Exercise 2

Due date: 25/05/2021, submission in pairs
Appointed TA: Ekaterina Chernodarov

Introduction

In this exercise you will implement various recommender system models and compare their performance.

Data

The data file is ratings.csv. Every record in the file is of the form user, item, rating, timestamp.

user – The user's unique identifier
item – The item's unique identifier
rating – The rating that was given to the item by the user, it is in the range $[0.5,5]$
timestamp – The timestamp in which the rating was given.

For the competitive part, the data file is ratings_comp.csv and it is in the same format.
The ratings.csv file has 100,000 ratings and the ratings_comp.csv file has 800,000 ratings.

Code Files

main.py – This file loads the data and calls the functions in ex2.py that you will need to implement. This file will not be submitted.

transform – This method removes users that have rated less than 5 items and items that have been rated by less than 7 users.

train_test_split – This method splits the data such that 0.8 of every user's ratings will be in the train dataset and 0.2 in the test dataset.

There are multiple classes defined in ex2.py. Recommender is an abstract class, and the other classes are derived from it. In every subclass of Recommender you will have to implement the initialize_predictor method (this is the method that will do most of the computation required to later make predictions), the predict method and some other methods that are task specific.

You may use the following external python modules: *pandas, numpy, sklearn* (in addition to the python internal modules).

You need to submit the file ex2_ID.py (rename the filename from ex2.py to ex2_ID.py before submission).

Part A (55 points)

In this part you will implement some baseline predictors.
The runtime of this part must not exceed 1 minute.

The first baseline model for recommender systems is $\hat{r}_{ui} = \hat{R} + b_u + b_i$ where $\hat{R}$ is the average of all the ratings in the user-item ratings matrix R, b_u is the average rating deviation for user u and b_i is the average rating deviation for item i.

1. Implement the methods of the class `BaselineRecommender`.
   Ignore the timestamp parameter for rating predictions.
   Make sure the predicted rating is in the range $[0.5,5]$.

To evaluate the model, we need to compute the RMSE of the predictions with respect to the real ratings.

$$RMSE = \sqrt{\frac{1}{|R|} \sum_{(u,i)\in R} (r_{ui} - \hat{r}_{ui})^2}$$

2. Implement the method
   `def rmse(self, true_ratings) -> float`
   which calculates the RMSE score for the predictor with respect to the true ratings.
   true_ratings is a DataFrame (user, item, rating, timestamp)
   Use the abstract function
   `def predict(self, user: int, item: int, timestamp: int) -> float`
   which is implemented in every subclass of `Recommender`.

Collaborative filtering makes use of the similarities of users and the similarities of items to make predictions. We will be using only the user similarities.

We define the following:

The centered ratings matrix:

$$\tilde{R} = R - \hat{R}$$

The covariance of two users i and j

$$Cov(u_i, u_j) = R_i^T R_j$$

The correlation of two users i and j

$$Corr(u_i, u_j) = \frac{R_i^T R_j}{\|R_i\|_2 \|R_j\|_2} = \frac{\sum_u \tilde{r}_{ui} \tilde{r}_{uj}}{\sqrt{\sum_u \tilde{r}_{ui}^2 \sum_u \tilde{r}_{uj}^2}}$$

We use the user correlation matrix to define the similarity between users.
The neighborhood-based predictor is

$$\hat{r}_{ui}^N = (\hat{R} + b_u + b_i) + \frac{\sum_{n \in \mathcal{L}_u} Corr(u, n) \tilde{r}_{ni}}{\sum_{n \in \mathcal{L}_u} |Corr(\mathrm{u}, n)|}$$

Where $\mathcal{L}_u$ are the k most similar users to the user u.

3. Implement the method
   `def user_similarity(self, user1: int, user2: int) -> float`
   of the class `NeighborhoodRecommender` which returns the correlation of the two users.

   This function is only required to test that you calculate the correlations correctly, and you don't have to use it anywhere else unless you choose to.

4. Implement the methods of the class `NeighborhoodRecommender`.
   The prediction should be done with the 3 nearest neighbors.
   Ignore the timestamp parameter for rating predictions.
   Make sure the predicted rating is in the range $[0.5, 5]$.

Part B (30 points)

In this part we will use a regression model to predict the ratings.
The runtime of this part must not exceed 3 minutes.

We define a parameter b_u for each user as well as b_i for each item.
Additionally, we can use the timestamp in our data to add new time parameters b_d, b_n and b_w.

The rating estimate for user u, item i and timestamp t is

$$r_{uit} = \hat{R} + b_u + b_i + b_d + b_n + b_w$$

where b_u is a user-specific parameter, b_i is an item-specific parameter and b_d, b_n, b_w are timestamp parameters.

b_d – A parameter for ratings that were given in daytime (between 6am and 6pm).
b_n – A parameter for ratings that were given in the night (between 6pm and 6am).
b_w – A parameter for ratings that were given in the weekend (Friday or Saturday).

This is a least squares problem

$$\min_{b_u, b_i, b_d, b_n, b_w} \|X\beta - y\|_2^2$$

where $X \in \mathbb{R}^{|R| \times (|U|+|I|+3)}$ is the training data, $\beta \in \mathbb{R}^{|U|+|I|+3}$ is the parameter vector and $y = R - \hat{R}$ is the centered ratings vector.

To solve the least squares problem, use *np.linalg.lstsq*.

Implement the methods of the class `LSRecommender`.

```
def solve_ls(self) -> Tuple[np.ndarray, np.ndarray, np.ndarray]
```

should return a tuple of the matrices X, β, y.

Note: you need to convert the timestamp on your own to find out whether the rating was given in the day, night, or weekend (use *datetime.fromtimestamp*).

Make sure to return a rating in the range $[0.5, 5]$.

Part C (15 points)

In this part you will compete for the lowest RMSE score on the ratings_comp data (800,000 ratings).
The competition data has been split in the same way as the data used for parts A and B, 80% of the ratings for every user are in the ratings_comp file and your predictions will be evaluated against the remaining 20%.

You will be awarded some of the points for achieving better results than a predetermined predictor and the rest of the points will be given based on your ranking in the competition.

Implement the methods of the class `CompetitionRecommender`.

The maximum runtime for this part is 15 minutes.

Make sure to check that your code runs on the VM before submitting!