Hierarchical Probabilistic Relational Models for Collaborative Filtering

Jack Newton NEWTON@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8 Canada

Russell Greiner GREINER@CS.UALBERTA.CA
Department of Computing Science & AICML, University of Alberta, Edmonton, AB T6G 2E8 Canada

Abstract

This paper applies Probabilistic Relational Models (PRMs) to the Collaborative Filtering task, focussing on the EachMovie data set. We first learn a standard PRM, and show that its performance is competitive with the best known techniques. We then define hierarchical PRMs, which extend standard PRMs by dynamically refining classes into hierarchies. This represnetation is more expressive that standard PRMs, and allows greater context sensitivity. Finally, we show that hierarchical PRMs achieve state-of-the-art results on this dataset.

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 challenge faced by these system is predicting what each individual will want.

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 (Mitchell, 1997) to learn a classifier, which we can later use to determine whether a (novel) 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 `People`×`Movies` database, where each tuple lists facts about a person, then facts about a movie, together with a vote (e.g., a number between 1 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 (Breese et al., 1998) and Bayesian Models (Chickering et al., 1997), among other technologies.

Notice this prediction does not consider other people (e.g., people "similar" to John) or other movies (similar to `SW`).

The other main type 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 collaboration-only 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 (Ungar & Foster, 1998b; Ungar & Foster, 1998a), 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 $\langle$`P`, `M`, `V`$\rangle$ records, which specifies that person `P` gave movie `M` the vote of `V`. The challenge is how to use all of this information to predict how `John` will vote on `SW`. Here, we want to use facts about John and about `SW`, and also find and exploit collaborative properties, that deal with people similar to John (in terms of liking similar movies), and movies similar to SoM (in terms of being liked by similar people).

Stepping back, the challenge here is learning a distribution over a set of *databases*, here descriptions of *sets* of people and *sets* of products, as well as their votes. 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.gender 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* movies with some other characteristic (e.g., Genre=action).

Probabilistic Relational Models (PRMs) (Koller & Pfeffer, 1198) were designed to address exactly this type of relational learning and inference problem. This paper 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 (EachMovie,) dataset, then evaluate the effectiveness of an extended version of PRMs called "hierarchical PRMS" (hPRMs) (Getoor, 2002). Our empirical results show that standard PRMs can achieve competitive results on the recommendation task, and then that hPRMs can outperform standard PRMs here.

As PRMs can be viewed as a relational extension of Belief Nets, Section 2 describes standard PRMs by showing how they extend Bayesian Networks; in particular, we provide both inference and learning algorithms here. It then presents our application of the PRM framework to the Recommendation task. Section 3 describes some limitations of standard PRMs for this task; addressing these limitations leads naturally to hierarchical PRMs. We introduce our implementation of hierarchical PRMs, and show how an hPRM can provide a more expressive model of the EachMovie dataset. Finally, Section 4 demonstrates the overall effectiveness of PRMs as a recmmendation system, and in particular the superiority of hPRMs over standard PRMs.

2. 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. However, by performing this transformation we lose the rich relational structure and introduce statistical skews (Getoor, 2002). A PRM can be learned directly on a relational database, thereby retaining and leveraging the rich structure contained therein.

We base our notation and conventions for PRMs on those used in (Getoor, 2002). In general, a PRM is a pair $\langle \mathcal{S}, \theta_{\mathcal{S}} \rangle$ defined over a Relational Schema $\mathcal{R}$, where $\mathcal{S}$ is the qualitative dependency structure of the PRM and $\theta_{\mathcal{S}}$ is the set of associated parameters. The *Relational Schema* $\mathcal{R}$ contains two fundamental elements: a set of *classes*, $\mathcal{X} = \{X_1, \ldots, X_n\}$, and a set of *reference slots* $\{\rho_i\}$ that define the relationships between the classes.

Each class $X \in \mathcal{X}$ is composed of a set of *descriptive attributes* $\mathcal{A}(X)$, which in turn take on a range of values $V(X.A)$. For example, consider a schema describing a domain describing votes on movies. This schema has three classes: Vote, Person, and Movie. For the Vote class, the single descriptive attribute is Score with values $\{0, \ldots, 5\}$; for Person the two descriptive attributes are Age and Gender, which take on values $\{$young, middle-aged, old$\}$ and $\{$Male, Female$\}$ respectively; and for Movie the single descriptive attribute is Rating which takes on values $\{$G, PG, M, R$\}$. Furthermore, a class can be associated with a set of *reference slots*, $\mathcal{R}(X) = \{\rho_1, \ldots, \rho_k\}$. A particular reference slot, $X.\rho$, describes how objects of class X are related to objects in other classes in the relational schema. Continuing our example, the Vote class would be associated with two reference slots: Vote.ofPerson, which describes how to link Vote objects to a specific Person; and Vote.ofMovie, which describes how to link Vote objects 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.

The dependency structure for a PRM defines the parents $Pa(X.A)$ for each attribute $X.A$. The parent for an attribute $X.A$ 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 instance, in the above example, Vote.Score could have the parent Vote.ofPerson.Gender.

In many cases, the parent of a given attribute will take on a *multiset* of values $\mathcal{S}$ in $V(X.\tau.A)$. For example, there could be discover a dependency of a Person's age on their rating of movies in the PRMclassChildren genre. However, we cannot directly model this dependency since the user's ratings on Children's movies is a multiset of values, say $\{4, 5, 3, 5, 4\}$. For such a numeric attribute, we may choose to use the Median database aggregate operator to

reduce this multiset to a single value, in this case 4. In this paper we reduce $\mathcal{S}$ to a single value using various types of aggregation functions.

The following definition summarizes the key elements of a PRM:

Definition 1 ((Getoor, 2002)) A probabilistic relational model (PRM) $\Pi = \langle \mathcal{S}, \theta_{\mathcal{S}} \rangle$ *for a relational schema* $\mathcal{R} = \langle \mathcal{X}, \mathcal{A} \rangle$ *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 Pa($X.A$)*, and a* conditional probability distribution (CPD) *that represents* $P_{\Pi}(X.A|Pa(X.A))$*,*

2.1. 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, model-based collaborative filtering algorithms 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 produce described in (Friedman et al., 1999), which provides an algorithm for both learning a legal structure for a PRM and estimating the parameters associated with that PRM. Figure 1(a) shows a sample PRM structure learned from the EachMovie dataset.

With the learned PRM in hand, we are left with the task of making an inference about a new, previously unseen Vote.score. To accomplish this task, we leverage the *ground Bayesian Network* (Getoor, 2002) 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 1(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 1(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 (Getoor, 2002).

3. Hierarchical Probabilistic Relational Models

3.1. Motivation

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. 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 (Friedman et al., 1999), and thus quality 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 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; or Collaborative Filtering: use person A's rating on a movie to predict person B's rating. 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.PersonOf.Gender, whereas a Documentary may depend on Vote.PersonOf.Age.

3.2. 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 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 $\mathcal{C}[X]$ (Getoor, 2002). For a given $c, d \in \mathcal{C}[X]$, $c \prec d$ 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 paper 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

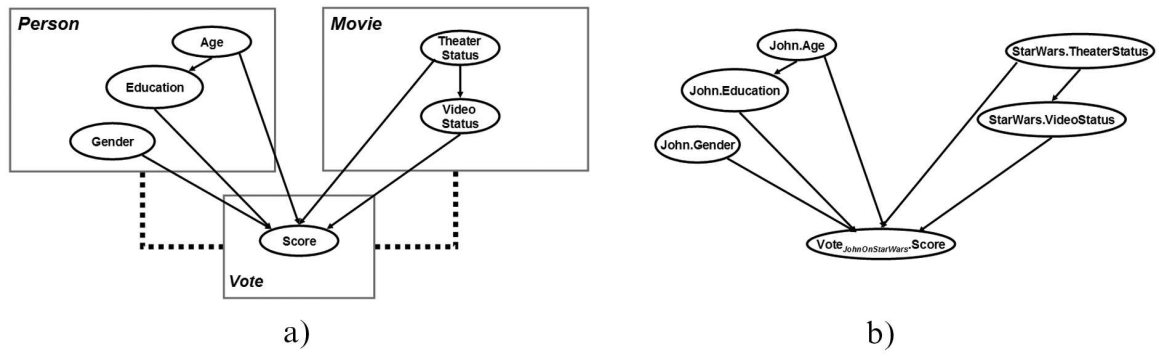

Figure 1. (a) Standard PRM learned on EachMovie dataset (b) Ground Bayesian Network for one `Vote` object

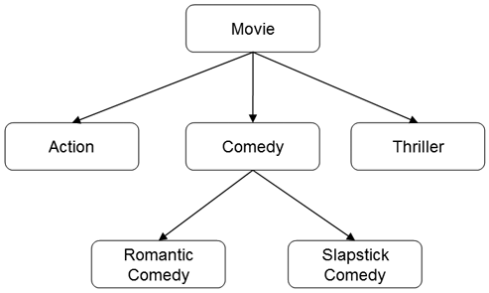

Figure 2. Sample class hierarchy

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 3, the `Vote` class is refined into four leaf classes, each associated with one of the hierarchy elements in $basic(H[X])$.

Definition 2 *The components of a* Hierarchical Probabilistic Relational Model (hPRM) Π_H *are:*

- *A class hierarchy* $H[X] = (\mathcal{C}[X], \prec)$
- *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* $c \in \mathcal{C}[X]$ *and attribute* $A \in \mathcal{A}(X)$ *we have a specialized CPD for* c *denoted* $P(X^c.A|Pa^c(X.A))$
- *For every class* Y *reachable via a reference slot chain from* X *we have a specialized CPD for* c *denoted* $P(Y^c.A|Pa^c(Y.A))$

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 $H[X]$. For inference, a similar technique is used, as 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.

3.3. 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, $G($`WhenHarryMetSally`$) =$ {`comedy`, `drama`, `romance`}. To build our hierarchy dynamically, we first enumerate all combinations of genres that appear in the EachMovie database, and denote this set $\mathcal{G}$. Of course, this set 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 $\mathcal{G}$ 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.

Given this hierarchy, the hPRM learning algorithm is applied to the EachMovie dataset, using the same algorithm used for learning standard PRMs (Section 2.1), with the exception that the learning procedure is modified as outlined

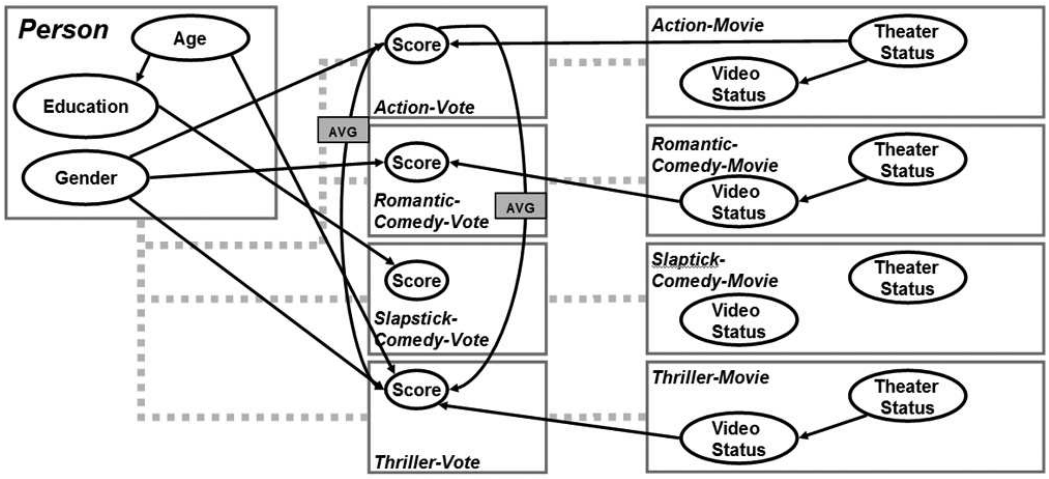

Figure 3. Example hPrm for EachMovie dataset

above.

4. Experimental 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.

The standard learning paradigm typically splits the data into n subsets, and 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 not trivial to divide the data into "independent" compartments, as the movies and people are intertwined

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 5-fold cross validation.

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

Table 1. Absolute Deviation scoring results for EachMovie dataset, using "Two Non-Active Votes" (Breese et al., 1998)

4.2. Evaluation Criteria

In this paper we adopt the *Absolute Deviation* metric (Miller et al., 1997; Breese et al., 1998) to assess the quality of our recommendation algorithms. We divide the data into a training and test set using the method described above, and 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 *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. 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:

$$S_a = \frac{1}{m_a} \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \qquad (1)$$

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.

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

Table 2. Absolute Deviation scoring results for EachMovie dataset, using "All-But-One Votes" (Breese et al., 1998)

4.3. Standard PRMs

In our experiments, we were able to achieve an absolute deviation error of 1.26. For comparison, Table 1 includes the results from (Breese et al., 1998): correlation (CR), Bayesian Clustering (BC), a Bayesian Network model (BN), and Vector Similarity (VSIM). We have elected to include the results from (Breese et al., 1998) 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`, `actionThriller`, `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 2 includes results from (Breese et al., 1998); 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 (Breese et al., 1998), 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 1 to Table 2, 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.

5. Conclusion

In this paper we outlined a framework for using PRMs to model the recommendation task. We first use a standard PRM, then extend this representation to hPRMs, to account for hierarchical relationships that are present in the data. hPRMs improve the expressiveness and context-sensitivity of standard PRMs, and also realize real-world performance benefits.

ACKNOWLEDGEMENTS

We thank Lise Getoor for useful discussions and encouragement (as well as access to her software), Dale Schuurmans for his advice on this project, and thank NSERC, iCORE and the Alberta Ingenuity Centre for Machine Learning for funding.

References

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *UAI98* (pp. 43–52).

Chickering, D. M., Heckerman, D., & Meek, C. (1997). A Bayesian approach to learning Bayesian networks with local structure. *UAI97* (pp. 80–89).

EachMovie. http://research.compaq.com/SRC/eachmovie/.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. *IJCAI-99* (pp. 1300–1309).

Getoor, L. (2002). *Learning statistical models from relational data*. Doctoral dissertation, Stanford University.

Koller, D., & Pfeffer, A. (1198). Probabilistic frame-based systems. *Proc. of the Fifteenth National Conference on Artificial Intelligence* (pp. 580–587). Madison, WI.

Miller, B. N., Riedl, J. T., & Konstan, J. A. (1997). Experience with GroupLens: Making Usenet useful again. *USENIX* (pp. 219–233).

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Ungar, L., & Foster, D. (1998a). Clustering methods for collaborative filtering. *Proceedings of the Workshop on Recommendation Systems*.

Ungar, L., & Foster, D. (1998b). A formal statistical approach to collaborative filtering. *CONALD'98*.