Your Name: YongKyung Oh
Your Andrew ID: yongkyuo
Your Nickname on Leaderboard: YK

# Homework 2

## 0. Statement of Assurance

1. Did you receive any help whatsoever from anyone in solving this assignment? Yes / **No**.
If you answered 'yes', give full details? (e.g."Jane explained to me what is asked in Question 3.4").

2. Did you give any help whatsoever to anyone in solving this assignment? Yes / **No**.
If you answered 'yes', give full details? (e.g. "I pointed Joe to section 2.3 to help him with Question 2").

3. Did you find or come across code that implements any part of this assignment? **Yes** / No.
If you answered 'yes', give full details? (e.g. book & page, URL & location within the page, etc)

For the Probabilistic Matrix Factorization part:

- Original Paper: https://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf
- Pytorch implementation: https://github.com/mcleonard/pmf-pytorch/blob/master/Probability%20Matrix%20Factorization.ipynb

## 1. Corpus Exploration (10)

Please perform your exploration on the training set.

### 1.1 Basic statistics (5)

| **Statistics** | |
|---|---|
| the total number of movies | 5353 |
| the total number of users | 10858 |
| the number of times any movie was rated '1' | 53852 |
| the number of times any movie was rated '3' | 260055 |
| the number of times any movie was rated '5' | 139429 |
| the average movie rating across all users and movies | 3.38 |

| For user ID **4321** | |
|---|---|
| the number of movies rated | 73 |
| the number of times the user gave a '1' rating | 4 |
| the number of times the user gave a '3' rating | 28 |
| the number of times the user gave a '5' rating | 8 |
| the average movie rating for this user | 3.15 |

| For movie ID **3** | |
|---|---|
| the number of users rating this movie | 84 |
| the number of times the user gave a '1' rating | 10 |
| the number of times the user gave a '3' rating | 29 |
| the number of times the user gave a '5' rating | 1 |
| the average rating for this movie | 2.52 |

## 1.2 Nearest Neighbors (5)

| | **Nearest Neighbors** |
|---|---|
| Top 5 NNs of user 4321 in terms of dot product similarity | 980, 551, 2586, 3760, 90 |
| Top 5 NNs of user 4321 in terms of cosine similarity | 8497, 9873, 7700, 8202, 3635 |
| Top 5 NNs of movie 3 in terms of dot product similarity | 1466, 3688, 3835, 4927, 2292 |
| Top 5 NNs of movie 3 in terms of cosine similarity | 4857, 5370, 5391, 4324, 5065 |

## 2. Rating Algorithms (50)

### 2.1 User-user similarity (10)

| Rating Method | Similarity Metric | K | RMSE | Runtime(sec) |
|---|---|---|---|---|
| Mean | Dot product | 10 | 1.0024 | 19.7750 |
| Mean | Dot product | 100 | 1.0067 | 44.0032 |
| Mean | Dot product | 500 | 1.0430 | 145.1234 |
| Mean | Cosine | 10 | 1.0630 | 21.0444 |
| Mean | Cosine | 100 | 1.0620 | 43.6756 |
| Mean | Cosine | 500 | 1.0753 | 144.7183 |
| Weighted | Cosine | 10 | 1.0627 | 20.2980 |
| Weighted | Cosine | 100 | 1.0615 | 41.3121 |
| Weighted | Cosine | 500 | 1.0740 | 140.3461 |

### 2.2 Movie-movie similarity (10)

| Rating Method | Similarity Metric | K | RMSE | Runtime(sec) |
|---|---|---|---|---|
| Mean | Dot product | 10 | 1.0213 | 53.4726 |
| Mean | Dot product | 100 | 1.0469 | 63.0150 |
| Mean | Dot product | 500 | 1.1110 | 101.0963 |
| Mean | Cosine | 10 | 1.0195 | 14.5352 |
| Mean | Cosine | 100 | 1.0644 | 22.5883 |
| Mean | Cosine | 500 | 1.1185 | 54.2922 |
| Weighted | Cosine | 10 | 1.0169 | 13.9151 |
| Weighted | Cosine | 100 | 1.0574 | 21.3708 |
| Weighted | Cosine | 500 | 1.1028 | 53.6476 |

### 2.3 Movie-rating/user-rating normalization (10)

I tried both algorithms.

2.3.1 PCC_user: User-rating normalization

| Rating Method | Similarity Metric | K | RMSE | Runtime(sec) |
|---|---|---|---|---|
| Mean | Dot product | 10 | 1.0035 | 34.0572 |
| Mean | Dot product | 100 | 1.0074 | 36.0830 |
| Mean | Dot product | 500 | 1.0437 | 44.3739 |
| Mean | Cosine | 10 | 1.0654 | 34.4780 |
| Mean | Cosine | 100 | 1.0641 | 35.7934 |
| Mean | Cosine | 500 | 1.0768 | 41.5984 |
| Weighted | Cosine | 10 | 1.0650 | 34.3364 |
| Weighted | Cosine | 100 | 1.0637 | 35.6801 |
| Weighted | Cosine | 500 | 1.0755 | 41.4333 |

2.3.2 PCC_item: Item-rating normalization

| Rating Method | Similarity Metric | K | RMSE | Runtime(sec) |
|---|---|---|---|---|
| Mean | Dot product | 10 | 1.0223 | 23.7546 |
| Mean | Dot product | 100 | 1.0469 | 27.1225 |
| Mean | Dot product | 500 | 1.1108 | 34.7822 |
| Mean | Cosine | 10 | 1.0189 | 23.4156 |
| Mean | Cosine | 100 | 1.0642 | 25.0632 |
| Mean | Cosine | 500 | 1.1183 | 31.3496 |
| Weighted | Cosine | 10 | 1.0164 | 23.4484 |
| Weighted | Cosine | 100 | 1.0573 | 24.9249 |
| Weighted | Cosine | 500 | 1.1027 | 30.9663 |

Add a detailed description of your normalization algorithm.

**Systematic Biases in CF**

- User-bias: If the system consistently favors some users over other users (not regarding the queries), we consider it a user-bias.
- Item-bias: If the system consistently favors some items over other items (not regarding the queries), we consider it an item-bias.
- To remove the user-bias (row-wise) or item-bias (column-wise), vector standardization is conducted to make mean zero and equal variance (1).

In the score date, different user shows different bias in the score. Some user may be generous or other user may be skeptical for the score. Some users get a wide range of scores, but others do not. These differences in ratings cause problems of understanding the similarity measure. Therefore, we should consider the normalized score for each user and movie for the fair comparison.

For the normalization, mean of each row (or column) is subtracted. So, I found the standard deviation of each value by row-wise (or column-wise). After that, I normalized the score value. Through this process, correlation between each element is calculated.

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}}$$

## 2.4 Matrix Factorization (20)

a. Briefly outline your optimization algorithm for PMF

$$\mathrm{E} = \frac{1}{2}\Sigma_{i=1}^{N}\Sigma_{j=1}^{M} I_{ij}\left(R_{ij} - U_i V_j^T\right)^2 + \frac{\lambda_U}{2}\Sigma_{i=1}^{N}\left||U_i|\right|^2 + \frac{\lambda_V}{2}\Sigma_{j=1}^{M}\left||V_j|\right|^2$$

Where R is training matrix with dimension N*M, U with dimension N*D and V with dimension V*D. Here D is the number of latent features. By using the I matrix, PMF only calculate the nonzero element in sparse matrix. In the training phase, we can calculate the error (or loss) using this factor. I implement the two PMF. Optimization algorithms are using gradient decent with numpy format and pytorch. To perform warm start, SVM is conducted. The result of U and V from SVM is used to initial value of iteration. It reduces the iteration time and increase the performance.

For the numpy version, I just calculated the error and target value to find the answer. In the pytorch implementation, I changed the data type into tensor and use adam optimizer. Also, to calculate efficiently, train matrix is normalized into both row-wise and column-wise.

The big idea behind this paper is that we're going to treat the latent vectors as parameters in a Bayesian model. As a reminder, Bayes theorem:

$$P(\theta \mid D) \propto P(D \mid \theta) P(\theta)$$

In our model we try to predict the rating matrix $R$ with $U$ and $V$

$$\hat{R} = U^T V$$

In our model we assume the ratings are drawn from a normal distribution with mean $\hat{R}$. What's really cool is that we can place priors on our latent features.

$$\begin{aligned} R &\sim \text{Normal}(U^T V, \sigma^2) \\ U &\sim \text{Normal}(0, \sigma_U^2) \\ V &\sim \text{Normal}(0, \sigma_V^2) \end{aligned}$$

The authors of the paper go further and build a hierachical structure for the user vectors

$$\begin{aligned} U_i &= Y_i + \frac{\sum_k^M I_{ik} W_k}{\sum_k^M I_{ik}} \\ Y &\sim \text{Normal}(0, \sigma_U^2) \\ W &\sim \text{Normal}(0, \sigma_W^2) \end{aligned}$$

With this model, we can maximize the posterior probability with respect to $U$ and $V$, or $V$, $Y$, and $W$ for the hierarchical model. In effect this is just a linear model with fancy regularization.

Reference: https://github.com/mcleonard/pmf-pytorch/blob/master/Probability%20Matrix%20Factorization.ipynb

b. Describe your stopping criterion

By repeating the iteration, the error is converged. So, I calculated the error (or loss) ratio between current step and previous step.

$$ratio_{error} = \frac{abs(err_{new} - err_{old})}{err_{old}}$$

If the error ratio is smaller than threshold, iteration stops. For the following case, I conducted grid search to find the best parameters. Also, I use the pytorch implementation on the GPU. Therefore, pmf implementation saves much time compare to CPU only setup. In the following case, I set up the threshold as 1e-7 and iterate 5000.

Also, to find the impact of each parameter, I conducted multiple experiments by varying the number of latent factors and lambda (regularization parameter)

| Num of Latent Factors | Lambda | RMSE | Runtime(sec)* | Num of Iterations |
|---|---|---|---|---|
| 2 | 0.1 | 0.9580 | 254.9333 | 3601 |
| 5 | 0.1 | 0.9427 | 230.8865 | 4601 |
| 10 | 0.1 | 0.9716 | 272.5029 | 4051 |
| 20 | 0.1 | 1.0826 | 346.6637 | 5000 |
| 50 | 0.1 | 1.2850 | 533.5090 | 5000 |

| Num of Latent Factors | Lambda | RMSE | Runtime(sec)* | Num of Iterations |
|---|---|---|---|---|
| 2 | 0.5 | 1.1528 | 110.7870 | 3551 |
| 5 | 0.5 | 1.2000 | 377.3789 | 4301 |
| 10 | 0.5 | 0.9448 | 314.3787 | 4751 |
| 20 | 0.5 | 1.0043 | 361.7273 | 4601 |
| 50 | 0.5 | 1.1458 | 531.0490 | 5000 |

| Num of Latent Factors | Lambda | RMSE | Runtime(sec)* | Num of Iterations |
|---|---|---|---|---|
| 10 | 0.1 | 0.9716 | 272.5029 | 4051 |
| 10 | 0.2 | 0.9570 | 343.7067 | 5000 |
| 10 | 0.3 | 0.9497 | 308.4689 | 3901 |
| 10 | 0.4 | 0.9470 | 346.6788 | 5000 |
| 10 | 0.5 | 0.9448 | 314.3787 | 4751 |
| 10 | 0.6 | 1.3016 | 279.8658 | 2951 |
| 10 | 0.7 | 1.1459 | 350.0704 | 5000 |
| 10 | 0.8 | 1.5220 | 350.5470 | 5000 |
| 10 | 0.9 | 1.1353 | 293.5784 | 3251 |

| Num of Latent Factors | Lambda | RMSE | Runtime(sec)* | Num of Iterations |
|---|---|---|---|---|
| 10 | 0.001 | 1.0337 | 343.5474 | 5000 |
| 10 | 0.01 | 1.0184 | 505.7108 | 4551 |
| 10 | 0.1 | 0.9716 | 272.5029 | 4051 |
| 30 | 0.001 | 1.3027 | 504.5471 | 5000 |
| 30 | 0.01 | 1.2622 | 475.1798 | 5000 |
| 30 | 0.1 | 1.1689 | 330.5048 | 5000 |
| 50 | 0.001 | | | |
| 50 | 0.01 | 1.4532 | 507.0091 | 5000 |
| 50 | 0.1 | 1.4056 | 521.4932 | 5000 |

Also, according to the original paper, regularization parameter lambda U is larger than lambda V. However, in the experiments, it is hard to find the optimal parameters. So, I use the same value for both. Additional to that, I found the ensemble of weak model is much better than original one. So, I selected best 3 out of parameter grid search and combine the matrix features.

## 3. Analysis of results (15)

Discuss the complete set of experimental results, comparing the algorithms to each other. Discuss your observations about the various algorithms, i.e., differences in how they performed, what worked well and didn't, patterns/trends you observed across the set of experiments, etc. Try to explain why certain algorithms or approaches behaved the way they did.

**Memory-Based Collaborative Filtering** approaches can be divided into two main sections: user-user filtering and item-item filtering.

- User-User Collaborative Filtering: “Users who are similar to you also liked …”

-Item-Item Collaborative Filtering: “Users who liked this item also liked …"

**Similarity between users and items:** Similarity between user-user and item-item are used. In the case of user collaborative filtering, system recommend the item based on the rating of similar user. Compare to that, item collaborative filtering system recommend the item which is similar characteristic between other items. Generally, it appears that user-item similarity matrices should be more spaced than item-user matrix rows. This is because an item is more likely to be evaluated by many users, and it is less likely that one user has rated most items. However, overall performances (RMSE) seem similar.

**Normalization for the user-user / item-item => PCC(Pearson Correlation Coefficient):**
The biggest reason for data normalization is to avoid impacting the system by user and item bias. If we normalize our ratings, if we subtract the average rating of the corresponding user from each rating, the lower rating changes to negative and the higher rating to positive. If we choose Cosine measure to find a similar user, we find that users with views that are contrary to the movies they have seen in common will have vectors in almost the opposite direction, and can be considered as far away as possible.

**Matrix Factorization:** As explained in the previous part, the biggest problem is sparsity of the original matrix. Therefore, when we deal with the sparse information, we need to capture potential information and similarity. We can capture the hidden information using probabilistic matrix factorization. This can help to increase the system’s performance.

Collaborative Filtering algorithms are most commonly used in recommended systems, and CF algorithms are now problematic in large data sets and gaps in the rating matrix. The roles of various Matrix Factorization models have been studied to address the Collaborative Filtering challenges.

## 4. The software implementation (5)

Add detailed descriptions about software implementation & data preprocessing, including:

1. A description of what you did to preprocess the dataset to make your implementations easier or more efficient.

For the efficiency, I change the original data into sparse matrix format. Through this step, I can manage high-dimension sparse matrix. Also, for convenience, I save the user and movie index into dictionary. Therefore, ratings and predictions saved in the 2D dictionary structure. Compare to other approach (ex. DataFrame), it is much faster to deal with the data.

(I use the util.py code, which is provided)

2. A description of major data structures (if any); any programming tools or libraries that you used; (requirement.txt is attached)

main.py : Preprocess and main pipeline for the prediction

function.py: collaborative filtering prediction code for user and item. Also, matrix factorization code written in numpy-based approach and pytorch implementation.

util.py: File Input/Output Support

```
$ python main.py -h
usage: main.py [-h] --Method METHOD --File FILE [--Similarity SIMILARITY]
               [--Mean MEAN] [--K K] [--Latent LATENT] [--Lambda LAMBDA]

Python module for Collaborative Filtering

optional arguments:
  -h, --help            show this help message and exit
  --Method METHOD       CF methods: user | item | pcc_user | pcc_item | pmf_gd
                        | pmf_torch
  --File FILE           Select File: dev | test
  --Similarity SIMILARITY
                        Similarity measure: dot | cosine
  --Mean MEAN           Mean calculation: mean | weight
  --K K                 KNN parameter (default: 5)
  --Latent LATENT       PMF latent parameter (default: 10)
  --Lambda LAMBDA       PMF lambda parameter (default: 0.2)
```

3. Strengths and weaknesses of your design, and any problems that your system encountered;

It is easy to setup parameters using parser framework. Also, all functions are built in module, so it is easy to implement other cases. However, there are bottleneck in calculating matrix (especially, similarity value.) So, if it is improved, the code will perform better.

* Test Environment #1

Ubuntu 18.04

vCpu * 2 / Ram: 4GB / AWS EC2 t2.medium

* Test Environment #1

Ubuntu 18.04

AMD Ryzen™ Threadripper 2950X / Ram: 128GB / Titan RTX 24GB