Movies RecSys Report

Done by B20-AI-01 student Sergey Golubev

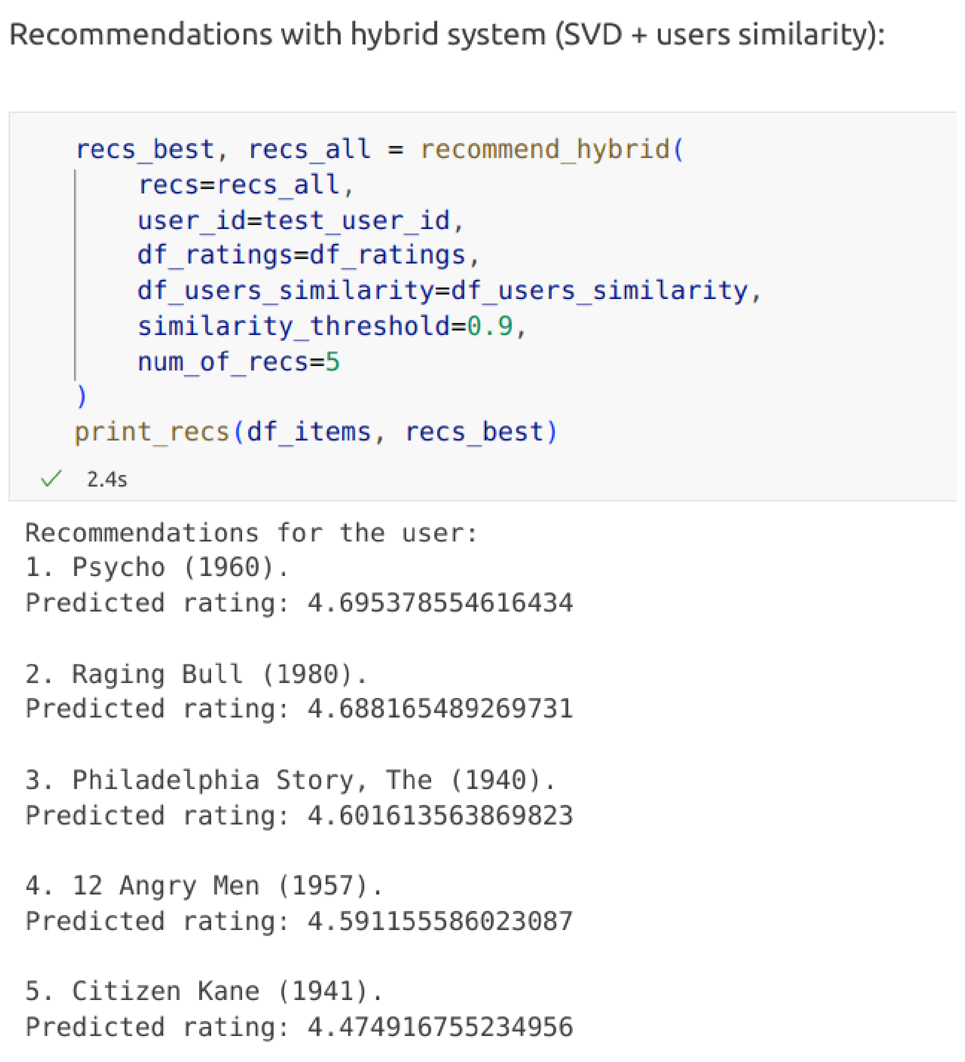
Introduction

The task is to implement a movies recommender system using MovieLens 100K dataset. My solution includes 2 models:

1. Pure singular value decomposition (SVD)
2. Hybrid: SVD + bias from similar users

Both algorithms were evaluated using RMSE metric and cross-validation.

Interface for user-friendly recommendations is provided:



Recommendations with hybrid system (SVD + users similarity):

```python
recs_best, recs_all = recommend_hybrid(
    recs=recs_all,
    user_id=test_user_id,
    df_ratings=df_ratings,
    df_users_similarity=df_users_similarity,
    similarity_threshold=0.9,
    num_of_recs=5
)
print_recs(df_items, recs_best)
```

✓ 2.4s

```
Recommendations for the user:
1. Psycho (1960).
Predicted rating: 4.695378554616434

2. Raging Bull (1980).
Predicted rating: 4.688165489269731

3. Philadelphia Story, The (1940).
Predicted rating: 4.601613563869823

4. 12 Angry Men (1957).
Predicted rating: 4.591155586023087

5. Citizen Kane (1941).
Predicted rating: 4.474916755234956
```

Data analysis

Data analysis is performed in `notebooks/1-EDA.ipynb`. The main steps are:

1. Removing unused columns.
2. Recognizing poor data (movies of unknown genre) and removing it from datasaset.
3. Re-indexing the `user_id` column, so that it would start from 0, not 1.

Users Vectors

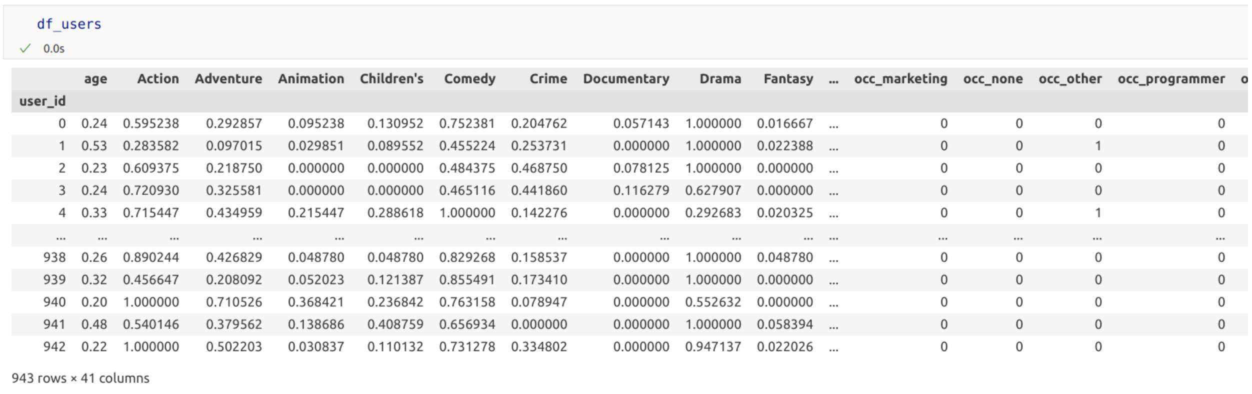
In the `notebooks/2-users-vectors.ipynb`, the users vectors are built. Each user vector contains the following info about users:

1. Age - integer, then normalized (divided by 100).
2. Sex - a boolean column (is male).
3. Occupation - one-hot encoded representation of the user's occupation
4. Genre favor vector. It is a vector of length 18 (number of genres). Each column contains the sum of ratings given to all the films of the considered genre. Then, each user's genre fevor vector is normalized independently.
    a. For example, a user rated five films: two Action & Adventure films with 9, and three Drama films with 7. Their genre favor vector will be the following:

Action	Adventure	Drama	(All the rest genres)
18	18	21	0

    And after normalization:

Action	Adventure	Drama	(All the rest genres)
0,3158	0,3158	0,3684	0



```
df_users
✓ 0.0s
```

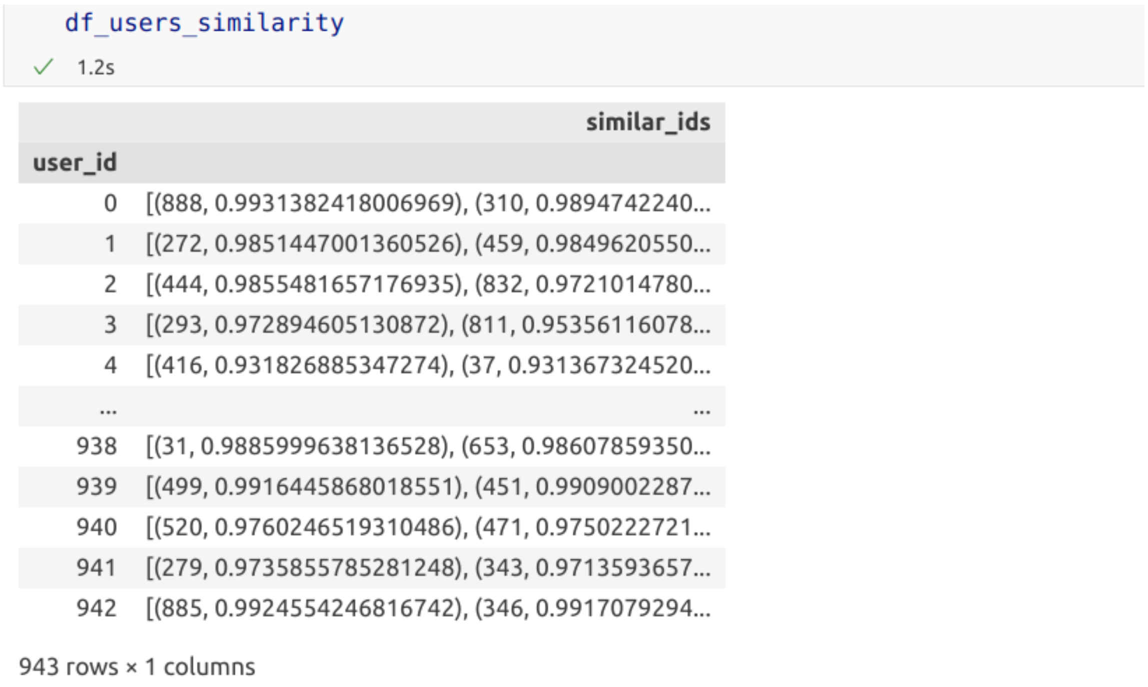
```
         age    Action  Adventure  Animation  Children's    Comedy     Crime  Documentary     Drama   Fantasy  ...  occ_marketing  occ_none  occ_other  occ_programmer  o
user_id
      0  0.24  0.595238   0.292857   0.095238    0.130952  0.752381  0.204762     0.057143  1.000000  0.016667  ...              0         0          0               0
      1  0.53  0.283582   0.097015   0.029851    0.089552  0.455224  0.253731     0.000000  1.000000  0.022388  ...              0         0          1               0
      2  0.23  0.609375   0.218750   0.000000    0.000000  0.484375  0.468750     0.078125  1.000000  0.000000  ...              0         0          0               0
      3  0.24  0.720930   0.325581   0.000000    0.000000  0.465116  0.441860     0.116279  0.627907  0.000000  ...              0         0          0               0
      4  0.33  0.715447   0.434959   0.215447    0.288618  1.000000  0.142276     0.000000  0.292683  0.020325  ...              0         0          1               0
    ...   ...       ...        ...        ...         ...       ...       ...          ...       ...       ...  ...            ...       ...        ...             ...
    938  0.26  0.890244   0.426829   0.048780    0.048780  0.829268  0.158537     0.000000  1.000000  0.048780  ...              0         0          0               0
    939  0.32  0.456647   0.208092   0.052023    0.121387  0.855491  0.173410     0.000000  1.000000  0.000000  ...              0         0          0               0
    940  0.20  1.000000   0.710526   0.368421    0.236842  0.763158  0.078947     0.000000  0.552632  0.000000  ...              0         0          0               0
    941  0.48  0.540146   0.379562   0.138686    0.408759  0.656934  0.000000     0.000000  1.000000  0.058394  ...              0         0          0               0
    942  0.22  1.000000   0.502203   0.030837    0.110132  0.731278  0.334802     0.000000  0.947137  0.022026  ...              0         0          0               0

943 rows × 41 columns
```

Users Vectors Examples

Similarity Lists

After the users vectors are built, we compute the pairwise cosine similarity between users, and save the similarity list for each user. The similarity list of a user A contains tuples `(user_B_id, cosine_similarity_between_A_and_B) for B in all_users`. The tuples are sorted by descending order of cosine similarity. We save the similarity lists for each user in separate file. We compute these similarities in offline in order to get them quickly during inference.



```
df_users_similarity
✓ 1.2s
```

	similar_ids
user_id	
0	[(888, 0.9931382418006969), (310, 0.9894742240...
1	[(272, 0.9851447001360526), (459, 0.9849620550...
2	[(444, 0.9855481657176935), (832, 0.9721014780...
3	[(293, 0.972894605130872), (811, 0.95356116078...
4	[(416, 0.931826885347274), (37, 0.931367324520...
...	...
938	[(31, 0.9885999638136528), (653, 0.98607859350...
939	[(499, 0.9916445868018551), (451, 0.9909002287...
940	[(520, 0.9760246519310486), (471, 0.9750222721...
941	[(279, 0.9735855785281248), (343, 0.9713593657...
942	[(885, 0.9924554246816742), (346, 0.9917079294...

943 rows × 1 columns

Similarity Lists Examples

Model Implementation

Pure SVD

SVD is a mathematical algorithm for matrix decomposition. The key idea behind SVD in recsys is to decompose the user-item rating matrix into three matrices: a user matrix, a singular value matrix, and an item matrix. Such decomposition captures hidden patterns in the data. We can reconstruct the original matrix from these components and predict missing values, i.e., the ratings of yet not rated items.

I used an SVD implementation from `scikit-surprise` library. You can find the model implementation in `models/recsys.ipynb`, and inference in `notebooks/3-recsys.ipynb`. f

Hybrid Model

Suppose we want to predict the rating of user U for a film F. The hybrid model will modifyThe pipeline of the hybrid model is the following:

1. Run the same pure SVD algorithm and obtain the predicted `rating_svd`. We will modify this rating to get `weighted_rating`.

2. From the similarity list of user U, take all the “similar users”: the ones who have the cosine similarity with U at least 0,9. Such a threshold is selected in `notebooks/2-users-vectors.ipynb`.
3. The resulted `weighted_rating` is basically a mean of the SVD prediction and the ratings by “similar users”.

   The ratings of the film F given by “similar users” are considered `relevant_ratings`. We use these relevant ratings to correct the `rating_svd`. The formula for the `weighted_rating` is the following:

   ```
   weighted_rating = (rating_svd + sum(relevant_ratings)) / (num_relevant_ratings + 1)
   ```

Model Advantages and Disadvantages

Advantages

1. Fast, no training process.
2. All the necessary data for the hybrid model is computed in offline.

Disadvantages

1. The whole model is stored in memory.
2. Cold start problem: the fewer ratings of the film we have, the more biased will be the prediction.
3. The same cold start issue is applicable to the hybrid model: the fewer ratings by “similar users” we have, the more each rating affects the final prediction.

Training Process

As SVD is a mathematical algorithm, there is no explicit training. What can be called training happens during the singular values computation.

Regarding the hybrid model, it is an add-on to the SVD that uses pre-computed cosine similarities. It is not trained in classical sense as well.

Evaluation

I computed the RMSE metric for both models. The cross-validation on 5 folds was performed. Refer to `benchmark/evaluate.ipynb`.

The results are quite impressive:

Model	RMSE
SVD	0.9382
SVD + users similarity	0.4694

Results

I enjoyed working on this solution, and I am quite satisfied with the resulted metrics. Seems like the average error of less than 0.5 out of 5 is not bad when we are trying to suggest content to a user.

I see more space for enhancement:

- introduce new features to the user vector, e.g., the preferred movie epoch (if a film is released in 50s, 60s, 70s, etc.)
- compute the average film rating and include it to the `weighted_rating` formula
- search for hyperparameters (similarity threshold)