Information Retrieval and Analysis
December 2020

Lab Session 10: CF recommender from scratch

PROJECT REPORT

Elías Abad Rocamora
Victor Novelle Moriano
Barcelona, UPC - FIB & FME & ETSETB

Implementation of User-to-user and Item-to-item recommenders:

When using Collaborative filtering to recommend products to users we have two main families of methods: *User-to-user CF* and *Item-to-item CF*. The first of them exploiting the correlation between users' rating criteria and the second one the correlation between products' common buyer ratings.

User-to-user CF: $$Pred(a,s) = \overline{r_a} + \frac{\sum_{b \in User} sim(a,b) \cdot (r_{b,s} - \overline{r_b})}{\sum_{b \in User} sim(a,b)};\ a \in Users,\ s \in Movies$$

Item-to-item CF: $$Pred(a,s) = \overline{r_s} + \frac{\sum_{t \in Movies} sim(s,t) \cdot (r_{a,t} - \overline{r_t})}{\sum_{t \in Movies} sim(s,t)};\ a \in Users,\ s \in Movies$$

Despite being two quite different approaches, these two methods have many things in common and when implementing them, many functions could be used for both recommenders to save up space and make our code more modular.

But it wasn't all so easy. In order to structure these two algorithms in modules, a higher level of abstraction was needed and therefore we spent more time debugging, commenting and understanding what we were doing.

When coming to the finest details of our code, we would like to highlight some aspects. First of all, to compute the similarity between two rating vectors, we decided that they had to have at least 3 films of users (depending on whether they are user or movie ratings) in common to estimate with some guarantees their similarity. Also, a threshold variable was added in order to decide if two lists were similar.

Testing our code:

In order to analyze the performance of our recommenders, we have elaborated two lists with some of our favourite films and discussed if the films the system recommends are relevant to us.

Elías	**Víctor**
Forrest Gump (1994): 5	Interstellar (2014): 5
Pulp Fiction (1994): 5	Inkheart (2008): 4
Star Wars: Episode I - The Phantom Menace (1999): 5	Avatar (2009): 4.5
American Gangster (2007): 4	Zathura (2005): 4
Night at the Museum (2006): 4	Saw II (2005): 3.5
Interstellar (2014): 5	Inception (2010): 4
Fight Club (1999): 5	Sherlock Holmes (2009): 4
Terminal, The (2004): 4,5	WALL·E (2008): 5
	Elysium (2013): 4.5

Elías	**Víctor**
User-to-User CF	
1. Barcelona (1994)	1. Pirates of the Caribbean: The Curse of the Black Pearl (2003)
2. Where the Buffalo Roam (1980)	2. Star Wars: Episode VI - Return of the Jedi (1983)
3. Meatballs (1979)	3. Lord of the Rings: The Return of the King, The (2003)
4. Cheech and Chong's Up in Smoke (1978)	4. Airplane! (1980)
5. Wolf of Wall Street, The (2013)	5. Apollo 13 (1995)
Item-to-item CF	
1. Impostors, The (1998)	1. Gozu (Gokudô kyôfu dai-gekijô: Gozu) (2003)
2. Central Station (Central do Brasil) (1998)	2. The Pacific (2010)
3. Strada, La (1954)	3. Happy Together (a.k.a. Buenos Aires Affair) (Chun gwong cha sit) (1997)
4. Whole Wide World, The (1996)	4. Dead Man's Shoes (2004)
5. [REC] (2007)	5. Saw (2003)

When analyzing the recommendations for Elías, we see that in general, the films recommended are not quite relevant. He has only seen *The Wolf of Wall Street* and thinks that he might like *Where the Buffalo Roam (1980)*. He also hates *[REC],* so it's a quite poor recommendation. Maybe by adding more films to the list and also adding disliked movies, this recommendation could be better.

However, the recommendations for Víctor are, in general, pretty good. Most of the films were already seen by him and he liked them. In the item-to-item recommendation, *Happy Together (a.k.a. Buenos Aires Affair) (Chun gwong cha sit) (1997)* doesn't fit in the list, as it is a romantic movie, differing from the rest of the topics on the list. This also occurs in the user-to-user recommendations for the movie *Airplane! (1980).*

Optional 1: ML approach

In order to perform an evaluation of the ratings obtained by both types of collaborative filtering, we coded a new script, *train_test.py*. In it, we split our original user data in *[percentage]* (which is a hyperparameter set by the user) for training and the rest for the test.

After applying this partition, the training subsample is used as the checking database for the *Recommender.py* script and the test file is used as input. We decided to use this implementation because it allowed us to use the *Recommender* file as a library, without needing to perform modifications on its code to adapt it to this new approach.

For each user on the test sample, we perform a *Leave One Out* error testing. This means, for each one of the movies rated by the user, we predict the rating of that film using the rest of the movies rated by the user. Once we have generated the prediction, the computation of the error regarding the original rating is performed.

The prediction of the rating is performed by both methods, *user-to-user CF* and *item-to-item CF,* in order to check the difference of performance by both approaches. As a comparison metric, the mean absolute error by the film is used.

As encoding decisions, in order to avoid problems from the appearance of labels on the test ratings that do not exist on the test data, we decided to not provide any prediction on those cases. As a result of that, none of these cases has an impact on the final error computation. This decision was made with the intention to evaluate the “true” performance of both methods, which means comparing the predictions that both can perform accurately, instead of adding the mean of the user/film prediction to the test label, which could add bias.

After several tests, we have obtained the following results:

	User-to-user		**Item-to-item**	
Threshold	**Avg Error**	**# Predictions**	**Avg Error**	**# Predictions**
0	0.59813	5733	0.63970	5529
0.3	0.58637	5221	0.64262	5529
0.6	0.53750	2400	0.67564	5311
0.9	0.50122	492	0.75872	3753

As it can be seen, the number of predictions performed by the *item-to-item CF* is higher than the number of predictions performed by the *user-to-user CF* for all the values of the *threshold* hyperparameter except for the first case. This can lead to excluding relevant films for the user in the *User-to-user CF* recommendation and therefore having a more poor diversity.

An aspect that we find interesting and that we would like to study if we had more time is that while with *User-to-user CF,* the Avg Error decreases with the number predictions performed, in the *Item-to-item CF* case, the behaviour is completely opposite, as the number of predictions decreases, Avg Error increases. We find this behaviour very strange and cannot think of an explanation to it.

Optional 2: Asymptotic time cost

In this section, an asymptotic time complexity analysis for both recommender systems has been performed. In order to ease this analysis, several variables have been defined:

- **U =** Number of users
- **M =** Number of movies
- **LR =** Average length of a user rating list
- **LC =** Average length of a movie rating lists
- **L =** Length of the rating list of the user to recommend films to

which in our case, for the small dataset, have the following values:

- **U =** 610
- **M =** 9724
- **LR =** 165.30491803278687
- **LC =** 10.369806663924312
- **L = LR** (We suppose is an average user rating list)

Now, with the use of these parameters, we proceeded to analyze the functions shared by both methods and afterwards, the ones that are specific for each one of them:

SHARED FUNCTIONS

similar_ratings:
We compare with every other (user or movie)'s rating list and compute the similarity. As the complexity of the similarity is *O*(LR or LC), the total complexity of this function is *O*((U or M) · (LR or CL)).

pred:
We iterate over every element rated by a similar user or movie and if it is in the elements of the list to compare, we compute the average ratings of the user or movie and iteratively compute the prediction, so it takes: *O*(LR · LC) in the user-to-user case *O*(LC · LR) in the item-to-item case which is the same, so *O*(LR · LC)

sort results:
There are asymptotically *O*(M-LR) movies that can potentially be recommended, so to sort this takes: *O*((M-LR) · log(M-LR))

USER-TO-USER FUNCTIONS:

The user-to-user implementation mainly consists in:

similar_ratings + pred_list + sort

so each one of these functions will be checked:

similar_ratings: *O*(U · LR)

pred_list:
We compute the average rating of the user: *O*(LR).
Then we create a set containing all the films that could be potentially recommended, that is, the films that the users similar to ours have seen.
This set is created by taking all the films from the similar users, so it takes *O*(|similar_users_size|) = (*as similar_users is contained in the set of all users*) = *O*(U) to create this set.

After this, we iterate over all the films in the set and check that our user hasn't rated them, so that is *O*(M) iterations, and for each one of those films we compute the prediction. This loop takes *O*(M · LR · LC)

So, in total, this function has an asymptotic cost of: *O*(LR) + *O*(U) + *O*(M · LR · LC)
= *O*(M · LR · LC)

sort results: O((M-LR) · log(M-LR))

So, in total, the user-to-user implementation has a time complexity of:

$$O(U \cdot LR) + O(M \cdot LR \cdot LC) + O((M-LR) \cdot \log(M-LR))$$

ITEM-TO-ITEM FUNCTIONS:

The item-to-item implementation,on the other hand, consists in:

```
for new movies: (O(M-LR))
    similar_ratings + average_item + pred
+sort
```

We iterate over the complete list of movies in order to find new movies and compute the average ratings for that new movie in *O*(LC).
Inside the same loop, we compute the similar movies to that one: *O*(M · LC).
Then we compute the prediction for the new movie and append it: *O*(LR · LC)
Finally, we sort the results, which is *O*((M-LR) · log(M-LR)) in total:
O((M-LR) · *O*(LC + M · LC + LR · LC)) + *O*((M-LR) · log(M-LR)) = *O*((M-LR)) · *O*(LC · (1+M+LR)) + *O*((M-LR) · log(M-LR))) = *O*((M-LR) · *O*(LC · (M+LR)) + *O*((M-LR) · log(M-LR)))
~ *O*(M · M · LC) + *O*((M-LR) · log(M-LR))) = *O*(M^2 · LC)

Optional 3: Modifying the rating system

In this section, we will explain how our algorithms could be modified in order to adapt them to two different rating systems: the like/dislike and only considering only upvotes.

The main difference between the algorithms regarding the new rations would be the similarity function used between users/items. We should adjust the formulas to take into account the new nature of the data. We purpose the following functions:

Like/dislike:

$$sim(a,b) = 1 - \frac{\sum_{r \in AUB} |(r_a - r_b)|}{|A \cup B|}, \qquad \begin{cases} r_x = 1 \text{ iff } X \text{ liked the film} \\ r_x = 0 \text{ iff } X \text{ hasn't watched the film} \\ r_x = -1 \text{ iff } X \text{ disliked the film} \end{cases}$$

We have chosen the absolute difference between the vectors instead of using the Hamming distance in order to penalize to a greater extent if a like-dislike discrepancy is produced, instead of considering that it has the same weight as a rating-non rated one.

Upvotes:

$$sim(a,b) = 1 - \frac{Hamming\ dist|(r_a - r_b)|}{|A \cup B|}$$

In this case, we decided to use the classic Hamming distance, as our rating vectors are $\{0,1\}^d$. This means that all the discrepancies have the same importance, unlike the previous case.

Apart from these changes in the similarity measure, we wouldn't change anything else from our recommender class as it would work properly the way it is.