Web Search, Télécom ParisTech

Recommender Systems

Alban Galland (firstname.lastname@inria.fr)
Bogdan Cautis (lastname@telecom-paristech.fr)

18 March 2010

The purpose of this labs session is to compute recommendations of movies using a snapshot of the MovieLens database and the collaborative filtering techniques. More information about MovieLens can be found on `http://www.movielens.org`.

1 What to know before starting?

1.1 SQL

These labs uses SQL to access and manipulate data. If you are not familiar with this language, a little introduction can be found on `http://www.w3schools.com/sql/sql_intro.asp`.

1.2 Connection to the database

On a unix shell, use the command `mysql -h mysql.infres.enst.fr -P 3307 -u login -p login` where login must be replaced by toto _ _ to connect to MySQL. You can then type SQL command in the MySQL shell. You should use a file to paste and copy command rather than directly type it in the MySQl command. You can also execute all the SQL commands of a file by using the command `SOURCE filename`. The full documentation of mysql can be found on `http://dev.mysql.com/doc/refman/5.1/en/index.html`.

1.3 MovieLens data

This labs uses two tables of the MovieLens Snapshot: the `ratings` table and the `items` table. The `ratings` table is the list of the ratings, the `items` table is the list of the movies. One can access their schema by using commands `DESCRIBE ratings;` and `DESCRIBE items;`.

2 Data analysis

As discussed in the lecture, the quality of the data has a big influence on the result of the recommendation. There is a list of interesting analyses. The SQL commands corresponding to the analyses are given on `http://alban.galland.free.fr/Documents/Enseignements/INF396/data_analyses.txt`

- number of users, movies and ratings and average rating.
- distribution of the ratings

- distribution of the number of ratings by user
- distribution of the number of ratings by movies
- distribution of the average ratings by user
- distribution of the average ratings by movies

1. Given the result of these analyses, what can you say on the data?

3 Global recommendation

Global recommendation is roughly a way to give the movies with the best average rating. The following command give this list:

```
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;
```

1. What is the problem with this command? How can you correct it to have a better result? Try the corrected command.

The following command give the 30 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 30
   ) AS avgratingbyitems
WHERE items.itemid = avgratingbyitems.itemid
ORDER BY nbratings DESC;
```

Pick 10 movies you know in this list and give them a rating using the command

```
INSERT INTO me VALUES (id1,rating1), (id2,rating2), ...  (id10,rating10);
```

You may want to verify your list using the command

```
SELECT title, me.itemid, rating
FROM me, items
WHERE me.itemid=items.itemid;
```

4 User-based collaborative filtering

4.1 Correlation

Usually, one should compute the whole correlation matrix between users. To avoid too much computation time, we will only compute the correlation between all users and you. The following command

do the job with the standard cosine correlation measure:

```
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;
```

Compare the best user XXX to your ratings using the following command:

```
SELECT me.itemid AS itemid, me.rating AS myrating, ratings.rating AS herrating
FROM ratings, me
WHERE userid=XXX AND ratings.itemid=me.itemid
ORDER BY me.itemid;
```

1. What is the problem with the computation of the correlation? How can you correct the command, knowing that you can define a constant named `varname` using the command `SET @varname=(SELECT ...);` and then use this constant using `@varname` everywhere (reasonnable)? Try the corrected command.

2. What are the problems now? How can you correct them? Try the corrected command.

When it is done store the result into the `temp` table, using the following command:

```
INSERT INTO temp SELECT ...
```

4.1.1 Recommendation

The weighted aggregation corresponds to the following command

```
SELECT title, items.itemid, score, nbratings
FROM items,
   (SELECT itemid, sum(ratings.rating*temp.score) AS score,
      count(temp.userid) AS nbratings
   FROM temp, ratings
   WHERE temp.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 you already know about by adding the following expression on the right place on the previous command

```
itemid NOT IN (SELECT itemid FROM me)
```

1. What is the problem with these recommendations? How can you correct it? Try the corrected command.

2. Finally, try a different number of users as intermediate seeds using the commands `DELETE FROM temp` to clean the `temp` table. What can you observe?

You may also want to see the movies you are likely to dislike, replacing `DESC` by `ASC` in the previous commands.

5 Item-based collaborative filtering

1. How can you rewrite the previous commands to do item-based collaborative filtering? Try the corrected commands.