Recommender System

author Dae In Lee

Quick Start

All programs are tested on macOS 10.14

Requirements

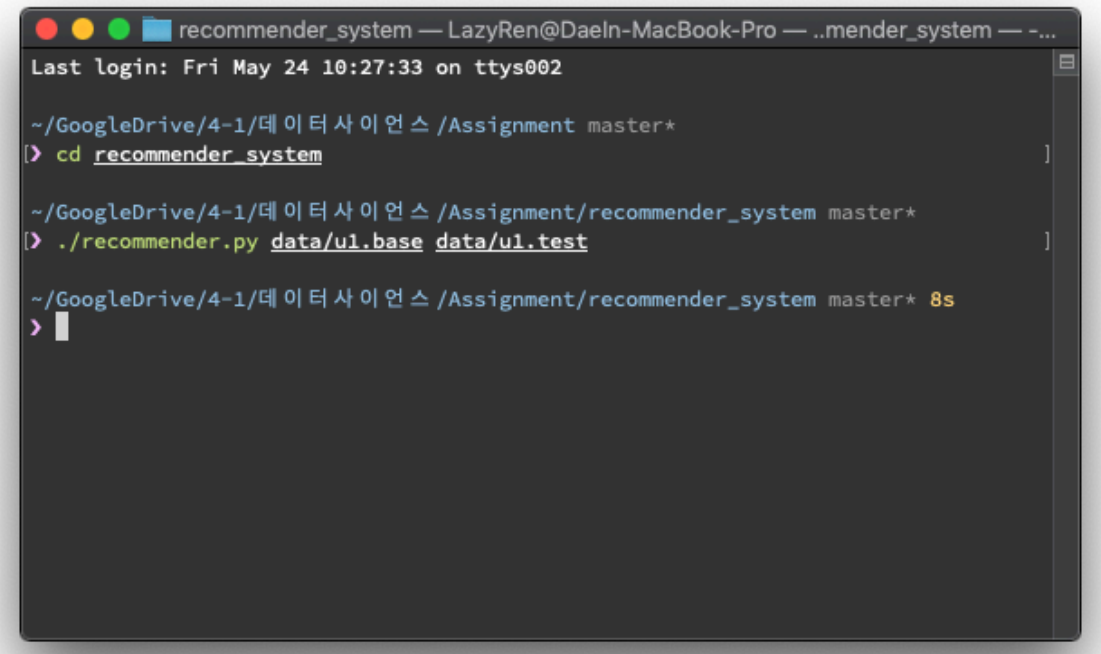
- Python3

How to Run

recommender.py

```
./recommender.py [train file] [test file]
python3 recommender.py [train file] [test file]
```

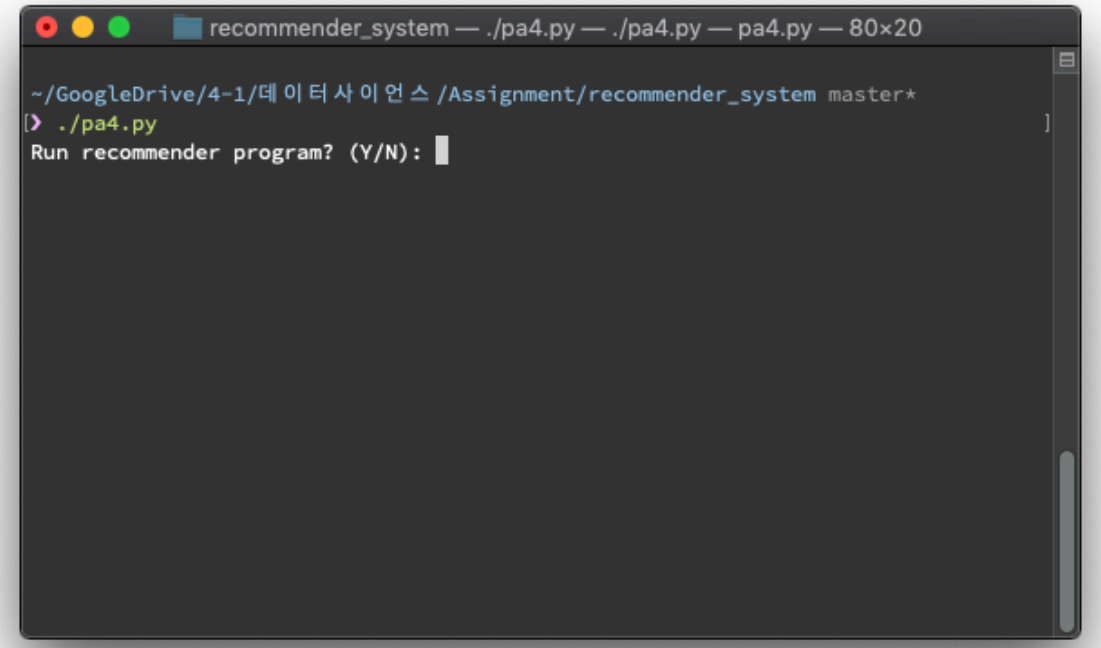
In order to execuate program using first command, you must have execution permission for `recommender.py`.
If it gives permission error, either give it an execution permission or use second command.



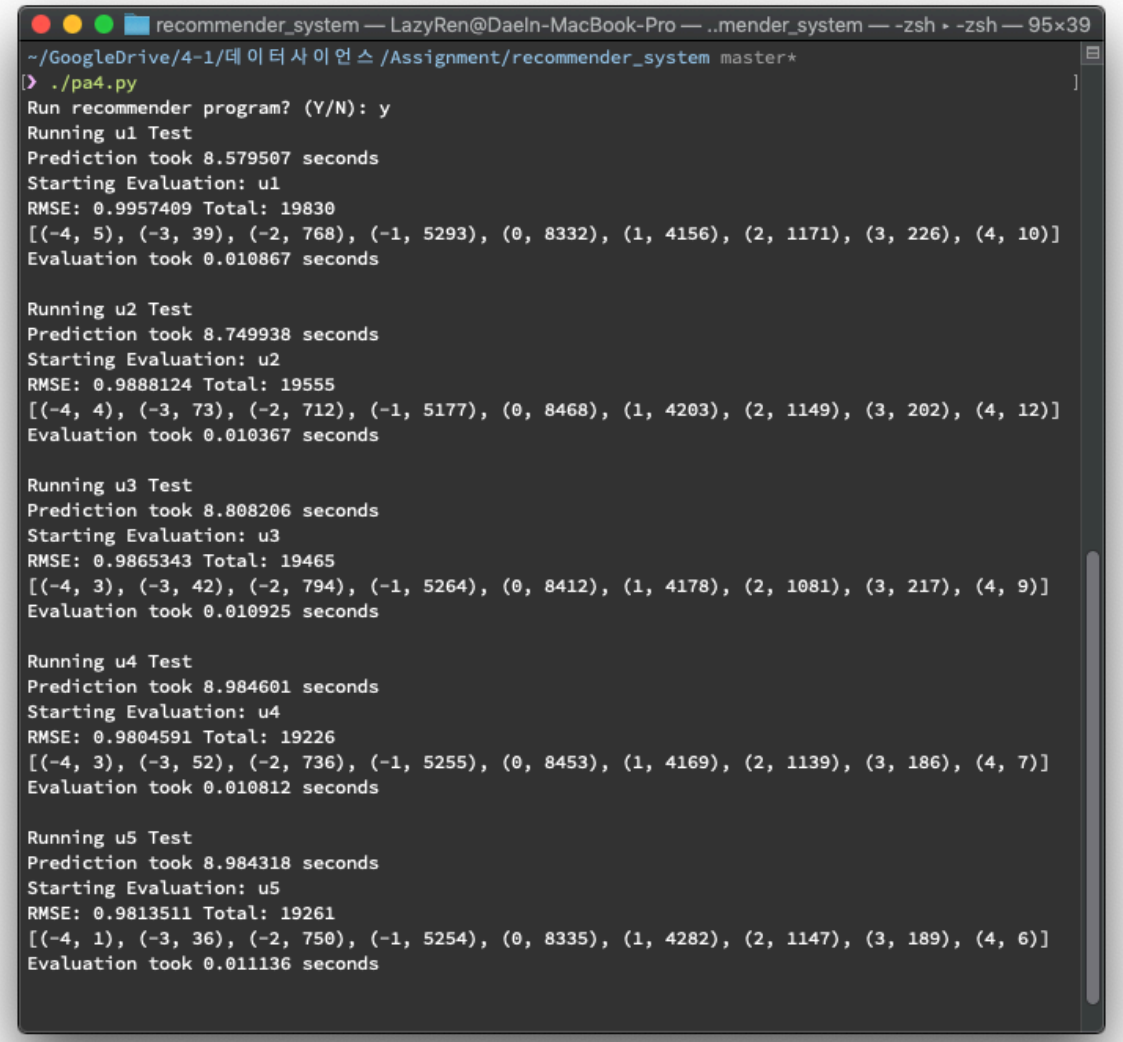
pa4.py

```
./pa4.py
python3 pa4.py
```

Running above command will ask you whether to run recommender program or not.

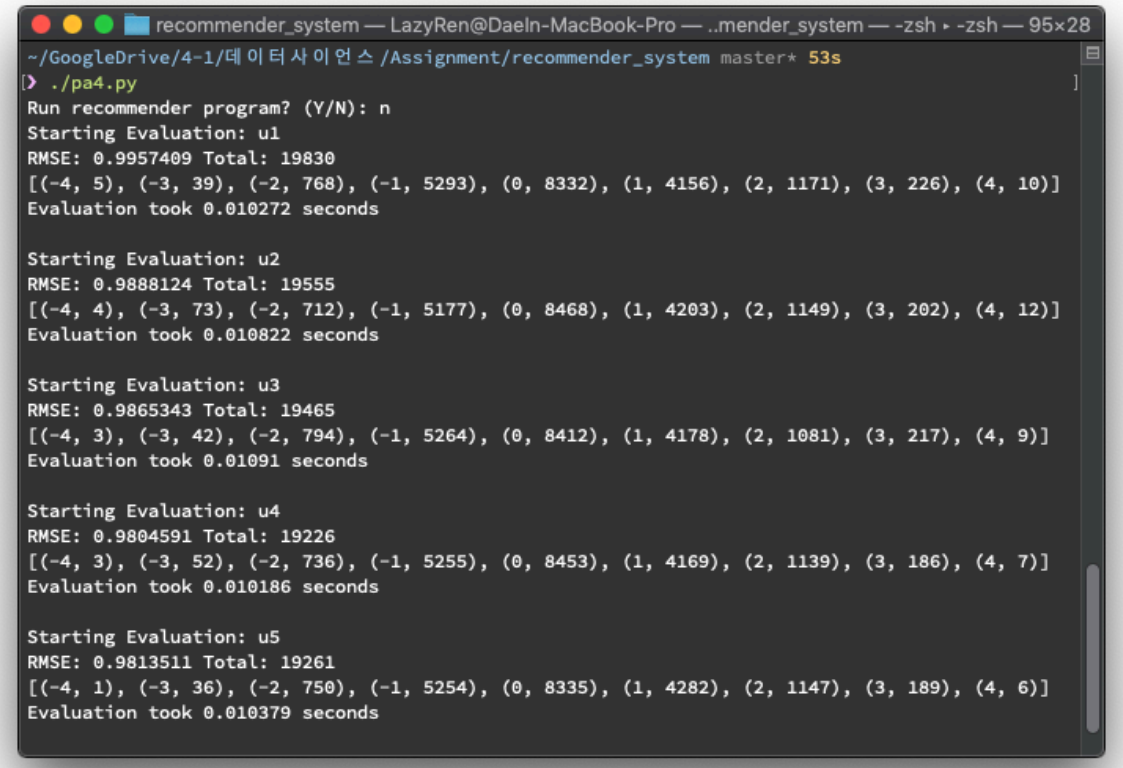


If you choose to run recommender program, `pa4.py` will automatically run `recommender.py` to train & test all u1 ~ u5.



Else, `pa4.py` will use existing u`N`.test & u`N`.base_prediction.txt files to evaluate **RMSE**.

You **MUST** have `base_prediction.txt` files ready to run in this way.



Detail of the execution, such as directory of data files, recommender program's name or which tests to run can be modified by changing below globar variables.

```
inputFileDir  = "data/"
idealFileDir  = "data/"
outputFileDir = "data/"
EXECUTABLE_NAME = "recommender.py"
testNameList   = ["u1", "u2", "u3", "u4", "u5"]
```

Implementation

loadData()

Simply read line to line from *fileName* and generate list of *[(user_id), (item_id), (rating)]*
Note that *time_stamp* has been removed when parsed. Since prediction does not use that information in any way with current implementation.

preprocessData()

Preprocess train data that has been loaded by `loadData()`.
For now, it only creates *ratingDict* for the future use.

```
ratingDict[user][movie]  # rating of moive of user (user & movie must be int)
ratingDict[user]['mean'] # mean of user's all rating
```

Time Complexity = O(2r + u)
r = number of rows in data, u = number of users

similarityMeasure()

$$\text{simil}(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}}$$

Calcurate similarity using *Pearson Correlation Coefficient*.
Function only needs `ratingDict` created by `preprocessData()`.

Time Complexity = O(u^2 * m)
u = number of users, m = number of movies
Since m in time complexity refers to commonItem of two user, in most cases function will run in O(u^2).

findNeighbors()

Calculate similarity of all user and sort them in descending order of similarity.
Return 2D list of tuple(uid, smilarity).

```
# Returned 2D list
neighbors[uid][i] == (uid2, sim) # uid's ith closest neighbor who's id is
uid2, with similarity of sim.
```

Time Complexity = O(u^2)
u = number of users

predictRating()

$$pred_{u,i} = \bar{r}_u + \frac{\sum_{v \in V} sim_{u,v}(r_{v,i} - \bar{r}_v)}{\sum_{v \in V} |sim_{u,v}|},$$

Predict rating of movie for user using KNN collaborative filtering.
Prediction use the mean of the neighbouring ratings weighted by their similarity [1].
Algorithm stops in advance if similarity reaches negative for better prediction.

Time Complexity = O(u)
u = number of users

---

1. Florent Garcin, Boi Faltings, Radu Jurca, Nadine Joswig, **Rating Aggregation in Collaborative Filtering Systems**↩