http://webdam.inria.fr

Web Data Management

Introduction to recommendation systems

Serge Abiteboul
INRIA Saclay & ENS Cachan

Ioana Manolescu
INRIA Saclay & Paris-Sud University

Philippe Rigaux
CNAM Paris & INRIA Saclay

Marie-Christine Rousset
Grenoble University

Pierre Senellart
Télécom ParisTech

`http://webdam.inria.fr/Jorge/`

Contents

This chapter proposes an introduction to recommendation techniques and suggests some exercises and projects. We do not present a recommendation system in particular but rather focus on the general methodology. As an illustrative example, we will use the MovieLens data set to construct movie recommendations.

The chapter successively introduces recommendation, user-based collaborative filtering and item-based collaborative filtering. It discusses different methods parameterizations and evaluates their result with respect to the quality of the data set. We show how to generate recommendations using SQL queries on the MovieLens data set. Finally, we suggest some projects for students who want to investigate further the realm of recommendation systems.

1 Introduction to recommendation systems

Given a set of ratings of items by a set of users, a recommendation system produces a list of items for a particular user, possibly in a given context. Such systems are widely used in Web applications. For example, content sites like Yahoo! Movies (movies), Zagat (restaurants), LibraryThing (books), Pandora (music), StumbleUpon (website) suggest a list of items of interest by predicting the ratings of their users. E-commerce sites such as Amazon (books) or Netflix (movies) use recommendations to suggest new products to their users and construct bundle sales. Usually, they exploit the recent browsing history as a limited context. Finally, advertisement companies need to find a list of advertisements targeted for their users. Some of them, like Google AdSense, rely more on the context (e.g., keywords) than on an estimation of the user's taste based on her/his recent browsing history. Nevertheless, techniques close to recommendation methodologies are successfully used, for example, by DoubleClick or Facebook ads.

One usually distinguishes two kinds of tasks on data: information retrieval and information filtering. *Information retrieval* is the problem of answering dynamic queries on static content. Typical examples are answering keyword queries on the Web or SQL queries on a

database. The general method relies on data modeling, providing structure and semantics to the data, that is then organized using indexes. *Information filtering* is the problem of answering static queries on dynamic content. A typical example is the monitoring of Web server logs. The general method is to model the queries, which are then organized as filters. Under this general perspective, recommendation stands between information retrieval and information filtering: data (the set of ratings) varies slowly at the scale of a user but quickly at the scale of the system; queries (a user and possibly some context) depend on a few parameters, each having a wide domain.

Specifically, a recommendation system may either produce top-k ranking (list of "best" items) or prediction of ratings. The focus of the result may be generic (everyone receives the same recommendations), demographic (everyone in the same category receives the same recommendations) or personal. In the present chapter, we are mostly interested in personal recommendation. The context may rely on the user's current activity or on her/his long-term interests

The information that serves as a basis to recommendation systems consists of the following components:

1. the users' description (e.g., sex, age, localization, profession of the user);
2. the items' description (e.g., genre, author, date, price of the item);
3. and the ratings matrix, giving the rating of each item by each user.

The ratings matrix is incomplete, being fed only by either acquiring data from the user (e.g., an item is bought, or a level of interest is explicitly collected), or by monitoring her/his activity (an item is visited, which gives some hint on the user's interests). Recommendation is indeed the process of filling empty cells of the matrix with *predicted ratings* derived from the other sources of information, including known ratings.

2 Pre-requisites

This chapter uses SQL: we assume the reader familiar with the language. You will need access to a relational database, for example by installing MySQL on your computer: see `http://www.mysql.com`. Here is a very brief introduction to MySQL commands (refer to the Web for information on any other SQL database systems). Assuming that you have an account on the MySQL server, the connection is established with:

```
mysql -h [servername] -P [port] -u [login] -p
```

The utility asks for your passwords and gives you access to the command-line interpreter. You may directly type SQL commands, or execute command(s) stored in a file `myCom.sql`:

```
mysql> source myCom.sql;
```

We will play with the MovieLens (`http://www.movielens.org`) data set to generate recommendations of movies. The data set must first be imported in your database. Create the following tables and indexes (the SQL scripts can be found on the book's site):

```
# Tables creation
create table ratingsdata (
   userid int,
   itemid int,
   rating int,
   timestamp int,
   primary key (userid, itemid));

create table items (
   itemid int primary key,
   title text,
   date text,
   videodate text,
   imdb text,
   unknown boolean,
   action boolean,
   adventure boolean,
   animation boolean,
   childrens boolean,
   comedy boolean,
   crime boolean,
   documentary boolean,
   drama boolean,
   fantasy boolean,
   noir boolean,
   horror boolean,
   musical boolean,
   mystery boolean,
   romance boolean,
   scifi boolean,
   thriller boolean,
   war boolean,
   western boolean);

create table users (
   userid int primary key,
   age int,
   gender char,
   occupation text,
   zip int);

# Indexes creation
create index usersdata_index on ratingsdata (userid);
create index itemsdata_index on ratingsdata (itemid);
```

You can get the MovieLens 100K Ratings data set from `http://www.grouplens.org/node/73`. The files are respectively named `u.data`, `u.item`, and `u.user`. They are loaded in the database as follows

```
load data infile '[path to u.data]' into table ratingsdata;
load data infile '[path to u.item]' into table items fields
```

```
   terminated by '|';
load data infile '[path to u.user]' into table users fields
   terminated by '|';
```

Table `ratingsdata` table now contains the list of ratings. Most of the computation presented further rely on its content. Table `items` and `users` contain respectively the list of movies and the list of users.

3 Data analysis

The quality of a given recommendation method highly depends on the quality of the input. It can be characterized by the *support* (number of users and items, and distribution of the number of ratings by users and by items) and by the *rating quality* (distribution of the ratings by user and by movies). Let us consider the support first, which can be determined by the following SQL commands:

- number of users, movies and ratings;

```
select count(distinct userid) as nbusers,
   count(distinct itemid) as nbitems, count(*) as nbratings
from ratingsdata;
```

- distribution of the number of ratings by user (histogram rounded to a precision of 10 ratings);

```
select count(userid) as nbusers, nbratings
from (select round(count(itemid)/10,0)*10 as nbratings, userid
      from ratingsdata
      group by userid
     ) as nbratingsbyusers
group by nbratings
order by nbratings desc;
```

- distribution of the number of ratings by movies (histogram rounded to 10 ratings).

```
select count(itemid) as nbitems, nbratings
from (select round(count(userid)/10,0)*10 as nbratings, itemid
      from ratingsdata
      group by itemid
     ) as nbratingsbyitems
group by nbratings
order by nbratings desc;
```

1. Run the queries and examine the result. Note first that there is no user with less than 20 ratings, since such users have already been filtered out by MoviesLens. However,

one can find some movies with very few ratings. Recommending an item with a small support yields unreliable results. The problem is known as "cold-start" in the area of recommendation system, and is difficult to solve: we will not elaborate further on this aspect.

2. Can you determine the law followed by these distributions? This law is frequently observed in practice, and means that a few users are very productive and a few items are very famous, while the huge majority of items are hardly rated by any user. A good recommendation method should avoid giving more importance to items or users based on their number of ratings, since quantity does not always implies quality.

We now examine the quality of the ratings with the following SQL queries:

- average rating

```
select avg(rating) as avgrating
from ratingsdata;
```

- ratings distribution

```
select count(*) as nbratings, rating
from ratingsdata
group by rating
order by rating desc;
```

- distribution of the average ratings by users (histogram rounded to 0.1)

```
select count(userid) as nbusers, avgrating
from (select round(avg(rating),1) as avgrating, userid
      from ratingsdata
      group by userid
     ) as avgratingbyusers
group by avgrating
order by avgrating desc;
```

- distribution of the average ratings by movies (histogram rounded to 0.1)

```
select count(itemid) as nbitems, avgrating
from (select round(avg(rating),1) as avgrating, itemid
      from ratingsdata
      group by itemid
     ) as avgratingbyitems
group by avgrating
order by avgrating desc;
```

Run the queries and examine the result. Can you determine the distribution law? What happens regarding the distribution of the average ratings by movies, compared to the natural expectation? Try to figure out what would be the normal curve for such an application, and explain the "picks" associated to each rounded rating. Also note the curve behavior for extreme values, and provide an explanation. Finally note that the distribution of ratings is not centered. Why?

As for most data analysis tasks, raw data has to be cleaned up during a preprocessing step. We will limit ourselves to the centering of the ratings distribution. This normalization makes easier the comparison of the users' behavior. A more involved normalization would, among others, also correct the standard deviation. This is left as an exercise. The centering is obtained by the following query:

```
create table ratings (
   userid int,
   itemid int,
   rating int,
   timestamp int,
   primary key (userid, itemid));
create index usersratings_index on ratings (userid);
create index itemsratings_index on ratings (itemid);
insert into ratings (userid,itemid,rating,timestamp)
   (select ratingsdata.userid, ratingsdata.itemid,
        ratingsdata.rating-avgratingbyusers.avgrating,
        ratingsdata.timestamp
    from ratingsdata,
       (select userid, avg(rating)
        from ratingsdata
        group by userid
       ) as avgratingbyusers
    where ratingsdata.userid=avgratingbyusers.userid
   );
```

4 Generating some recommendations

4.1 Global recommendation

Global recommendation roughly retrieves the movies with the best average rating. The query is straightforward:

```
select title, avgrating, nbratings
from items,
   (select round(avg(rating),1) as avgrating,
       count(userid) as nbratings, itemid
    from ratings
    group by itemid
    order by avgrating desc
    limit 10
   ) as avgratingbyitems
```

```
where items.itemid = avgratingbyitems.itemid
order by avgrating desc;
```

If you carefully look at the result, you should observe that items with the best average ratings are those with a very small support (only a few ratings are known). This is a classic problem in statistics: an estimation of the average cannot be accurate if the support is too small. Problems related to the low quality of the support are very common in recommendation. In practice, a safe rule is to base any estimation on at least ten measurements. How can you correct the query to obtain a better result? Write and run the corrected query.

The next query retrieves the 40 movies with the largest number of ratings.

```
select title, items.itemid, avgrating, nbratings
from items,
   (select round(avg(rating),1) as avgrating,
       count(userid) as nbratings, itemid
    from ratings
    group by itemid
    order by nbratings desc
    limit 40
   ) as avgratingbyitems
where items.itemid = avgratingbyitems.itemid
order by nbratings desc;
```

Pick 20 of those movies (if possible those you know) and give them a rating using the command:

```
create table me (
   itemid int primary key,
   rating int);
insert into me values (id1,rating1), (id2,rating2), ... (id20,rating20);
```

You may want to check your updates with:

```
select title, me.itemid, rating
from me, items
where me.itemid=items.itemid;
```

We will use this table to compute some movie recommendations for you. Keep in mind that in a real recommendation system, one has to find recommendation for every user, so scaling is a real issue.

4.2 User-based collaborative filtering

The collaborative filtering class of methods focuses on the ratings matrix and ignores the users or items description. It usually proceeds in two steps: first the correlation step determines a similarity between users (for the user-based approach) or between items (item-based), then the aggregation step predicts new rating from this similarity information.

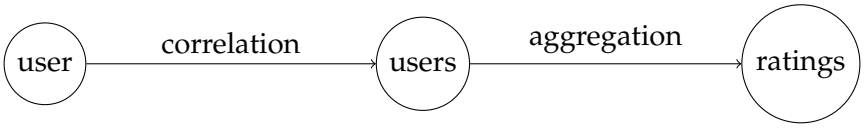

Figure 1: User-based collaborative filtering

In the case of user-based collaborative filtering (Figure 1), the correlation between a pair of users is computed by comparing their ratings. For simplicity (and efficiency), we only compute the correlation between you and all the other users. Then the ratings of these users for a given item are aggregated to predict the rating of the initial user for this item.

4.2.1 Correlation

There exist several possible measures of correlations. Let U_i be the vector of ratings of user u_i (seen as a line), then

- the scalar product similarity is:

$$sim(u_j, u_l) = U_j {}^t U_l$$

- the cosine similarity is:

$$sim(u_j, u_l) = \frac{U_j {}^t U_l}{\|U_j\| \|U_l\|}$$

The cosine correlation is obtained by the following query:

```
select distances.userid as userid, dist/(sqrt(my.norm)*sqrt(users.norm))
   as score
from (select userid, sum((me.rating)*(ratings.rating)) as dist
      from ratings, me
      where me.itemid = ratings.itemid
      group by userid
     ) as distances,
     (select userid, sum((rating)*(rating)) as norm
      from ratings
      group by userid
     ) as users,
     (select sum((rating)*(rating)) as norm
      from me
     ) as my
where users.userid = distances.userid
order by score desc
limit 30;
```

You can compare the ratings of user u_i to yours with the following query:

```
select me.itemid as itemid, me.rating as myrating,
```

```
   ratings.rating as herrating
from ratings, me
where userid=u_i and ratings.itemid=me.itemid
order by me.itemid;
```

You should observe that the estimation of the correlation is not accurate for pairs of users with a small support (i.e., users who rated only a few common items). How can you modify the correlation formula to take the support into account? You should in particular try the other formula suggested above. This should lead to the conclusion that there is a trade-off regarding the support: giving too much weight to the support may bias the result toward popular items, whereas simply ignoring it leads to a bad estimation quality.

We used the normalized table in the SQL commands. What could happen if we had used the initial, non-normalized data?

We keep the correlated users whose behavior is close to yours, into the `sim` table, using the following command:

```
create table sim (
   userid int primary key,
   score double);
insert into sim (userid,score)
   (select ...)
```

4.2.2 Recommendation

Let $\hat{r}(u_i, i_k)$ be the rating prediction of user u_i and item i_k and let $S_t(u_i)$ be the user highly correlated with u_i (the users that you put in the `sim` table). The following formula represent some possible ways of computing aggregated values:

- Means on the best users (the rating of a user for an item is considered to be equal to 0 if it does not exist in the rating matrix).

$$\hat{r}(u_j, i_k) = \frac{1}{|S_t(u_j)|} \sum_{u_l \in S_t(u_j)} r(u_l, i_k)$$

- Weighted average on the best users.

$$\hat{r}(u_j, i_k) = \frac{\sum_{u_l \in S_t(u_j)} sim(u_j, u_l) r(u_l, i_k)}{\sum_{u_l \in S_t(u_j)} sim(u_j, u_l)}$$

The means aggregation is obtained by:

```
select title, items.itemid, score, nbratings
from items,
   (select itemid, sum(ratings.rating)/simsize.size as score,
       count(sim.userid) as nbratings
    from sim, ratings,
       (select  count(*) as size from sim) as simsize
```

```
    where sim.userid= ratings.userid
    group by itemid
    order by score desc
    limit 10
   ) as itemscores
where items.itemid = itemscores.itemid
order by score desc;
```

You probably want to remove the movies that you already know by adding the following filter to the `where` clause:

```
and itemid not in (select itemid from me)
```

You may also want to see the movies you probably dislike, replacing `desc` by `asc` in the previous command.

1. We used the normalized table. What may happen if you use the raw data?
2. You may have already observed that we kept only the 30 closest users. Try a different number as intermediate seeds (clean first the `sim` table with `delete from sim;`). Usually, the choice of $S_t(u_i)$ is very sensitive. If you are too selective, you get results with very small support (few aggregated ratings by items), and a bad estimation. If you are not selective enough, you get results very close to the global recommendation (the majority wins), and thus a bad precision. This is another illustration of the concerns related to the support: there is a trade-off between sparsity (bad estimation) and noise (bad precision).
3. To soften the previous problem, one may try to estimate the quality of the correlation. For example, try the weighted average (or even a quadratic weight). Try also to use a threshold on the value of the correlation

4.3 Item-based collaborative filtering

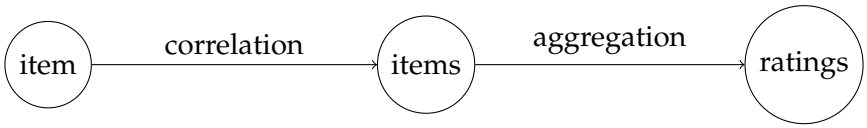

Figure 2: Item-based collaborative filtering

For item-based collaborative filtering (Figure 2), we compute the correlation between any pairs of items by comparing their ratings. We then aggregate the ratings of a user for these items to predict the rating of this user for the initial item. To avoid too much computation time, you may only compute the correlation between all items and yours. Let I_k be the vector of ratings of item i_k (seen as a column). You may use:

$$sim(i_l, i_k) = {}^tI_l \, I_k$$

$$\hat{r}(u_j, i_k) = \frac{1}{|S_t(i_k)|} \sum_{i_l \in S_t(i_k)} r(u_j, i_l)$$

1. How can you rewrite the previous queries to do item-based collaborative filtering?
2. What is usually the benefit of using item-based collaborative filtering instead of user-based collaborative filtering, from the support point of view? In particular, what changes if some attacker in the system attempts to improve the recommendation of some items by adding new ratings?

5 Projects

The following projects outline some suggestions to extend the basic recommendation scheme presented above.

5.1 Scaling

So far, we limited the computation to recommendations for a single user. In general, recommendation systems attempt to provide recommendations to every of their users. Several methods can be envisaged to achieve scalability:

1. distribution,
2. clustering methods to group similar users and similar items,
3. or by reducing the dimension on the ratings matrix.

Any of these methods can be used as a starting point for a project aiming at scalable computation of the recommendations. Distribution is a suitable objective if you wish to investigate in the context of a practical project some of the main techniques described in the book. You could for instance design and experiment the computation of the correlation and aggregation indicators with the MAPREDUCE paradigm, taking the opportunity of implementing your functions in one of the systems that we present in other *Putting into Practice* chapters (e.g., HADOOP or COUCHDB).

5.2 The probabilistic way

Some recommendation methods are fully based on a probabilistic model. In general, they consist in choosing a probabilistic model of generation (e.g., using Markov Chains), followed by an estimation of the model parameters (e.g., using Expectation Maximization). The project can be conducted by finding academic references to model-based recommendation. You should then choose a probabilistic model of generation and use the standard statistics methods to estimate the ratings.

5.3 Improving recommendation

Many refinements can improve the recommendations obtained by the basic methods presented here. In particular, in some cases, *content filtering*, i.e., prediction of ratings given the description of items and users, provides some useful additional information. The description can also be used to increase diversity. For example, one may look for the list of items that maximize the sum of aggregated ratings under the constraint that the elements do not share all their attributes. The description of users and items are respectively in the files `u.user` and `u.item` of the MovieLens database. The `imdb` field of the item table can be used to get more attributes from the IMDB database.

Another standard improvement is to manage more precisely *serendipity*, i.e., to suggest items that are more risked. It may happen for instance that an item has been rated by only few users. If it turns out that all of them are enthusiastic, it may be worth proposing the item even if the support is low. For example, in user-based collaborative filtering the first aggregation function can be modified to base the means only on users who have produced ratings. It yields the same problem of trade-off between sparsity and noise

Taking context into account to filter the recommendation results is another interesting issue. For example, one may try to produce a recommendation for a given bag of keywords looked up in the attributes of the items. Explanation is another direction of improvement of recommendation (i.e., help the user to understand why s/he got this recommendation). The cold start problem (users or items with very few ratings) is also an important topic of research and can be easily experimented. Recommendation can benefit from interacting with the user to modify the recommendation process based on feedback. Finally, one may try to recommend to a group of users instead of a single user.

The project will try to improve the recommendation in some of these directions.