A Framework for Offline Evaluation of Recommender Systems based on Probabilistic Relational Models

Rajani Chulyadyo, Philippe Leray

A Framework for Offline Evaluation of Recommender Systems based on Probabilistic Relational Models

Rajani Chulyadyo[1,2] Philippe Leray[2]

[1]Capacités SAS, Nantes, France
[2]DUKe research group, LS2N Laboratory UMR 6004,
University of Nantes, France
December 2017

Abstract

Recommender systems and their evaluation have been widely studied topics since more than past two decades. Implementation of such systems can be found in numerous commercial and non-commercial software. However, most of the existing open-source/free libraries for recommender systems still deal with single-table data whereas recent studies on recommender systems focus on the use of relational (multi-table, multi-entity) data. In our earlier work (Chulyadyo and Leray [2014]), we had presented a personalized recommender system that works with relational data, and is based on Probabilistic Relational Models (PRM), a framework for modeling uncertainties present on relational data. With the aim to benefit from existing software for evaluating recommender systems, we propose a framework for evaluating such PRM-based recommender systems in this report. The basic idea is to delegate the tasks of evaluation to an external library while the focus for the implementation of the recommender system under study is only on learning a recommendation model and making recommendations from it. Our proposed evaluation framework is generic, and should not be limited only to PRM-based recommender systems. Any recommender systems dealing with relational data can follow this evaluation framework.

1 Introduction

Recommender systems aim at assisting users in making decisions by suggesting them the items that might be interesting for them. Such systems are widely adopted in a large number of domains, such as on-line shopping, news portal, music/videos websites, on-line travel agencies, on-line courses etc. to name a few. The growing popularity of such systems has led to the development of numerous methods for recommendations. Most of these methods use flat data (or

single-table) data that contain the information about the interactions between users and items (aka user-item matrix). However, recent works advocate the use of additional information (such as location, context, demographics, explicit or implicit information objects associated with users and/or items, etc.) in the form of relational data to obtain improved recommendation. In the same direction, we had proposed a recommender system based on Probabilistic Relational Models (PRMs) (Friedman et al. [1999], Getoor [2002]) with the aim to ameliorate the quality of recommendations by exploiting various properties of users and items along with those of objects related to them (Chulyadyo and Leray [2014]).

A common practice to evaluate recommender systems is to partition an existing data into train and test sets, then learn a recommender system from the train set and measure the quality of recommendations made for the test set by comparing them with the original data. This method along with various evaluation metrics can be found implemented in many recommender systems software/libraries. However, most of the non-commercial libraries implement recommendation methods that deal with single-table data, and none with PRMs. But it would still be interesting to use such existing libraries for evaluating our PRM-based recommender system to avoid redundant effort of implementing some functionalities that already come in mature libraries for recommender systems. With this in mind, we propose a framework for evaluating PRM-based recommender systems using existing libraries for recommender systems. Though we have formalized our framework for PRM-based recommender systems, it is not limited to such systems, and should be applicable to recommender systems that use relational data. The main idea is to outsource the functionalities of partitioning the data and computing evaluation metrics to an external library, and let ourselves focus only on the implementation of learning from the train set and making recommendations for the test set.

Evaluating recommender systems dealing with relational data is not as straightforward as evaluating those dealing with single-table data. The main reason behind this is the need to have train sets in the form of relational data to be able to learn a recommender system. Most of the existing libraries for recommender systems provide the functionality of partitioning only single-table data for recommender system evaluation. Therefore, we also propose an algorithm for transforming such single-table partition to obtain a partition of the relational data.

This report is organised as follows. Section 2 will provide a brief overview of recommender systems, PRMs and PRM-based recommender systems. Our proposed framework will be discussed in detail in Section 3. It will continue with an illustration of transformation of a single-table partition to a relational data partition using our proposed algorithm in Section 4.

2 Technical background

In this section, we present a brief overview of technical aspects involved in the proposed framework. We will begin by introducing recommender systems, their evaluation methods, and some libraries that implement such systems in Section 2.1. We will then provide a brief overview to Probabilistic Relational Models (PRMs), and present some examples of PRM-based recommender systems in Section 2.2. Section 2.3 will introduce PILGRIM, a library for PRMs.

2.1 Recommender Systems

Recommender systems (RS) (Ricci et al. [2015]) aims at aiding users in making decision by suggesting them the items that they might find interesting. These days such systems have been widely adopted in many domains, notably the e-commerce. The growing interest on these systems has led to numerous studies resulting in a plethora of algorithms for recommendation. These algorithms are broadly classified into three categories: Collaborative Filtering (CF), content-based filtering and hybrid approaches.

Collaborative filtering (Schafer et al. [2007], Ekstrand et al. [2011]) is probably the most widely used method for recommendation. Assuming that similar users have similar tastes, CF methods recommend to the users the items that are liked by the users who have similar behaviors as the target user. *Content-based filtering* methods (Melville et al. [2002]) identify items with similar contents to those that are liked by the target user in the past. Hybrid approaches combine different techniques to benefit from their advantages. Burke [2007] and Jannach and Friedrich [2013] have presented different ways for hybridization.

Relational data

Most of these recommendation techniques use flat data which contains only the information about interactions between users and items, such as a *"user-item matrix"*. However, more recent studies advocate the use of additional information in the form of relational data in order to improve the recommendations. *Relational data* organize data as sets of objects of different types (users, items, transactions, context, etc.), where each object is defined by a set of attributes. Usually, user objects are described by users' demographic information, and item objects by various features/characteristics of the items. Such data are generally stored in the form of table, graph or document.

Evaluating recommender systems

Growing use of recommender systems in various domains has resulted in numerous recommendation algorithms. As choosing an appropriate recommendation method for a specific system requires comparison of several algorithms, the RS research community has been active in the study of effective evaluation techniques. Many researchers evaluate recommender systems using *off-line analy-*

sis, *live user experimental methods* (aka *on-line evaluation*) or a combination of these two approaches (Herlocker et al. [2004]).

In *on-line evaluation* methods, actual users interacts with a recommender system, and receive recommendations, which they may or may not accept. Recommendation quality can be then measured by asking users for their feedback or by monitoring the users' behavior during their interaction with the system (for example, by observing if they click on the recommended items or not). On-line evaluations are usually expensive and time-consuming.

Off-line evaluation is a widely used technique for evaluating recommender systems. This approach measures the accuracy of recommendations made by a recommender system without the involvement of actual users. It uses an existing data as ground truth, and expects the results of a RS to match the ground truth. To evaluate a recommender system, the dataset is split into a training set and a test set. Then, using the training set, the system tries to predict the ratings in the test set. The predicted ratings are then compared to the actual ratings in the test set. Several metrics can be computed to analyze the results.

Evaluation metrics

Among various metrics for evaluating recommender systems, accuracy metrics are the ones that are widely adopted. These metrics measure how close the predictions made by the recommender systems are to the ground truth. *Classification accuracy metrics* such as *Precision*, *Recall*, and *F-measure*, measure to what extent a RS is able to correctly classify items as interesting or not. *Mean Absolute Error* (MAE) is a widely used metric for measuring how well users' actual ratings match the ratings predicted by the recommender system. *Rank accuracy metrics* measure to what extent the ranking proposed by the recommender system differ from the actual ranking, assuming that the order of the items in the recommendation list is important. Though these evaluation metrics are still widely used, researchers in the RS community are exploring more practical evaluation metrics to cover different aspects of recommendations, such as *coverage*, *diversity*, *serendipity*, and *novelty*.

Software/libraries for recommendation

Implementation of recommender systems can be found in numerous commercial and non-commercial software. Some implementations are available as free and/or open-source software libraries. Among them, LensKit[1], MyMediaLite[2], mrec[3], Mahout recommendation engine[4], pyRecLab[5], rrecsys[6], Recommender-

[1] http://lenskit.org/
[2] http://www.mymedialite.net/
[3] https://github.com/Mendeley/mrec
[4] https://mahout.apache.org/users/recommender/recommender-documentation.html
[5] https://github.com/gasevi/pyreclab
[6] https://cran.r-project.org/web/packages/rrecsys/index.html

lab[7], Good Enough Recommendations (GER)[8], etc. are popular in the recommender system community, and are actively being used for research or in applications. Most open-source libraries are found to implement collaborative filtering, and deal only with user-item matrix. Very few libraries can deal with relational databases. For example, RecDB[9] is a PostgreSQL extension that allows developers to write SQL queries to generate recommendations for their applications. HapiGER[10] works not only with in-memory event stores but also with PostgreSQL, MySQL and RethinkDB to make recommendations based on collaborative filtering techniques. However, these libraries do not provide methods for evaluating recommender systems. Some mature libraries like LensKit, MyMediaLite, Recommenderlab, rrecsys, etc. provide functionalities for performing offline evaluation of recommender systems along with a range of mostly recommendation accuracy metrics.

2.2 Probabilistic Relational Models (PRM)

Statistical Relational Learning (SRL) aims at learning statistical models exploiting relational information present in the data. It is primarily dominated by methods that are based on Probabilistic Graphical Models (PGMs). PGMs use graph theory for expressing complex probabilistic dependencies between random variables (Wainwright and Jordan [2008]). Probabilistic Relational Models (PRMs) are one of the directed versions of PGMs for relational settings. A PRM models the uncertainty over the attributes of objects in the domain and the uncertainty over the relations between the objects (Friedman et al. [1999], Getoor [2002]). It basically defines a template for probabilistic dependencies in typed relational domains. *Inference* is performed on a PGM, called *Ground Bayesian Network* (GBN), which is obtained by applying the template over a particular set of objects and relations between them. Several algorithms for performing inference on *Bayesian networks* exists in the literature. When GBNs are small (i.e., when PRMs are simple and the dataset is small), *exact inference* can be performed. For complex PRMs and/or large datasets, *approximate inference* methods are required.

Constructing PRMs

A PRM can be constructed either with the help of domain experts or by learning it from data. In both of these cases, it requires to define two components: (i) parameters, (ii) the structure of the PRM. Standard statistical or Bayesian methods can be applied to learn parameters from data. However, very few algorithms for learning the structure of a PRM exist in the literature. The most cited is the Relational Greedy Search (RGS) algorithm (Friedman et al. [1999], Getoor [2002]), which learns the structure through greedy search.

[7] https://github.com/mhahsler/recommenderlab
[8] https://github.com/grahamjenson/ger
[9] https://github.com/DataSystemsLab/recdb-postgresql
[10] http://www.hapiger.com/

PRM extensions

A *regular PRM* is the simplest type of PRM that models only the uncertainty over the attributes of the objects. Several research works have extended this type of PRM to address more complex scenarios. For example, PRMs with Reference Uncertainty (PRMs-RU) and PRMs with Existence Uncertainty (PRMs-EU) are two important extensions of PRM, that deals with the uncertainty over the relationships between the objects (Getoor [2002]). Among these two extension, PRMs-EU are interesting for recommender systems, where the goal is to predict the existence of a relationship between objects. Huang et al. [2005] and Chulyadyo and Leray [2014] have applied this extension to propose their recommender systems. Other significant PRM extensions include PRM with Spatial Attributes (PRM-SA) (Chulyadyo and Leray [2015]), PRM with Clustering Uncertainty (PRM-CU) (Coutant et al. [2015]), PRM with relational uncertainty (Fersini et al. [2009]), and hybrid PRM (Närman et al. [2010]).

PRM-based recommender systems

Using PRMs in recommender systems has been a topic of research from the beginning of PRM formalism. Getoor and Sahami [1999] were the first ones to propose a PRM-based recommender system. They have been followed by several other researchers who have proposed different manners of integrating PRMs into the recommendation process. Huang et al. [2005] have proposed a PRM-based recommendation framework that combines concepts from PRM-EU and PRM structure learning. They address the recommendation task as the problem of predicting links between users and items, and show that PRMs-EU can be useful in such problems. Our earlier work (Chulyadyo and Leray [2014]) is also in the same direction, and proposed a personalized recommender system based on PRM-EU for recommending not frequently purchased items. Newton and Greiner [2004] have proposed *hierarchical PRM* (hPRM), which aims at addressing the issue of cyclicity, i.e. the scenario where users' ratings depend on the previous ratings of the users' neighbors, which usually appears in recommendation tasks. Ben Ishak et al. [2013] have also tried to find a solution to the same problem of cyclicity in their hybrid approach for recommendation based on PRMs. Gao et al. [2007] have proposed a hybrid method for recommendation that combines collaborative filtering and PRMs using a *user grade function* to adapt weights for these two recommendation techniques thereby making it suitable for both cold-start as well as non cold-start situations.

2.3 PILGRIM

PILGRIM is a software tool for working with probabilistic graphical models. It allows modeling, learning, and reasoning upon Bayesian networks, and probabilistic relational models. This library consists of three parts: the "non-relational" part, the relational part, and their applications. The "non-relational" part deals with static and dynamic Bayesian networks whereas the relational

part provides several functionalities for modeling PRMs and its extensions PRM-RU, PRM-CU and PRM-SA, learning such models from relational databases, and making inferences from those models. The application part is concerned with the use of these models in different practical applications, including recommender systems.

3 Proposed framework

We propose a framework for offline evaluation of recommender systems based on PRMs. The proposed framework defines a process that delegates the tasks of learning a recommender system and making recommendations from it to the system under study, and the task of evaluation data preparation and evaluation metrics calculation to an external library. The reason behind using an external library is to avoid redundant effort for implementing the functions that are already available in external libraries. In the proposition of this framework, we will mention PILGRIM library to refer to our PRM-based recommender system. However, readers should be able to replace this library with their own system when using the proposed framework.

The basic idea of our approach is to use a free/open-source library, such as LensKit, to partition a dataset into test and train datasets, then use PILGRIM to learn a recommender system from the train set, use the learned system to make recommendations for the test set, and finally calculate evaluation metrics using LensKit or a similar library. LensKit is one of the mature libraries for recommender systems, and implements several partition methods and evaluation metrics. Besides, it is well-documented too. That is why we chose LensKit to outsource the functionalities of partitioning of data and computation of evaluation metrics.

One problem with LensKit (and other free/open-source recommender system libraries) is that it deals only with single table data whereas PRM-based recommender systems need to deal with multiple tables at once. The partitions generated from LensKit contain only one table, and thus cannot be directly applied to such recommender systems. An additional step is needed to partition all remaining tables to create a complete multi-table dataset for evaluation. We also propose an algorithm for such transformation.

Figure 1 depicts the main steps of our proposed framework.

Step 1 Transform the relational data to the format required by LensKit, i.e. extract the "user-item matrix" from the relational data

This step can be done with the help of functions provided by DBMS. For example, the COPY command of PostgreSQL can be used to extract a "user-item" matrix from the relational database.

Step 2 Execute LensKit method(s) for partitioning the user-item matrix to obtain train and test datasets

The LensKit command `crossfold` partitions a user-item matrix into mul-

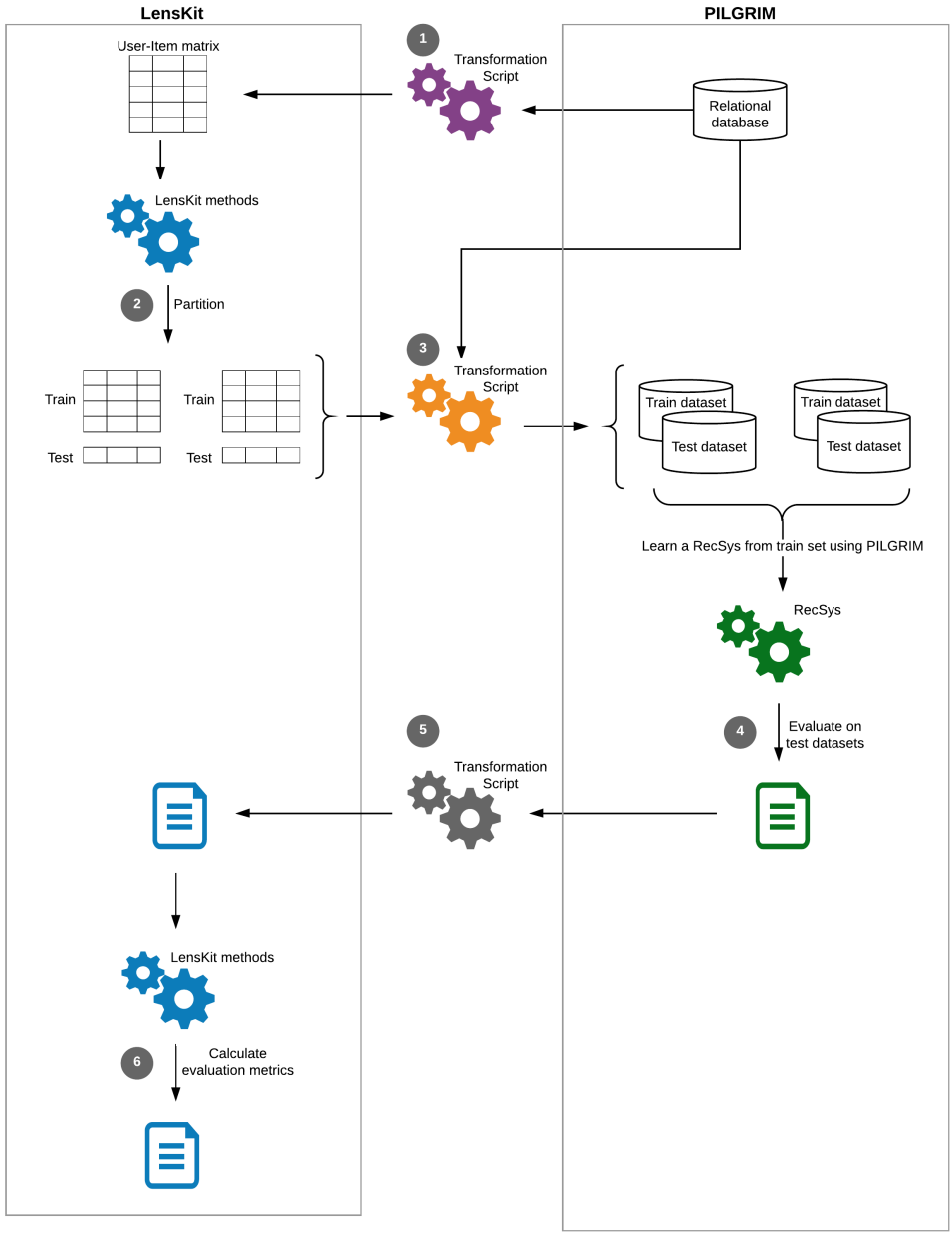


Figure 1: Proposed framework for offline evaluation of PRM-based recommender systems

tiple train and test datasets. It takes parameters, such as number of partitions, partition algorithm to use, etc., to create the partitions.

Step 3 Transform the train and test sets to relational databases with the help of the original relational database

Since LensKit deals with only user-item matrix, we need to transform the obtained partitions, which are subsets of the input user-item matrix, into relational databases to be able to learn PRM-based recommender systems. We have proposed an algorithm for this transformation. The idea is to create a partition of the original relational database by partitioning all tables with the help of the original relational database and the partition dataset obtained from LensKit. We explain our algorithm in detail in Section 3.1.

Step 4 Learn a recommender system from the train dataset and make recommendations on the test dataset

In our case, we use PILGRIM to learn a PRM-based recommender system from a train dataset. Then, for each object of the test dataset, recommendations are made. These recommendations need to compared to the actual values in the test set to evaluate the performance of the learned system.

Step 5 Transform the results obtained from Step 4 to the format required by LensKit to compute evaluation metrics

Several evaluation metrics are implemented in LensKit. The results obtained from Step 4 might not be compatible with the LensKit format. If so, we need to get the results in LensKit format to be able to compute evaluation metrics using LensKit.

Step 6 Execute LensKit methods to calculate evaluation metrics

Several classes (such as `TopNPopularityMetric`, `CoveragePredictMetric`, `RMSEPredictMetric`, `NDCGPredictMetric`, `HLUtilityPredictMetric`, etc.) are available in LensKit to calculate various evaluation metrics.

3.1 Transformation of single-table train and test datasets to multi-table relational databases

The LensKit command `crossfold` generates flat datasets whereas PILGRIM requires relational databases. Besides, the output of this command contains only transactions between users and items. For PILGRIM to be able to use train and test sets obtained from LensKit, they need to be transformed into relational databases such that 'transaction' table along with all other tables are partitioned conforming to the train and test sets returned by `crossfold` command. We propose an algorithm for this transformation. For an input dataset, the `crossfold` command generates 2*n files, where n is the number of the partitions. Each of these files contains either a train or a test dataset in flat

format. As all of these output files have the same format, we can process them in the same way. Our algorithm takes one of these files, and the original database as the input, and then extracts a partition of the original dataset corresponding to the input file.

The basic idea of our approach is to start by populating the table that corresponds to the interaction between users and items (i.e. the table corresponding to the output files of the `crossfold` command) using the original relational database and the output file of the `crossfold`, then perform a traversal through the underlying relational schema DAG[11] to find the sequence of tables to be filled in. Our algorithm is listed as Algorithm 1.

The algorithm works as follows. First of all, only primary key columns and non-foreign key columns of 'transaction' table are populated with the data from the input file. Its foreign key columns will be populated as a graph traversal is performed on the relational schema DAG. Starting from this table, a depth first search (DFS) is performed on the relational schema DAG, keeping tracking of all encountered nodes and edges during the traversal. For each edge being visited, it populates from the original relational database the child table with only those data that are related to the data in the parent table. At this point, primary key columns and non-foreign key columns of the child table are populated, and then the parent table's foreign key column that is associated with the child table is updated. By the end of the DFS, all tables appearing in the DFS will have been completely filled with the objects related to the data in the input file. The next step is to fill the remaining tables. This involves picking unvisited leaf nodes and filling the tables backwards. In this algorithm, a visited node/table indicates that at least primary key columns of the table are already populated. Thus, to be able to fill a table, at least one of its parents or children should have been visited. In the case of an unvisited leaf node, it is mandatory that its parents be filled first. Thus, for each unvisited leaf node, an undirected path from this node to the nearest visited node is searched. Then, tables in this path is populated in the reverse order, i.e. starting from the parent or child of the visited node to the unvisited leaf node. In the process, we observe the following principle of populating tables (which is also observed during the above DFS) : if the parent table is already visited, populate the primary key and non-foreign key columns of the child table and update the associated foreign key column of the parent table, otherwise if the child table is already visited, populate the primary key, non-foreign key and associated foreign key columns of the parent table. After working with all leaf nodes, we will obtain a complete relational database that is a partition of the original relational database and corresponds to the partition of the input file.

[11] A relational schema DAG is a directed graph where each table in the schema is represented by a node, and each foreign key is represented by an edge directed from the table containing the foreign key to the table that is referred to by the foreign key.

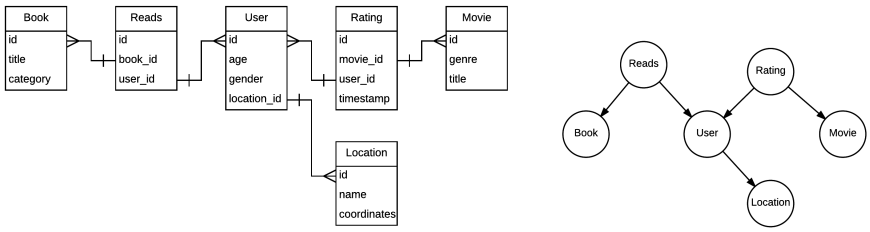
Algorithm 1 Transform a crossfold output file to a relational database

Require: Starting node/table N, original database R, `crossfold` output file O, Schema DAG $G = (V, E)$ associated to R
Ensure: Database R' that corresponds to O and is a subset of R

PN = {} ▷ Set of nodes/tables whose primary keys have already been populated
VE = {} ▷ Set of edges which have already been visited
Create or copy the schema of R to R'
for each object of N in O **do**
    Populate primary keys and non-foreign keys of table N of R' from table N of R
end for
PN = PN $\cup$ $\{N\}$
DE = Sequence of edges of G using DFS starting from N
for $L = A \to B \in$ DE **do**
    Using A of R, populate primary keys and non-foreign keys of objects of B in R' related to objects of A in R'
    Populate corresponding foreign key of objects of A in R'
    VE = VE $\cup$ $\{L\}$
    PN = PN $\cup$ $\{B\}$
end for
LN = Leaf nodes of G
for $L \in$ LN **do**
    EL = Sequence of edges on the undirected path from the nearest node in PN to L
    for $e = X - Y \in$ EL **do**
        if $e \notin$ VE **then**
            if $X \in$ PN and $e = X \leftarrow Y$ **then**
                Using X of R, populate all attributes of objects of Y in R' related to objects of X in R'
                VE = VE $\cup$ $\{Y\}$
                PN = PN $\cup$ $\{e\}$
            end if
            if $X \in$ PN and $e = X \to Y$ **then**
                Using X of R, populate all attributes of objects of Y in R' related to objects of X in R'
                Populate corresponding foreign key of objects of X in R'
                VE = VE $\cup$ $\{Y\}$
                PN = PN $\cup$ $\{e\}$
            end if
        end if
    end for
end for

4 A toy example

In this section, we illustrate our algorithm of transforming flat data to relational databases (Algorithm 1) with an example. We consider the relational schema of Figure 2, with 6 tables which are related to each other as shown in Figure 2a. This schema can also be represented as a DAG, where each node represents a table and each edge represent a foreign key and is directed to the class that contains the corresponding primary key, as shown in Figure 2b. For this illustration, we consider the dataset shown in Figure 3. The command 'crossfold' generates files containing either train or test partitions of the table Rating. Our algorithm takes one of these files and the original dataset to obtain a partition of the relational database compatible with this file. In other words, if we rep-



(a) (b)

Figure 2: (a) An example of a relational schema, and (b) the corresponding DAG

Book

id	title	category
1	t1	c1
2	t2	c1
3	t3	c2
4	t4	c2
5	t5	c1

Reads

id	b_id	u_id
1	1	1
2	1	2
3	2	2
4	3	2
5	2	3
6	2	4
7	3	4
8	4	4
9	5	4

User

id	age	gender	loc_id
1	a1	F	1
2	a2	F	1
3	a1	M	2
4	a2	F	2

Location

id	name	coord
1	Nantes	cc
2	Paris	xx

Rating

id	u_id	m_id	r
1	1	1	5
2	1	2	3
3	2	3	4
4	3	3	1
5	2	4	2
6	3	4	2
7	4	5	5

Movie

id	genre	title
1	g1	t1
2	g2	t2
3	g2	t3
4	g1	t4
5	g3	t5

Figure 3: Example of a relational dataset. Among the 6 tables, the LensKit command 'crossfold' uses and partitions only the Rating table. Based on the selected paramters, it returns multiple files, each of which contains either train or test dataset. In this example, the red color indicates that the line is in train set, and the green in test set.

resent our database as a DAG of Figure 4, our algorithm generates partitions indicated by the two colors. The red objects would belong to the train dataset, the green ones belong to the test set, and the objects with both colors belong to both datasets.

In the following, we execute Algorithm 1 and show the results step by step. For simplicity, we consider the test set which contains only Rating with ID 6 and 7 in this illustration.

Steps 1 - 3

In these steps a relational schema identical to the original database is created. All the tables are empty at this point. In this illustration, the column 'id' with yellow color will indicate that the table is in the set PN, and dashed lines between tables will indicate that they are not yet visited, i.e. they are not in the set VE, which contains only the solid lines.

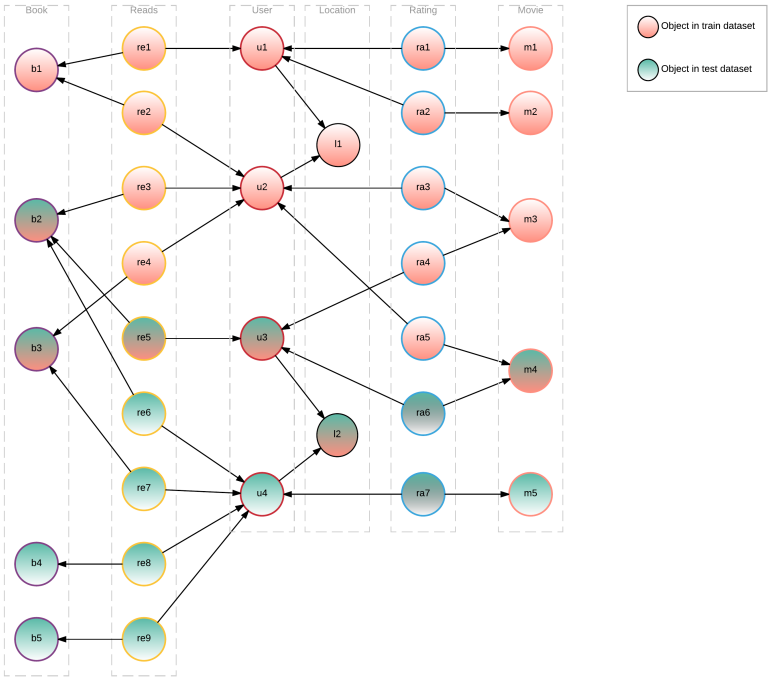


Figure 4: DAG corresponding to the dataset of Figure 3, where the green nodes should be in the test set, the red ones in the train set, and the nodes with both colors in both datasets.

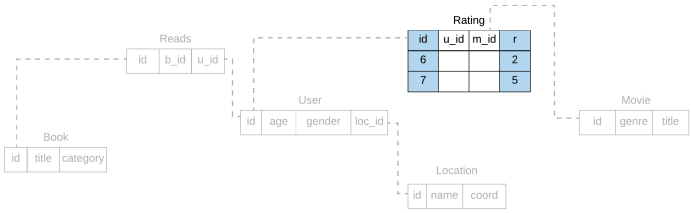


Figure 5: Steps 4 - 7 populate non-foreign keys of Rating table

Steps 4 - 7

These steps populate all columns except the foreign keys of the table Rating with the data from the input file. After this step, our database will be as shown in Figure 5.

Steps 8 - 17

These steps perform Depth First Search (DFS) over the DAG of the relational schema (Figure 2b) starting from the Rating table. In this example, we consider that the DFS would visit the edges in the following sequence:

Rating → User
User → Location
Rating → Movie

For each of these edges, we perform Steps 10-13, where all columns except foreign keys of the child table are first populated from the original database, and then the corresponding foreign key column of the parent table is populated. Here, only those objects of the child table which are associated to those of the parent table in the original database are considered.

We begin the iteration (of Step 9) with the edge Rating → User. First, using the Rating data obtained from Steps 4-6, all columns except foreign keys of the table User are populated from the original database. Since rating objects with id = 6 and 7 are associated with users with id = 3 and 4 in the original database, only these two users are taken into the User table. Then, the foreign key of Rating objects associated with User table is populated. The line between Rating and User is then changed to a solid line, indicating that this edge has been visited. After this iteration, we will obtain a partially populated User table containing users with id 3 and 4 as shown in Figure 6.

Steps 9-14 are repeated for the edges User → Location (Figure 7) and Rating → Movie (Figure 8). After the execution of DFS starting from Rating table, all table included in this DFS, i.e. Rating, Movie, User, and Location, will be completely filled.

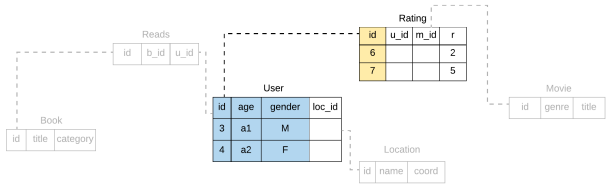
Step 15

Once the DFS is finished, we now search for the leaf nodes of the DAG of the relational schema to continue the population of the remaining tables. The leaf nodes in our example are Book, Movie and Location (Figure 9). As Location, and Movie are already populated (as indicated by the PN set), the table Book is chosen for the next steps.

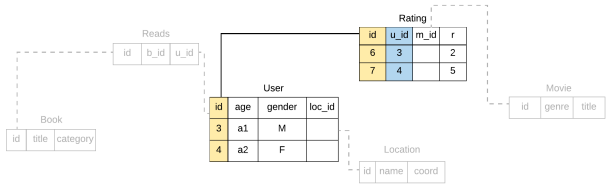
Steps 16 - 33

Book is related to Reads but neither of these two tables are populated yet. Thus, to continue the population of the database, we need to follow the edges to find a table that is already in the set PN (i.e. at least ID column is populated) and is accessible from Book. In our case, the closest table accessible from Book is User, and the sequence for this traversal is:

Book ← Reads, Reads → User

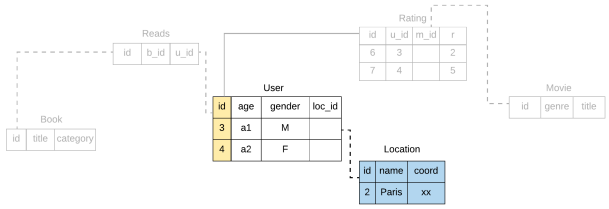


(a)

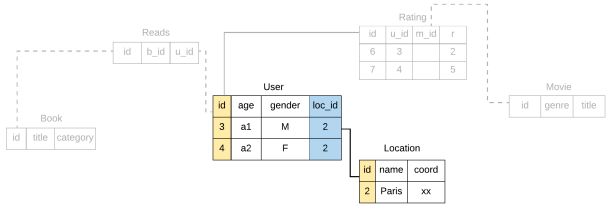


(b)

Figure 6: Steps 9-14 for the edge Rating → User will populate the User table with users 3 and 4 because only these two users are related to ratings 6 and 7 in the original database. `user_id` (`u_id` in short) of Rating table will then be populated with the corresponding user IDs.

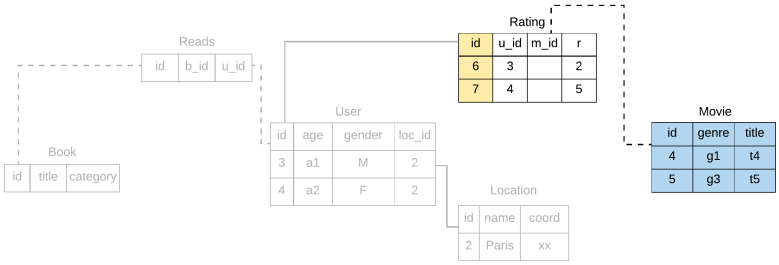


(a)

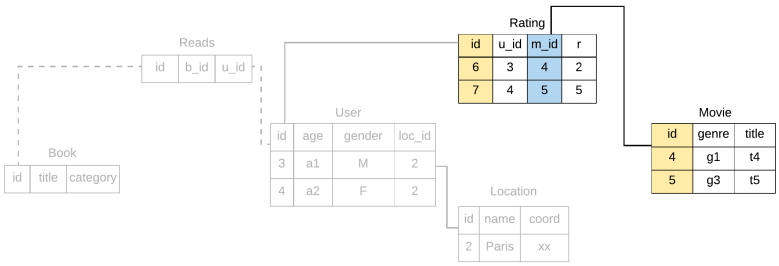


(b)

Figure 7: Steps 9-14 for the edge User → Location result in the completely populated tables Location and User with the related objects from the original database. Figures (a) and (b) correspond to Steps 10, and 11 respectively.



(a)



(b)

Figure 8: Steps 9-14 for the edge Rating → Movie also result in the completely populated tables Rating and Movie. After the DFS, all visited tables will have been completely populated with the relevant data.

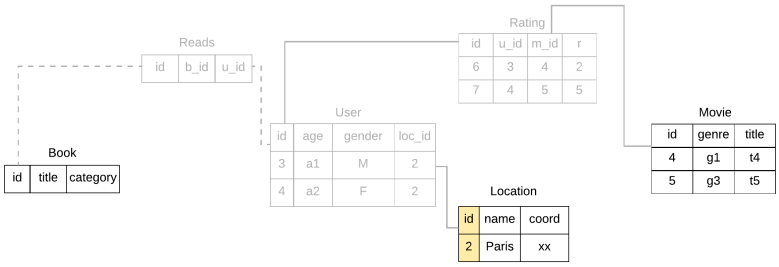


Figure 9: Leaf nodes of the DAG of Figure 2b

For each of the edges in the reverse order, we fill in the table according to the direction of the edge using the already populated table (i.e. the one already in PN). We populate all the columns except the foreign keys of the table that is not in PN and then populate the corresponding foreign key of either the same table or the other table according to the direction of the edge.

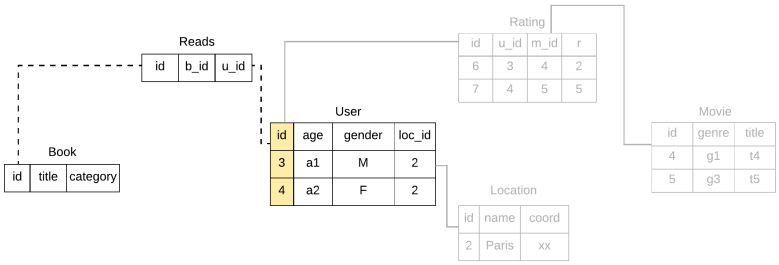


Figure 10: Since Location, and Movie are already visited, we choose Book, but we need to find an undirected path from Book to the nearest visited node. In this case, it is Book - Reads - User

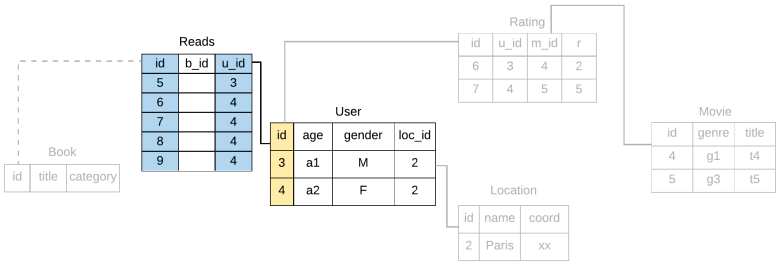
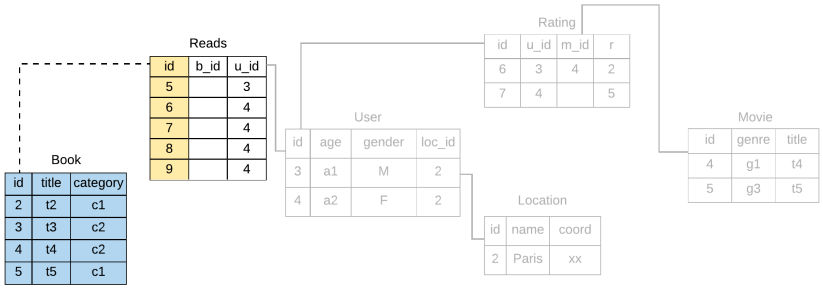


Figure 11: We now fill the tables in the reverse order of Book - Reads - User path. So, we start with the edge Reads → User, for which Steps 20-24 will be executed resulting in this situation of the database.

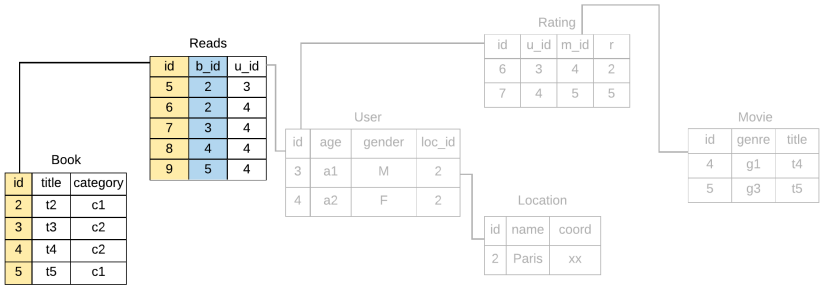
In our example, the first edge for this step is Reads → User. Since User is already populated, we can populate the primary key as well as the foreign key of Reads using the User objects. By the end of this iteration, the five Reads objects related to the two User objects will be in the Reads table. The result of this iteration will be the database shown in Figure 11.

Next, we repeat these steps for the edge Book ← Reads to obtain the result as shown in Figure 12b.

After the execution of Steps 16-33, we will obtain a partition of the original database, as shown in Figure 13, which includes the green nodes of Figure 4.



(a)



(b)

Figure 12: Database after the execution of Steps 25-30 for the edge Book ← Reads. (a) corresponds to Step 26, and (b) corresponds to Step 27

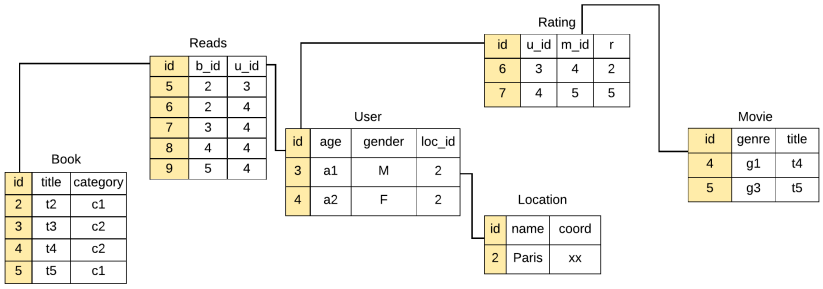


Figure 13: Algorithm 1 results in this database, which is a subset of the original database of Figure 3, and contains only green nodes of the DAG of Figure 4.

5 Conclusion

Several free/open-source libraries are available for recommender systems and their evaluation (mostly offline). However, most of them implement collaborative filtering methods, and work only with user-item matrix. When it comes to evaluating recommender systems based on relational databases, it becomes difficult to access functionalities provided by such libraries, so the system would have to re-implement the evaluation techniques to relational settings. This report addresses the problem of using free/open-source libraries of recommender systems for evaluating recommender systems based on Probabilistic Relational Models (PRMs) or, in general, those based on relational data. We aim at avoiding the redundant effort of re-implementing evaluation techniques for relational settings by using existing implementations of recommender system evaluation. We propose a general framework for offline evaluation of PRM-based recommender systems. It uses an external library for partitioning evaluation dataset, and computing evaluation metrics, while the system under study learns a model from the train dataset and makes recommendations for the test dataset. Since the external library inputs and outputs only single-table data, a transformation from single-table data to multi-table relational data and vice versa is required. We also propose an algorithm for this transformation, and provide a step-by-step illustration of this algorithm. Though our framework is originally proposed for evaluating recommender systems based on PRMs, it is still applicable in any recommender systems that deal with relational data.

Acknowledgement

The authors would like to acknowledge Mr. Yuan Xie for his technical skills and helpful discussions during the project.

References

M. Ben Ishak, N. Ben Amor, and P. Leray. A relational bayesian network-based recommender system architecture. In *Proceedings of the 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO 2013)*, 2013.

R. Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

R. Chulyadyo and P. Leray. A personalized recommender system from probabilistic relational model and users' preferences. In *18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems*, KES 2014, pages 1063–1072, Gdynia, Poland, 2014.

R. Chulyadyo and P. Leray. Integrating spatial information into probabilistic relational models. In *IEEE International Conference on Data Science and Advanced Analytics*, DSAA'15, pages 1–8, Paris, France, Oct 2015.

A. Coutant, P. Leray, and H. Le Capitaine. Probabilistic relational models with clustering uncertainty. In *International Joint Conference on Neural Networks*, IJCNN, pages 1–8. IEEE, 2015.

M. D. Ekstrand, J. T. Riedl, and J. A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

E. Fersini, E. Messina, and F. Archetti. Probabilistic relational models with relational uncertainty: An early study in web page classification. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, volume 3 of *WI-IAT'09*, pages 139–142. IET, 2009.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309. Lawrence Erlbaum Associates Ltd, 1999.

Y. Gao, H. Qi, J. Liu, and D. Liu. A recommendation algorithm combining user grade-based collaborative filtering and probabilistic relational models. In *Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery - Volume 01*, FSKD '07, pages 67–71, Washington, DC, USA, 2007. IEEE Computer Society.

L. Getoor. *Learning statistical models from relational data*. PhD thesis, Stanford University, 2002.

L. Getoor and M. Sahami. Using probabilistic relational models for collaborative filtering. In *Workshop on Web Usage Analysis and User Profiling (WEBKDD'99)*, pages 1–6, 1999.

J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

Z. Huang, D. Zeng, and H. Chen. A unified recommendation framework based on probabilistic relational models, November 2005.

D. Jannach and G. Friedrich. Tutorial: recommender systems. International Joint Conference on Artificial Intelligence, 2013. URL http://ijcai13.org/files/tutorial_slides/td3.pdf.

P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 187–192, Edmonton, Alberta, Canada, 2002. AAAI Press / The MIT Press.

P. Närman, M. Buschle, J. König, and P. Johnson. Hybrid probabilistic relational models for system quality analysis. In *14th IEEE International on Enterprise Distributed Object Computing Conference*, EDOC, pages 57–66. IEEE, 2010.

J. Newton and R. Greiner. Hierarchical probabilistic relational models for collaborative filtering. In *Workshop on Statistical Relational Learning, 21st International Conference on Machine Learning*, 2004.

F. Ricci, L. Rokach, and B. Shapira. Recommender systems: introduction and challenges. In *Recommender Systems Handbook*, chapter 1, pages 1–34. Springer, 2015.

J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1 (1-2):1–305, 2008.