

Recommenders: Task 2

Bernat Esquirol

Problem statement

The base problem is figuring out which movies to recommend to a given set of users that have already ranked a several movies before. We will do this by training an algorithm that learns which rating would a user give for an unseen movie. We'll use the dataset movielens-100k. This dataset consists of 100.000 ratings from 943 different users of 1682 movies.

The code for the implementation done in this task can be found at:

https://github.com/bernatesquirol/recommenders_task2

Used Methods

We'll start with the notebooks done in class, a user based recommender notebook and a SVD notebook.

Item based method

We tried different approaches for the item based method. The basis of this method is computing a similarity matrix for the items of the dataset. So we'll have a 1682x1682 symmetric matrix and in each cell we'll have the similarity between movies. This similarity will be computed by looking at the users ratings. We'll pivot the ratings database to one such that has as rows the different movies, as columns the different users and as values the rating of that given user for that given movie.

	User 1	...	User 943
Movie 1	$r_{1,1}$...	$r_{1,943}$
...
Movie 1682	$r_{1682,1}$...	$r_{1682,943}$

The similarity will be a value between 0 and 1, and will be a function that takes as input two row vectors. So if most of the users rated similarly two movies, we'll consider them close. We'll compare several similarity measures. To see which performs better. Once we have the distance matrix we have to establish a method to give a rating. We tried two of these methods with these formulas

$$\hat{r}_{u,j} = \frac{\sum_{i \in P_k(j)} \text{sim}(i, j) \times r_{u,i}}{\sum_{i \in P_k(j)} \text{sim}(i, j)} \quad (1)$$

$$\hat{r}_{u,j} = \bar{r}_j + \frac{\sum_{i \in P_k(j)} \text{sim}(i, j) \times (r_{u,i} - \bar{r}_i)}{\sum_{v \in P_k(j)} \text{sim}(i, j)} \quad (2)$$

$\hat{r}_{u,j}$: predicted rating for user u

$r_{u,i}$: rating for user u and movie i , \bar{r}_j is the mean rating for the film.

$P_k(j)$: the k -nearest movies to j

This method takes long time to train (calculate distances between movies) and also is very slow to evaluate in the dataset given. Could be that the code I did is not the most optimal one, but still there are a lot of operations involved.

The code is done so that we can save the symmetry matrix for a given training data, so we don't have to retrain everything again.

To evaluate we have two methods also: the minimum squared error and the top-n precision recall measure ¹. For both of them I had to subsample the test set as it was too much computationally intensive for the whole trainset (20% of the database).

Implementation

I chose three distances (three different models) to evaluate their performance. Cosine similarity, euclid similarity and pearson similarity. I also calculated the symmetry matrices for the adjusted cosine and adjusted pearson. Those distances consist of subtracting from all the ratings the mean rating for the user. So we avoid the user bias. But I made the error in the prediction function (probably I didn't transform the test dataset the same way as the train dataset), so the evaluations where very bad, and as it was a very costly process I decided to skip them.

Results

```
----- cosine -----
RMSE basic of cosine: 1.0366071543385287
RMSE mean of cosine: 0.9285720967063059
----- pearson -----
RMSE basic of pearson: 4.378503532488526
RMSE mean of pearson: 4.535351302983304
----- euclid -----
RMSE basic of euclid: 0.9879176904758223
RMSE mean of euclid: 0.9013302016495014
```

We can see that although `euclid` does produce less squared error, in the P/R performs worse than `pearson`.

```
----- cosine -----
P/R basic of cosine: 0.423728813559322
P/R mean of cosine: 0.7584745762711864
----- pearson -----
P/R basic of pearson: 0.8559322033898306
P/R mean of pearson: 0.9279661016949152
----- euclid -----
P/R basic of euclid: 0.6991525423728814
P/R mean of euclid: 0.8177966101694916
```

Factorization method

With the factorization method we start with a exactly the same matrix as the table in Item based methods, and we compute with a dimensionality reduction algorithm its compressed version. This is a much more quick learning procedure than Item based algorithm, and also will result in a much faster predicting model and evaluation. At the prediction step, things are easier also, as we only need to compute a product between the similarity between user reduced vectors and the item reduced vectors.

The drawback of this approach is that we lose some transparency in the algorithm as the dimensionality reduction algorithms are not as tangible as the procedure we did in the Item Based approach.

Implementation

As similarities in the prediction step we implemented the dot product and the cosine similarity.

Results

We can see how the dot product performs way better than the cosine similarity.

```
----- SVD_estimate -----  
RMSE dot 0.9340701096010212  
RMSE cosine 2.890285834744717  
  
----- SVD_estimate -----  
P/R dot: 0.8572741898414158  
P/R cosine: 0.29648356699609285
```

Conclusions

Overall the factorization methods give such speed that is difficult to stick to the item-based / user-based approaches. Although we can see more clearly how the prediction is made with the first approach, with a dataset such as that we have here, is not optimal at all.

Truth is the results we obtained are not very consistent, specially those in the item based recommendation. Also we couldn't have the adjusted cosine and adjusted pearson similarity that mayb improve the item based solution.

In the Factorization based method we could try different methods of dimentionalty reduction.

1. <https://www.researchgate.net/publication/221141030> Performance of recommender algorithms on top-N recommendation tasks ↗