TSKS33 Hands-On Session 2

Fall 2024

Version of this document: November 1, 2024

Preparation Work

Study

- In the course material (by E. G. Larsson): the sections on affiliation networks and cosine similarity
- In [1]: Chapter 4
- In [2]: Sections 3.1, 3.2.1, and 3.4.1–3.4.2

This lab assignment requires a significant programming effort.

Summary of tasks

In this computer project, we will study recommendation systems of the type that are used by, for example, Netflix. The objective of the recommendation system is to predict how much a user of the service would appreciate a certain movie and then recommend to each user those movies that the user is likely to enjoy. The prediction is based on how the user in question has rated other movies, and how other users have rated the different movies. You will be working in the file `session2.py` where a code skeleton is provided.

Obtaining the data files

All course participants will obtain three unique data files to be used for Task 1, `LiU-ID.training`, `LiU-ID.test`, `LiU-ID.moviename`, located in the directory

`/courses/TSKS33/ht2024/data/student_files`. This directory also contains the files `task---2.training`, `task---2.test` and `task---2.moviename`, to be used by all students for Task 2.

About the data files

We use real data from the MovieLens project (`www.movielens.org`). MovieLens is a recommendation website where users can rate movies they have seen and get personalized recommendations for other movies. We gratefully acknowledge the GroupLens Research Project faculty at the University of Minnesota, USA, who have kindly given us permission to re-distribute parts of their data to be used for this project work. We stress that the data files represent derivative works of the MovieLens database, obtained by randomly permuting the data and removing certain elements, and must not be mistaken for the original MovieLens dataset. **The data files must not be distributed beyond the TSKS33 class.**

Task 1: Implement the Baseline Predictor

Implement the baseline predictor presented in [1, Sec. 4.2.1]. Use the training data set in the file `LiU-ID.training`, which contains ratings made by $U = 2000$ users on $M = 1500$ movies. In total, the file contains roughly 200000 ratings. To evaluate the predictor you should use the test data set in the (individual) file `LiU-ID.test`, which contains roughly 20000 ratings. Each rating is a number from the set $\{1, 2, 3, 4, 5\}$. The files are stored in comma-separated value (CSV) format, and each row in the `.training` files has the format

```
u,m,r
```

where `u` is a user ID number, `m` is a movie ID number, and `r` is user `u`'s rating of movie `m`. The movie names in plaintext are listed in the file `LiU-ID.moviename`.

Baseline predictor. The baseline predictor models the data according to

$$\hat{r}_{um} = \bar{r} + b_{U,u} + b_{M,m} \tag{1}$$

where

- $\hat{r}_{um}$ is the predicted rating for user u of movie m,
- $\bar{r}$ is the average rating over all users and movies,
- $b_{U,u}$ is the bias of user u compared to the average $\bar{r}$,

- $b_{M,m}$ is the bias of movie m compared to the average $\bar{r}$.

The goal is to find $b_{U,u}$ and $b_{M,m}$ that minimize the root mean-square error (RMSE).

After the predicted rating, $\hat{r}_{um}$, is found as explained in [1, Sec. 4.2.1], do not round the predicted ratings to integers. However, truncate the predicted ratings by setting any prediction that falls below 1 to 1, and any prediction that exceeds 5 to 5.

Task. Implement the baseline predictor by solving the least-squares problem as explained in [1, Sec. 4.2.1]. To solve the least-squares problem in Python, use the function `scipy.sparse.linalg.lsqr` from the SciPy sparse array package. The matrix $\boldsymbol{A}$ and the vector $\boldsymbol{c}$ should be constructed based on the data in `LiU-ID.training`. After the optimal $\boldsymbol{b}^{\star}$ has been found, the predictor should be tested on the data sets in both `LiU-ID.training` and `LiU-ID.test`.

Make sure to obtain

(i) the RMSEs associated with the training data and with the test data. The RMSE values shall be reported with three decimals (`x.yyy`).

(ii) a histogram of the absolute errors (deviations between the predicted ratings and the true ratings), for the test data. When plotting this histogram, first round each predicted rating to the nearest integer in $\{1, 2, 3, 4, 5\}$. In view of the RMSE value computed in (i), does the plot look reasonable?

Use rounded data only for the histogram calculation. In Task 2, in the improved predictor, use truncated predicted ratings.

Task 2: Implement an Improved Predictor

Implement an improved predictor by using the "neighborhood method" with the cosine similarity metric for movies presented in [1, Sec. 4.2.4], see Equations (4.6)–(4.7). (It is also possible to use the cosine similarity for users.) Use the data in the file `task---2.training` to train the recommender algorithm and evaluate its performance on the test data in `task---2.test`. In addition, redo task 1 but for the files `task---2.training` and `task---2.test`. Compare the test RMSE values of the baseline predictor with the test RMSE values of the improved predictor, and report the improvement.

All students use the same files for this task. The data are stored the in the same format as the files used in Task 1.

The improved predictor in Eq. (4.7) [1], where N in $\hat{r}^N_{um}$ denotes "neighborhood method" is as follow:

$$\hat{r}^N_{um} = \hat{r}_{um} + \frac{\sum_{j \in \mathcal{L}_m} d_{mj} \tilde{r}_{uj}}{\sum_{j \in \mathcal{L}_m} |d_{mj}|}, \tag{2}$$

where $\mathcal{L}_m$ presents the set of the top L movies chosen as neighbors for movie m. In the summation in the denominator, all L values are summed regardless of whether user u has rated them. Note that this approach is slightly different than the example mentioned in [1].

Hints

- To load the $\boldsymbol{A}$-matrix, use the Python script `load_data.py`.

  Go through this script and explain how it works. Refer to the sparse matrix construction in session 1, task 3. Note that Python is 0-indexed! The dimension of $\boldsymbol{A}$ is $N \times (M + U)$, where N is the number of ratings, M is the number of movies, and U is the number of users.

- For task 2, it is suggested to use the cosine similarity for *movies*. (If using the cosine similarity for users instead, the reasoning is analogous.)

  Have a look at Eq. (4.6) in [1, Sec. 4.2.4]. Note that, summations are only over those users u that rated both movies i and j in Eq. (4.6).

  In calculating the cosine similarity d_{ij}, the textbook tells us that the summation is "of course only over those users u that rated both movies i and j in the training set".

  Note, however, that there might be only a single user that has rated a specific pair of movies, and in that case there is only one term in the sum. What is d_{ij} in that case? The nominator and denominator will cancel out each other, so $d_{ij} = 1$. This turns out to be the case for quite many movie pairs in the training set. Since d_{ij} always lies between -1 and $+1$, 1 is the maximum value that d_{ij} can assume.

  Now we are supposed to sort the absolute values $|d_{ij}|$ in descending order and pick the L largest ones. But if many of these values are 1, these will of course be among the top L, which would suggest that these movies were extremely similar – whereas in reality the reason was actually merely that they were only rated by a single user!

  Therefore, introduce a minimum number of users, u_{min}, that must have rated both movies i and j for the sum to be valid. For instance, d_{ij} is calculated as usual if at least, say, $u_{\text{min}} = 10$ users rated both movies, and otherwise $d_{ij} = 0$ (which is the most neutral value, neither similar nor dissimilar).

  By doing so it might make sense to increase L. You should experiment with different values of L and u_{min} to see what gives the lowest RMSE. For example, $u_{\text{min}} = 50$ and $L = 100$ may work fine, but you may experiment with a large range of values. Choosing these parameters is more art than hard science.

  For verification data set, the top 5×5 block of the similarity matrix with $u_{\text{min}} = 20$, i.e., `D[:5,:5]`, is

```
[[ 1.          0.00955895 -0.10700210 -0.12839974  0.          ]
 [ 0.00955895  1.          0.04347444 -0.02409967  0.          ]
 [-0.10700210  0.04347444  1.          0.10370475  0.          ]
 [-0.12839974 -0.02409967  0.10370475  1.          0.          ]
 [ 0.          0.          0.          0.          1.          ]]
```

The whole `D` matrix (`verification_D_mat.npy`) is given in student files folder for verification.

As a minor deviation from the algorithm described in [1], we keep all diagonal elements in the similarity matrix equal to one, i.e., $d_{ii} = 1$ for all movies i, regardless of u_{min}. This means that we always include a movie as its own neighbor. Note that the neighborhood method is not mathematically optimal in any sense; rather, the design of predictors is more art than hard science.

- To help you verify your code we provide a verification data set which can be found in the files `verification.*`. The RMSE for `verification.training` and `verification.test` for $L = 100$ and $u_{\text{min}} = 20$ is as follows

`verification.*`	RMSE for baseline predictor	RMSE for improved predictor
Training	0.891	0.792
Test	0.905	0.898

- When implementing the neighborhood method in task 2 you should see an improvement of at least one percent. The improvement will change when you change the parameters.

Examination

- Individual oral examination takes place in class (computer lab). Students are also expected to be able to answer questions relating to the course material. All students must study the pertinent course material before coming to the lab.

- Before asking for oral examination, you have to collect all generated plots/figures and answers into a document, for example, in PowerPoint or LibreOffice. Additionally, the code should be open and ready to be run upon request. It is suggested that you have already ran the code once so that the output is in the terminal.

- If a reference/sample solution is provided, you should format your solution the same way.

- Collaboration on this homework in small groups is encouraged, but each student should perform programming work individually, and individually demonstrate understanding of all tasks.
- Library functions from the Python standard library, numpy, scipy, matplotlib, and SNAP may be used freely. Copying of code from the Internet or from other students is prohibited.
- The program code you have written should be uploaded to Lisam (go to "submissions" and then select the lab session number) for an anti-plagiarism check.

  Plagiarism (copying of code from other students, from the Internet, or other sources) is a serious offense at LiU and normally leads to the filing of a disciplinary case.

Please note that we expect an improvement between the baseline and improved predictor of at least 1% for the file `task---2.test`.

References

[1] M. Chiang, *Networked Life: 20 Questions and Answers*. Cambridge University Press, 2012.

[2] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook, 2nd edition*, F. Ricci, L. Rokach, and B. Shapira, Eds. Springer, 2015.