voted on over forty movies are included), and is therefore not included in the comparisons to follow.

Algorithm	Absolute Deviation
CR	1.257
BC	1.127
BN	1.143
VSIM	2.113
PRM	**1.26**

Table 4.1: Absolute Deviation scoring results for EachMovie dataset

Algorithm	Absolute Deviation
CR	0.994
BC	1.103
BN	1.066
VSIM	2.136
hPRM	**1.060**

Table 4.2: Absolute Deviation scoring results for EachMovie dataset

4.3 Standard PRMs

In our experiments with Standard PRMs, we were able to achieve an absolute deviation error of 1.26. For comparison, Table 4.3(a) includes the results from [2]: correlation (CR), Bayesian Clustering (BC), a Bayesian Network model (BN), and Vector Similarity (VSIM). We have elected to include the results from [2] where algorithms were given two votes out of the non-active votes to use in making the prediction, since the standard PRM model does not have any direct dependency on other `Votes`.

In this experiment, standard PRMs are able to outperform the VSIM algorithm, and is competitive with the correlation-based algorithm. However, both Bayesian Clustering and the Bayesian Network model have superior results in this context.

4.4 Hierarchical PRMs

The first part of the experiment for hPRMs was constructing a class hierarchy from the EachMovie dataset. In our experiment we set the size of the hierarchy to be 12. Our greedy partitioning algorithm arrived at the following basic classes: { `drama`, `comedy`, `classic`, `action`, `art+foreignDrama`, `thriller`, `romance+comedy`, `none`, `family`, `horror`, `action+thriller`, `other` }.

By applying hPRMs to the EachMovie dataset, we are able to reduce the absolute deviation error from 1.26 (with standard PRMs) to 1.06. Again, for comparison Table 4.4(b) includes results from [2]; however, since hPRMs are able to leverage other votes the user has made in making predictions, we use the *All-But-One* results presented

in [2], where the prediction algorithm is able to use all of the active user's votes (except for the current active vote) in making a prediction. Comparing Table 4.3(a) to Table 4.4(b), we see that including the additional voting information results in a substantial reduction in error rate for most of the other four algorithms.

hPRMs not only provide a significant performance advantage over standard PRMs, but are also able to outperform all but one of the other four algorithms.

Chapter 5

Literature Review

This section we review literature relevant to our approach to the recommender system problem domain.

We first examine the pure collaborative filtering approaches in section 5.1. Next, we discuss content-based recommender systems in section 5.2. Finally, in section 5.3 we discuss some approaches that have been taken to combining pure collaborative filtering approaches with content-based systems.

5.1 Pure Collaborative Filtering Methods

A seminal work on traditional collaborative filtering algorithms is Breese *et al.*'s analysis of the relative performance of various collaborative filtering algorithms [2]. In this paper Breese *et al.* noted that while there had been a wide variety of collaborative filtering algorithms, there was a distinct lack of quantitative analysis of the relative performance of those various algorithms. [2] develops a taxonomy for categorizing various collaborative filtering algorithms, and defines a variety of metrics and test data sets for evaluating those algorithms.

In terms of pure collaborative filtering algorithms, Breese *et al.* define two major classes: *memory-based* algorithms and *model-based* algorithms. *Memory-based* algorithms operate over the entire data set to make predictions. An example of such an algorithm would be a vector-similarity based algorithm, which must evaluate a user's vector of votes in terms of its similarity to *all* the vectors of votes of other users in order to make a prediction. *Model-based* algorithms, on the other hand, use the data set to build a model of user preferences, which is then used to make predictions. An example of this kind of system would be a Bayesian Network whose structure and parameters were learned from the data set; future predictions are then made using the Bayesian Network.

Breese *et al.* consider two memory-based collaborative filtering algorithms in their comparison. The first, Correlation, first appeared in the context of the GroupLens project [25]. The correlation between users a and i is based on the Pearson correlation coefficient:

$$\mathrm{w}(a,i) = \frac{\sum_j (v_{a,j} - \overline{v}_a)(v_{i,j} - \overline{v}_i)}{\sqrt{\sum_j j(v_{a,j} - \overline{v}_a)^2 \sum_j (v_{i,j} - \overline{v}_i)^2}} \tag{5.1}$$

where the summations of j only include items where both users a and j have votes.

The second memory-based algorithm, Vector Similarity, uses a metric similar to the document similarity metric used in the information retrieval field. To assess the similarity between two documents, a "bag-of-words" representation is used, where a document is first

reduced to a vector of numbers representing how many times various words occurred in the document. To assess the similarity between two documents, the cosine of the angle between the two word frequency vectors of the respective documents is used [27]. We can use this framework in the context of collaborative filtering by regarding each user as a document, where the frequency vector instead represents the vector of votes the user assigned to various movies. The weight measuring the similarity between user a and user i is now:

$$\mathrm{w}(a,i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}} \tag{5.2}$$

(the denominator is simply a normalization term.)

The second major class of algorithms Breese *et al.* discuss is *model-based* algorithms. With the previously discussed algorithm, we make use of the entire user dataset in order to make predictions. With the following model-based algorithms, we instead learn a model that (hopefully) models the dependencies present in the dataset; at prediction-time, we simply use this model to make a prediction for a user.

Breese *et al.* present two model-based algorithms: a Bayesian Clustering model, and a Bayesian Network model. The Bayesian Clustering model is essentially a multinomial mixture model that assumes that users belong to a hidden class that captures their preferences and tastes; given this class, their votes on various items are independent. However, since this class variable is hidden, the EM algorithm [7] must be used to learn the parameters for this model with a fixed number of classes (in order to pick the appropriate number of classes, Breese *et al.* simply used the model that yielded the largest approximate marginal likelihood of the data).

The second model-based algorithm Breese *et al.* employed was a Bayesian Network model. In this model, a user's vote on a given item depends probabilistically on his/her votes on other specific items; which items it depends on, and the parameters that are associated with this model, are learned using a Bayesian Network learning algorithm described in [3].

Finally, Breese *et al.* applied the above-mentioned algorithms to a set of three datasets using two scoring metrics. Both scoring metrics have common evaluation framework: the data set of users (and votes) is divided into a training set and a test set. The training set is used as the collaborative filtering database for memory-based algorithms and used to learn the model in model-based algorithms. The test set of users is then iterated over, and each user takes a turn being the *active* user. The votes for the active user is then partitioned into a set of *observed* votes, I_a, and a set that will be predicted, P_a.

The first scoring metric, mean absolute deviation, is the most commonly used metric in assessing collaborative filtering algorithms, and was also used in the GroupLens project [25]. For a given user, we simply iterate over the hold-out votes, and measure the absolute deviation between the predicted vote and the actual vote (the number of hold-out votes is denoted m_a):

$$S_a = \frac{1}{m_a} \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \tag{5.3}$$

This score is then averaged over all the users in the test set of users to give the overall mean absolute deviation (thus, heavy users - users with many ratings - are weighted the same as light users).

The second scoring metric attempts to model how users typically user a recommender system. The main assumption is that users are only mainly concerned with the *highest* recommended items in a ranked list of recommendations, and the importance of each successive item in the list decays exponentially. The expected utility of a ranked list of items (sorted by index j in order of declining $v_{a,j}$) for a user:

$$R_a = \sum_j \frac{max(v_{a,j} - d, 0)}{2^{(j-1)(\alpha-1)}} \tag{5.4}$$

where d is the neutral vote and α is the viewing halflife, where there is a 50-50 chance the user will view the item. The overall ranking of an algorithm using this ranked score metrics is

$$R = 100 \frac{\sum_a R_a}{\sum_a R_a^{max}} \tag{5.5}$$

where R_a^{max} is the maximum achievable utility if the predicted order of items matched the observed votes ranked by vote value.

Finally, Breese *et al.* used four different protocols for deciding how many of user's votes were treated as observed votes. These protocols were denoted *Given2*, *Given5*, *Given10*, and *AllBut1*, which allocate two, five, 10, and all but one of the user's votes to the observed votes pool (I_a), respectively; the remainder of the votes belonged to the pool of votes to be predicted (P_a).

Breese *et al.* tested the above algorithms against three datasets; however, only the EachMovie dataset is publicly available. We will focus on the EachMovie dataset in the following discussion since it is one of the only widely-used datasets in recommender systems, and is the dataset we have used in our experiments. The tables below summarize the results of the experiments presented in [2]:

As can be seen in these results, the Bayesian Network and Correlation algorithms generally outperform the Bayesian Clustering and Vector Similarity methods.

Algorithm	Given2	Given5	Given10	AllBut1
Correlation	41.60	42.33	41.46	23.16
Vector Similarity	42.45	42.12	40.15	22.07
Bayesian Clustering	38.06	36.68	34.98	21.38
Bayesian Network	28.64	30.50	33.16	23.49

Table 5.1: Ranked score for EachMovie dataset

Algorithm	Given2	Given5	Given10	AllBut1
Correlation	1.257	1.139	1.069	0.994
Vector Similarity	2.113	2.177	2.235	2.136
Bayesian Clustering	1.127	1.144	1.138	1.103
Bayesian Network	1.143	1.154	1.139	1.066

Table 5.2: Mean absolute deviation score for EachMovie dataset

5.2 Content-Based Methods

On the opposite extreme of the recommender system spectrum from purely collaborative recommender systems is purely content-based methods. Content-based methods build a model of what users prefer not by modelling their similarity to other users, but instead by trying to find commonalities among the items that a user prefers. For example, an Amazon.com book recommender may, based on a user's past purchases, recommend books to that user that are of the *Murder-Mystery* genre and published between 1950 and 1975. More generally, a content-based recommender system will build a model of a user's preferences based on attributes (the *content* of the items the user is voting upon).
A popular approach to pure content-based recommender systems is based on a bag-of-words naive Bayesian text classifier algorithm [20]. This algorithm has been employed in the context of book recommender systems [21] and, more recently, on the EachMovie dataset [18]. In this framework, the prediction task is viewed as a text-categorization problem, where a movie corresponds to the document, the document's content is composed of information retrieved from the Internet Movie Database (IMDB), and every document is labelled with the user's rating of the document.
In [21], Mooney *et al.* use a multinomial text model [16], where a document is modelled as an ordered sequence of word events drawn from the same vocabulary, V. The naive Bayes assumption states that the probability of each word event is dependent on the document class but independent of the word's context (i.e., its position). The probability of a document (movie) belonging to a given class (a given rating) can the be calculated using Bayes' rule; the class with the highest posterior probability is the predicted rating.

Using pure content-based collaborative filtering, Mooney *et al.* were able to achieve a mean absolute deviation of approximately 1.45 using the *Given2* protocol, approximately 1.40 using the *Given5* protocol, and approximately 1.31 using the *Given10* protocol. Unfortunately, in these results, Mooney *et al.* did not use the same experimental framework as Breese *et al.* in [2], and did not present results using the *AllBut1* protocol. Furthermore, the EachMovie dataset was not used in an unadulterated form, but was rather filtered to only include users that had rated forty movies. This filtering would bias the scoring metrics downwards, and are therefore difficult to compare directly to [2].

5.3 Combined Approaches

Recent work on recommender systems, including this thesis, have begun focussing on methods to combine aspects of both pure collaborative filtering systems with content-based recommender systems.

There are several strong arguments for combining the two approaches. First, with either extreme, we are using less information than is available to us; by taking advantage of both types of information, we have the opportunity to outperform pure collaborative filtering and pure content-based methods. Another strong argument for incorporating content information into recommender algorithms is to help mitigate the *data sparsity* problem that is common in collaborative-filtering-based recommender systems. The data sparsity problem refers to the fact that most users have not rated most items, and therefore the user-item rating matrix is generally extremely sparse. New users to a recommender system ,for example, typically enter only a few ratings, then want to view what their recommendations are. With a severe lack of vote information, the content-based methods can be viewed as a safety net of sorts, where the content of a user's rated movies factors very heavily into the prediction process. For example, consider a user who has only rated three movies: *True Lies* with a rating of 5; *Terminator 2* with a rating of 4; and *Armageddon* with a rating of 5. Given this sparse rating vector, a purely collaborative-filtering based system would have a difficult time making effective recommendations, but a content-based system could easily recommend more movies of the *Action* genre, and in particular *Action* movies that perhaps star Arnold Schwartzenegger. Since this is a new area of research, there has been relatively few publications on methods for combining content-based and collaborative filtering based algorithms. In [18], a *pseudo user-ratings vector* is created for every user u in the database. This pseudo vector is composed of the votes the user has *actually* made, along with votes predicted using a

content-based recommender system for all the items the user hasn't voted on. The pseudo user-rating vectors for all the users in the system forms a dense pseudo-ratings matrix V, and standard collaborative-filtering based methods can be applied to this dense matrix. Using a modified version of the EachMovie dataset, this approach was able to achieve a Mean Absolute Deviation score of 0.962.

Chapter 6

Conclusion

In this dissertation we outlined a framework for using PRMs to model the recommendation task, and applied it to one of the most-studied recommender tasks, the *EachMovie* dataset. We were able to demonstrate that PRMs provide a powerful and expressive way of modelling the recommender system domain. In our experiments, we were able to demonstrate that PRMs and hPRMs, with no further tuning, were able to achieve competitive results against algorithms engineered specifically for the recommender task. Furthermore, PRMs offer an extensible framework for integrating a wide variety of data sources and types in the future. Finally, this dissertation provides a case study of how hPRMs improve the expressiveness and context-sensitivity of standard PRMs, and also realize real-world performance benefits.

6.1 Future Work

In this dissertation we presented promising results indicating the applicability of PRMs in general to the recommender system problem, and demonstrated the efficacy of statistically modelling hierarchical datasets using hierarchical PRMs.

However, it is difficult to make general statements about the overall effectiveness of this approach given results from a single dataset; validation on multiple datasets would be more persuasive. This single biggest obstacle here is, unfortunately, a lack of available datasets. The *EachMovie* dataset is one of the most extensive available, and most other interesting datasets are private and not disclosed (for example, the Amazon.com, Yahoo! LAUNCHCast, and CDNow! datasets). As such, most of the recommender system literature uses the *EachMovie* dataset as the sole metric for the efficacy of a given algorithm, which may result in algorithms that are tailored to perform well against this one dataset, but do not perform well in a more general context.

In the context of learning hPRMs, when we are given a dataset without a predefined hierarchy, as was the case with the *EachMovie* dataset, we used a greedy, but somewhat naive algorithm to learn a hierarchy. Although the learned hierarchy was reasonable and similar to a hierarchy that a human being might design, it would be possible incorporate the learning of the hierarchy directly into the PRM learning algorithm, where new leaf nodes are introduced in the hierarchy if that alteration would improve the overall likelihood of the model. Various alterations to the hierarchy could be considered at each step of the learning algorithm, just as various alterations to the PRM structure are considered at each step. The computational cost of this improvement would be high, but may lead to significantly improved results.

To further test the efficacy of hierarchical PRMs, modelling datasets that have intrinsic hierarchies would be a valuable experiment. In the case of the *EachMovie* dataset, we learned a hierarchy of the dataset, since a pre-defined hierarchy was not available. However, some datasets may have implicit hierarchies, and the hPRM could represent this hierarchy directly. Many datasets in the field of bioinformatics are emerging that have this property. For example, the KDD Cup competition in 2002 [14] featured an extensively hierarchical dataset where gene locations and functions were placed in a hierarchy designed by biologists. The task was to predict the function of uncharacterized genes based on a training set of known genes. PRMs, and especially hPRMs, could be applied to this, and many other biological datasets, to further study the efficacy of modelling using hierarchical versus non-hierarchical probabilistic models.

Finally, most evaluation metrics for recommender systems have tended to be very user-centric, where we "leave-one-user-out" and evaluate how we perform on predicting the votes for that user. However, we may want to assess various other aspects of performance, such as when a "new" movie is introduced to the database; how accurately are we able to predict votes for this movie? A typical scenario where this aspect of performance would be important for a recommender system is the "opening weekend" for a movie, where we have no history of votes for the movie on which to base predictions; however, users want to go see a new release, and we need to make as accurate a recommendation as possible given the information we have. Developing new metrics to assess varying usage scenarios for recommender systems is an important direction for future research.

also evaluate various other perfor

6.2 Contributions

Finally, we will summarize the contributions made by this thesis:

- The first application of standard and hierarchical PRMs to the recommender system problem.
- Validation of the above using the *EachMovie* dataset. The empirical results from this validation show both:
  - The overall efficacy of PRMs and hPRMs in the context of recommender systems; results were competitive with similar systems tailored specifically for the recommender task.

- The superiority of hierarchical PRMs over standard PRMs; results with hPRMs were significantly better than with standard PRMs.

- A complete implementation of PRM and hPRM learning algorithms in the Java language called *Tadpole* The features of this implementation include:
  - The ability to learn a complete PRM given two inputs: a connection to a database, and a metadata file describing the relationships between the objects modelled in the library. The output of the algorithm is a completely learned PRM (structure and parameters)
  - A database interface architecture that allows the PRM learning algorithms to operate directly on the database. The data the PRM model is based upon never needs to be transformed or exported to an alternate representation, as the PRM learning algorithm acts directly on the native, relational representation.
  - A highly abstracted database connectivity layer that allows the user to use multiple databases from a variety of vendors; this allows the *tadpole* system to learn models on small-scale *MySQL* databases or enterprise-sized *Oracle* database. Virtually every relational database product in the market is supported by the *Tadpole* system.

Bibliography

[1] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Uncertainty in Artificial Intelligence*, pages 115–123, 1996.

[2] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI98*, pages 43–52, 1998.

[3] David Maxwell Chickering, David Heckerman, and Christopher Meek. A Bayesian approach to learning Bayesian networks with local structure. pages 80–89.

[4] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[5] http://www.robotics.stanford.edu/~koller.

[6] R. Dechter. Bucket elimination: A unifying framework for structure-driven inference, 1998.

[7] A.P. Dempster, N.M.Laird, and D.B.Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal Royal Stat. Soc., Series B*, 39(1):1–38, 1977.

[8] http://research.compaq.com/SRC/eachmovie/.

[9] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI-99*, pages 1300–1309, 1999.

[10] Dan Geiger and David Heckerman. A characterization of the dirichlet distribution with application to learning bayesian networks. In *Proceedings of the 1st Annual Conference on Uncertainty in Artificial Intelligence (UAI-85)*, pages 196–207, New York, NY, 1985. Elsevier Science Publishing Comapny, Inc.

[11] L. Getoor. *Learning Statistical Models from Relational Data.* PhD thesis, Stanford University, 2002.

[12] David Heckerman. *A Tutorial on Learning With Bayesian Networks.* Microsoft Research, Redmond, WA, 1995.

[13] Y. Kai, S. Anton, T. Volker, M. Wei-Ying, and Z. HongJiang. Collaborative ensemble learning. In *UAI-03*, 2003.

[14] 2004. http://www.biostat.wisc.edu/~craven/kddcup/tasks.html.

[15] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proc. of the Fifteenth National Conference on Artificial Intelligence*, pages 580–587, Madison, WI, 1198.

[16] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification, 1998.

[17] Jim Melton and Alan R. Simon. *Understanding the new SQL: a complete guide.* Morgan Kaufmann Publishers Inc., 1993.

[18] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192. American Association for Artificial Intelligence, 2002.

[19] Bradley N. Miller, John T. Riedl, and Joseph A. Konstan. Experience with GroupLens: Making Usenet useful again. In *USENIX*, pages 219–233, 1997.

[20] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[21] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, pages 195–204, San Antonio, US, 2000. ACM Press, New York, US.

[22] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 467–475, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[23] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[24] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, 1988.

[25] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[26] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[27] G. Salton and M. J. McGill. *Introduction to Modern Retrieval.* McGraw-Hill Book Company, 1983.

[28] L. Ungar and D. Foster. Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems*, 1998.

[29] L. Ungar and D. Foster. A formal statistical approach to collaborative filtering. In *CONALD'98*, 1998.

[30] Tim Van Allen and Russ Greiner. Model selection criteria for learning belief nets: An empirical comparison. In *Proc. 17th International Conf. on Machine Learning*, pages 1047–1054. Morgan Kaufmann, San Francisco, CA, 2000.

[31] http://www.research.microsoft.com/xbn.