Lab 5: MongoDB Application

Due date: February 10, 11:59pm.

Lab Assignment

Assignment Preparation

This is a pair programming lab. Select your partner. There are 36 people in the class, so there should be 18 pairs working on this lab.

Application

Build a simple moive recommender/movie rating predictor system using the MovieSurvey dataset.

The core ideas behind movie rating predictions are based on a collection of techniques called collaborative filtering. Collaborative filtering assumes the following key idea:

> Users who share similar ratings on a lot of items, will continue having similar ratings on items that have not yet been rated by one of them.

That is, in order to predict, which movie a person X may like, we need to look at the current rating history for person X, find other users whose ratings are *similar* to X, and find what movies that X has not seen, they like.

Note: Because the MovieSurvey data you generated has random ratings, there is no expectation that your predictions will actually be any good. However, the goal of this assignment is for you to implement a specific set of computations, and confirm its validity. If you do it right, you *will* be able to predict movie ratings within reason on real, not spoofed data[1]

[1]There are some movie ratings datasets available. We can discuss incorporating their

Similarity between users. Traditionally, collaborative filtering methods use only the ratings information to compare individual users. Given two vectors of ratings: $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, a typical way of computing similarity between $\mathbf{x}$ and $\mathbf{y}$ is Pearson correlation:

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n}(x_{-}\bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the means of vectors $\mathbf{x}$ and $\mathbf{y}$ respectively.

Movie rating estimators. Let $U = \{u_1, \ldots, u_n\}$ be a list of people, and $M = \{m_1, \ldots, m_K\}$ be a list of movies. Let $r(u_i, m_j) = r(i, j) = r_{i,j}$ be the notation reserved to represent the rating of movie m_j by person u_i. In what follows, we assume that if $r_{ij} = 0$, then person u_i has not seen movie m_j and does not have a rating for this movie.

There is a variety of ways in which one can estimate a movie rating $r_{i,j}$ for user u_i based on the existing ratings $(r_{i,1}, \ldots r_{i,K})$ (some of which can be 0). Let K_i be the number of *non-zero ratings* user u_i has.

Average user rating. The most simple way is to ignore everyone else and estimate the rating of a given movie as the average of all the rankings user has given to other movies.

$$r_{i,j} = \frac{1}{K_i} \sum_{k=1, k \neq j}^{n} r_{i,k}.$$

Average movie rating. Another simple way is to ignore other ratings the user gave to the movie and estimate the rating for this movie as the average of the ratings *everyone else gives it*. Let M_j be the number of non-zero ratings for move m_j. Then the rating estimate is:

$$r_{i,j} = \frac{1}{M_j} \sum_{k=1}^{n} r_{k,j}.$$

Weighted sum predictor. As we mentioned above, there are reasons to believe that people who likes the same movies as person u_i would be better as predictors of u_i's rating of the movie. Therefore we should be giving their predictions more weight.

The weighted sum predictor then can be written as follows:

$$r_{i,j} = C_i \cdot \sum_{k=1, k \neq i} sim(u_i, u_k) \cdot r_{k,j},$$

use into this assignment.

where C_i is the normalizing factor set to be

$$C_i = \frac{1}{\sum_{k=1,k\neq i} sim(u_i, u_k)}$$

Adjusted weighted sum. Users u_i and u_k may have very similar tastes, as captured by their similarity score $sim(u_i, u_j)$, however one user may tend in general to assign higher ratings to the movies than the other (what one thinks is a 5 the other thinks is a 7). Pearson correlation coefficient will ignore such differences and still be high. But our predictor will be incorrect.

A way to fix this is to combine our average user rating predictor with the weighted sum predictor and to predict the difference between the movie rating for movie m_j and the average movie rating for the user. This can be done using the following estimator:

$$r_{i,j} = r(\bar{u_i}) + C_i \cdot \sum_{k=1,k\neq i} sim(u_i, u_k) \cdot (r_{k,j} - r(\bar{u_k})),$$

where $r(\bar{u_k})$ is the average rating of user u_k (see above).

N Nearest Neighbors Predictors. People with low similarity to u_i can probably be ignored when producing predictions. To do this, the average movie rating, the weighted sum predictor, and the adjudsted weightded sum predictor can be restricted to the set of N (for a given number N, e.g., $N = 10$) people who have the **highest similarity** of ratings to u_i.

This gives you a total of **seven** predictors to implement.

Your Program

For this lab, do the following.

Dataset Generation. Generate a reasonable sized dataset (a few hundred people) using your Lab 1-2 movie ratings generator. Ensure that sufficient number of movie ratings in your dataset are set to 0.

Develop MongoDB implementations of predictors. Develop seven MongoDB aggregation pipelines: one for each predictor, that given a specific unique Id of a record in the movieRatings collection, and a number between 0 and 12 specifying the Id of the movie, returns the appropriate predicted value for the movie given your current collection of movie ratings. The first MongoDB query shall implement the average user ratings, the next – the average movie rating, and so on.

For Nearest N Neighbors predictors assume that you will also have a value N passed to the MongoDB aggregation pipeline.

Debug and test these pipelines.

Note. These pipelines may involve the `$lookup` operation as you may need to join your collection to another collection. You may need to create temporary collections to host the information you need to join. You may need to play tricks to ensure that the proper join happens. The temp collection creation can happen as a separate operation, also involving an aggregation pipeline followed up by an insertion operation.

Note 2. If you are asked to predict a rating that already exists in the collection (i.e., a score that is not set to 0), you must exclude the score itself from consideration (i.e., essentially replace it with a 0 for the purposes of predicting the rating).

Compute Prediction Errors. For each predictor, set up a computation of the prediction error, for the prediction requests, where you are asked to predict a score that already exists in the collection. Be ready to output this error when the appropriate requests come (see below).

Ratings predictor software. Write a ratings predictor program in Java (`ratingsPredictor.java`) that works as follows:

1. Using Java MongoDB connectivity package connects to our MongoDB server and properly authenticates the user. The program shall read authentification information from the `user.auth` file stored in the same directory as the program itself. The `user.auth` file will have the following format:

   ```
   {
   authDb: <authentificationDB>,
   user: <user>,
   password: <password>
   db: <database>
   }
   ```

   providing the name of the authentification database, the username, the password, and the name of the database into which the data collection of movie ratings will be uploaded.

   Check collection existence. Your program shall check if a collection by the name `ratings` exists in the database specified in the `user.auth` file. If the collection exists, your program proceeds. If the collection does not exist, your program shall upload the dataset you generated (see above) to this collection.

   Ratings computations. Your program shall take as input a JSON file with the following format:

```
[
{ RID: <id>,
  Movie: <N>
},
...
{ RID: <id>,
  Movie: <N>
}
]
```

where each JSON object in the array is a specification for one rating to be predicted. For the sake of simplicity, we assume that you will generate all your users with RIDs that start at 1, and increment by 1.

For each JSON object in the input JSON array, you must compute all seven predicted ratings. Additionally, if the given rating already exists in the system, you shall compute and report both the predictions and the rating itself.

Output each rating prediction as follows. For a rating that does not exist in the database:

```
{
 RID: <RID>,
 respondent: {first: <firstName>,
              last: <lastName>
             },
 MovieId: <movieId>,
 MovieTitle: <title>,
 predictions: [ {type: "Average User", prediction: <Number>},
                {type: "Average Movie", prediction: <Number>},
                {type: "Weighted sum", prediction: <Number>},
                {type: "Adjusted weighted sum", prediction: <Number>},
                {type: "NNN Average User", N: <Number>, prediction: <Number>},
                {type: "NNN Weighted sum", N: <Number>,  prediction: <Number>},
                {type: "NNN adjusted weighted sum", N: <Number>,  prediction: <Number>},
              ]
}
```

For a prediction of an existing rating, report your results in the following format:

```
{
 RID: <RID>,
 respondent: {first: <firstName>,
              last: <lastName>
             },
 MovieId: <movieId>,
 MovieTitle: <title>,
 rating: <rating>,
 predictions: [ {type: "Average User", prediction: <Number>, error: <Number>},
```

```
                    {type: "Average Movie", prediction: <Number>, error: <Number>},
                    {type: "Weighted sum", prediction: <Number>, error: <Number>},
                    {type: "Adjusted weighted sum", prediction: <Number>, error: <Number>},
                    {type: "NNN Average User", N: <Number>, prediction: <Number>, error: <Number>},
                    {type: "NNN Weighted sum", N: <Number>,  prediction: <Number>, error: <Number>},
                    {type: "NNN adjusted weighted sum", N: <Number>,  prediction: <Number>, error: <Number>},
                  ]
}
```

For the output — hardcode the movie title when generating the output itself. This can be done in Java rather than in MongoDB.

For N Nearest Neighbor methods, your program should pick some value for N, and it should report it in the appropriate places when outputting the ratings. You can use $N = 10$, for example.

Please note, for this assignment, **we do not care how accurate your predictions are**, because you are generating the data randomly. But we do care that the predictions are computed correctly, so all of your programs shall produce the same output on the same input.

Submission

Submit all the code you have developed, a copy of `user.auth` file you have used (I may use yours, or my own for the purposes of testing), a JSON file containing one test collection you have used. Include a `README` file with the list of students on the team, **precise compilation and running instructions** and any other information you deem important.

Use `handin` to submit as follows:

```
$ handin dekhtyar lab05 <FILES>
```

Good Luck!