Lab Session 9: Recommenders from Scratch

Alex Carrillo Alza, David Bergés Lladó
Cerca i Anàlisi de la Informació, Ciència i Enginyeria de Dades (UPC)

November 25, 2019

1 A Recommender skeleton

In order to implement in Python the item-to-item and user-to-user recommenders, we build a skeleton that repeatedly ask for a list of movies with ratings and calls the recommender to provide recommendations given this list, and prints the title of those recommended movies with their predicted rating.

It should be pointed out that our first aim was to generate a single output by defining the two previous algorithm approaches and combining their recommendations somehow. We tried unsuccessfully to put the above into practice but many implementation errors were given and we desisted because it was not the focus of the report.

In the following subsections we describe the implementation details of each recommender. However, please note that the provided code has been modified to ease data structure access and type consistency. The most important is that in the `__init__` method of the `Recommender()` class, the *user_ratings* and *movie_ratings* dictionaries have been filled with dictionaries as well, instead of lists that were used before.

2 Implementation

2.1 User-to-user recommender

Broadly speaking, our implementation of the user-to-user recommender is based on: firstly compute the k-nearest-neighbors of the user given her/his ratings list, get their corresponding reviewed movies and that the user has not seen, and finally predict the rating of the *top*-recommended movies.

Moving on to the extraction of the neighbors movies, the implementation has been straightforward thanks to the use of `set()` structures and list comprehension, which have helped a lot when building the listing. Now, the computation of the predicted rating of those movies has been tricky. On the one hand, many redundant computations have been avoided by, for example, using the precalculated similarities obtained in the 'search for neighbors by Pearson correlation' phase. On the other hand, after several trial and test implementations, the prediction of each movie has been calculated by combining the ratings and using the average weighted by similarity of user a to b. The fact of not having used the adjustment to consider differences among users is because the addition of the user rating average $\bar{r}_a$ and the difference factor $(r_{b,s} - \bar{r}_b)$ induced increments above 5.0 in the predicted ratings.

2.2 Item-to-item recommender

In general terms, the item-to-item recommender is similar to the previous one but with the particularity of finding a set of items similar to the target one. Our implementation is based on: firstly get the movies that the user has not seen, find the k-closest movies by similarity for each one, and finally predict the rating of the *top*-recommended movies.

As regards similarity and prediction formulas, those are essentially the same as for the user-to-user case. For this reason, the procedure is the same as the previous one but now focusing on the movies, not on the users.

We take advantage of the previous similarity function to compute Pearson correlation between two movies given their ratings and users list. However, we modified the `predict()` function in order to avoid checking if the user has seen each movie, as the movies-to-rate set defined at the beginning of the procedure already verifies that. In addition, after observing that some predictions were really high, we detected it was due to really small similarity values in the denominator of the predict formula. To solve this, we only considered significant positive similarities from a 0.01 threshold when iterating. This same nuance has been applied in the user-to-user predict of rating function.

3 Experimenting

3.1 The sci-fi geek

We have created a set of 10 sci-fi movies which most of the rates are 5.0 and some have a value of 4.0.

Top-3 recommended movies

User-to-user	*Item-to-item*
Dark City (1998), Adventure/Film-Noir/Sci-Fi/Thriller	Stargate (1994), Action/Adventure/Sci-Fi
20 Million Miles to Earth (1957), Sci-Fi	The Fantastic Planet (1973), Animation/Sci-Fi
The Time Machine (2002), Action/Adventure/Sci-Fi	20 Million Miles to Earth (1957), Sci-Fi

To depict the result of asking the recommender we show the top-3 recommended movies. Surprisingly (or as we expected at first), those have at least one sci-fi tag, which means the recommender makes sense. It should be noted that both returned sets, *user-to-user* and *item-to-item*, are different. As the approaches are different, we expected they were not the same at least. In addition, all those movies (and the whole long list up to 40, say) have a predicted rating of 5.0 and we think it is due to the sheer volume of movies there are.

3.2 The cowboy girl

We have created a set of 10 movies about cowboys with all the ratings set to exactly 5.0.

Top-3 recommended movies

User-to-user	*Item-to-item*
The Searchers (1956), Drama/Western	Lonesome Dove (1989), Adventure/Drama/Western
High Noon (1952), Drama/Western	Paint Your Wagon (1969), Comedy/Musical/Western
Mildred Pierce (1945), Musical/Film-Noir	She Wore a Yellow Ribbon (1949), Western

For those recommendations, all are related to the topic again, both sets are different and the predicted ratings are kept to 5.0. Just highlight that *Mildred Pierce* film is not labeled on the western topic, but it seems related to it.

3.3 The kids family

We have created a set of 10 animation and children movies which most of the rates are 5.0 and some have a value of 3.0.

Top-3 recommended movies

User-to-user	*Item-to-item*
Looney Tunes (2003), Animation/Children/Fantasy	Muppets From Space (1999), Children/Comedy
The Iron Giant (1999), Adventure/Drama/Sci-Fi	A Christmas Story (1983), Children/Comedy
Milo and Otis, (1986), Adventure/Comedy/Drama	Thumbelina (1994), Animation/Children/Fantasy

The expected results are the same as before: the recommended movies are topic-related and with a predicted rating of 5.0. It should be pointed out that, in our opinion, the *item-to-item* recommender seems to work a little bit better than the other one, not just in this case but in many of the evaluated tests.

4 Optional work

4.1 Complexity analysis

Let m be the total amount of movies, u be the total amount of users and r the total amount of ratings that these users produced.

Memory complexity or storage complexity is as simple as considering the total amount of memory that would cost to store the dictionary structures representing the latter m, u and r. For simplicity, if we consider that all the movies have been rated by the same amount of users and that all the users have rated the same amount of movies, *i.e.* they follow a uniform distribution, every movie will have r/u ratings and every user will have rated r/m movies. Moreover, if we want to give a more concise approximation of the total memory cost, we will consider the structure of the latter dictionaries, where every user and movie are represented by a unique integer ID and every rating is a float between 0.0 and 5.0 (both `32 bits` or `4 bytes`), in total we will have (in bytes):

$$(4^2 \times m \times \frac{r}{u}) + (4^2 \times u \times \frac{r}{m})$$

plus the implicit additional overhead for hashing the dictionaries in memory.

Regarding the complexity of computing every similarity, it is easy to notice that in the worse case scenario both users will have rated the same movies (or both movies will have been rated by the exact same users) thus having to loop over the entire list of dimension r/u or r/m for each case respectively. The cost of every loop iteration it is considered to be constant $O(1)$, and therefore the cost of similarity computation will be of $O(r/u)$ and $O(r/m)$ respectively.

When computing predictions with the *User-to-User* scheme, we need to compute all the similarities between the new rating list and all the users, with cost $O(u \cdot r/m)$, followed by sorting the whole dictionary with cost $O(u \cdot log(u))$ and keep the highest knn values.

The next step is to find the movies to rate, *i.e.* the movies that have been reviewed by some of the knn nearest neighbors and that the user in question has not reviewed. In order to do that we loop through all the movies of the knn neighbors checking that our user has not seen them in $O(knn \cdot r/m)$.

Finally, for every movie found in the latter step we have to calculate its prediction, where we have to loop through all the movies rated by all the neighbors taken into consideration accessing into their dictionaries and sub-dictionaries. Accessing dictionaries takes constant cost thus leaving the total cost for every prediction to be $O(knn \cdot r/m)$.

The same cost for *Item-to-Item* can be analogously calculated.

4.2 Like-Dislike ratings

If we wanted to implement a recommendation system working with a **like-dislike** scale we could map the latter to 0 and 1 respectively, thus leaving us with a binary vector. As we now have 'integers' it would be very easy to implement the same system as before (the one with (0,5) rating scale), and predicting the rating as it would be a probability indicating if the user likes the movie in question. In this manner, we would be recommending to the user the movies that he or she is more likely to like with a certain probability.

If we wanted to implement a system that only worked with *likes*, it would be more difficult because the fact that the user did not *like* a certain movie does not mean that he/she did in fact not like it. We could make the latter assumption and implementing the same system proposed above, with a (0,1) rating scale, but notice that in theory it should be much less precise.