Collaborative Filtering for Implicit Feedback Datasets

Y. Hu, Y. Koren and C. Volinsky, 2008, IEEE

24.01.10 (수)

발표자 : 윤민석

ISysE KAIST
Industrial & Systems Engineering

Contents

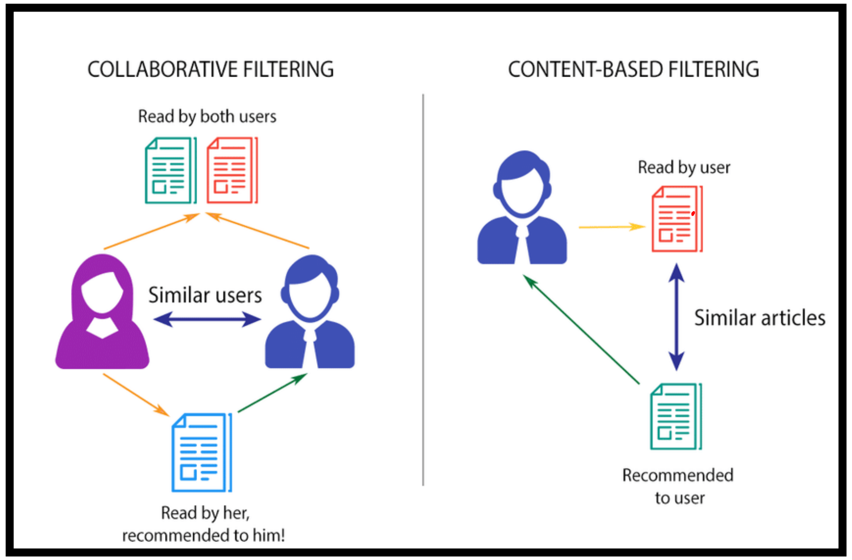
Paper Review

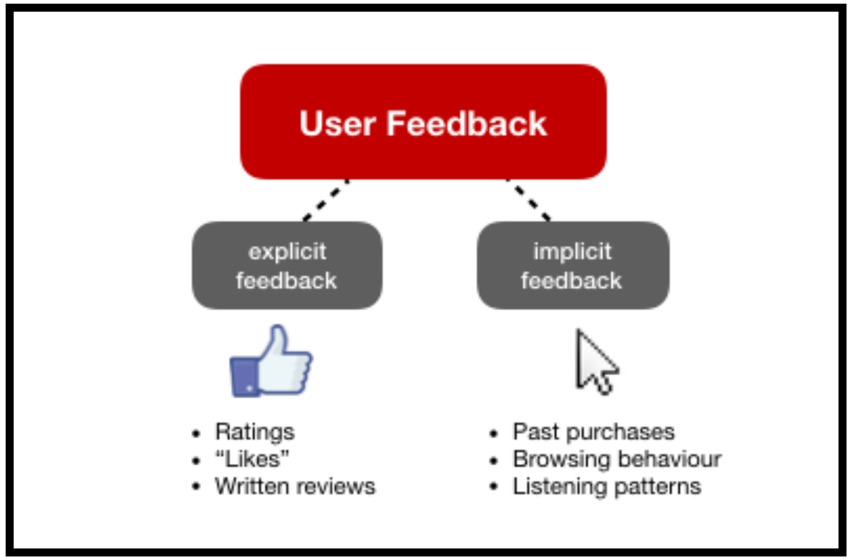
- Abstract
  - We identify unique **properties of implicit feedback datasets**.
  - We propose treating the data as indication of positive and negative **preference** associated with vastly varying **confidence levels**.
  - We also suggest **a scalable optimization** procedure, which scales linearly with the data size.
  - We offer a novel way to give **explanations to recommendations** given by this factor model.

- Introduction
  - Strategies for recommender system (Content based vs Collaborative Filtering)
  - Types of data (Explicit vs Implicit)

Paper Review

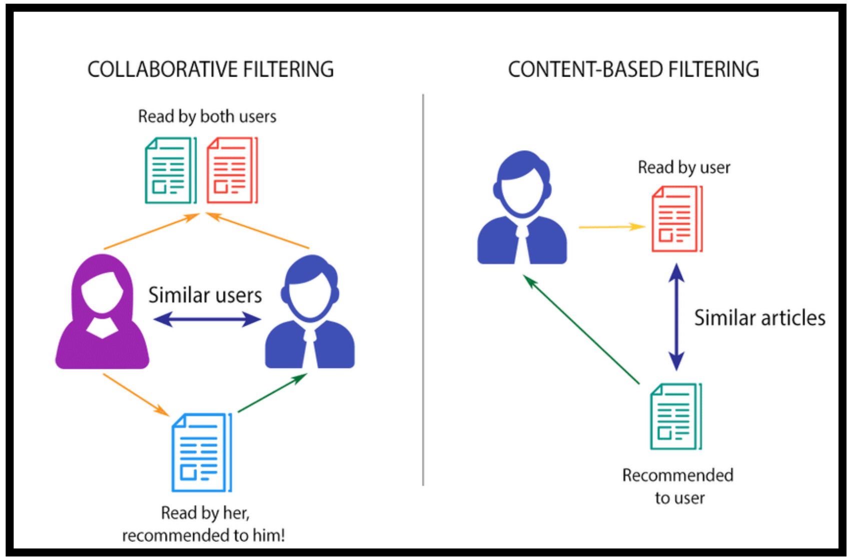
- Introduction
  - Strategies for recommender system
    - Content based
      - Creates a profile for each user or product to characterize its nature. (Ex. Genre, Actors, Directors)
    - Collaborative Filtering
      - CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations
  - Types of data
    - Explicit (Ex. Ratings, "Likes")
    - Implicit (Ex. Purchase records)

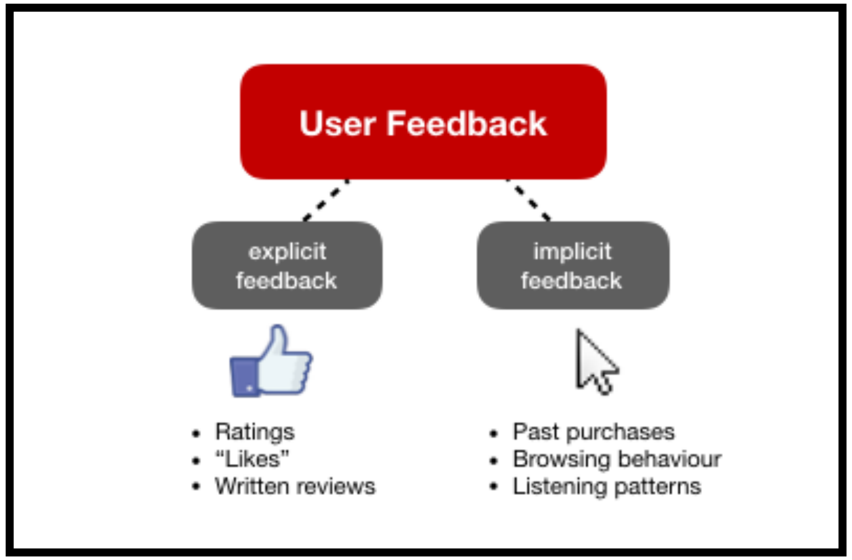




Paper Review

- Introduction
  - Strategies for recommender system
    - Content based
      - Creates a profile for each user or product to characterize its nature. (Ex. Genre, Actors, Directors)
    - Collaborative Filtering
      - CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations
  - Types of data
    - Explicit (Ex. Ratings, "Likes")
    - Implicit (Ex. Purchase records)





Paper Review

- Introduction
  - Motivation
    - However, content-based strategies require gathering **external information that might not be available or easy to collect**.
    - Explicit feedback is not **always available**.
    - In many practical situations recommender systems need to be centered on **implicit feedback**.

-> Use <u>Collaborative Filtering & Implicit datasets</u> for real world

Paper Review

- Introduction
  - Prime characteristics of Implicit datasets

    ① No negative feedback
    - It is crucial to address also the **missing data**, which is where **most negative feedback is expected to be found.**

    ② Implicit feedback is inherently noisy
    - Ex) We may view purchase behavior for an individual, but this does **not** necessarily indicate **a positive view of the product**.

    ③ The numerical value of explicit feedback indicates preferences, wheareas the numerical value of implicit feedback indicates confidence.
    - A larger value is **not** indicating a **higher preference**.
    - The numerical value of the feedback is **definitely useful**, as it tells us about the **confidence** that we have in a certain observation.

    ④ Evaluation of implicit feedback recommender requires appropriate measures

Paper Review

- Preliminaries
  - For **explicit feedback datasets**, those values would be ratings that indicate the preference by user u of item i, where high values mean stronger preference.
    - Explicit ratings are typically unknown for the vast majority of user-item pairs, hence applicable algorithms work with the relatively few known ratings **while ignoring the missing ones**.
  - For **implicit feedback datasets**, those values would indicate observations for user actions.
    - it would be natural **to assign values to all r_{ui} variables**.

-> Emphasize that implicit data should also focus on missing values

Paper Review

- Previous work (CF)
  - Neighborhood models
    - Central to most item-oriented approaches is a **similarity measure** between items, where s_{ij} denotes the similarity of i and j.
    - Our goal is to **predict r_{ui} the unobserved value** by user u for item i.
    - The predicted value of r_{ui} is taken as a weighted average of the ratings for neighboring items.

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i;u)} s_{ij}} \quad (1)$$

  - Latent factor models
    - A typical model associates each user u with a **user-factors vector** $x_u \in R^f$, and each item i with an item-factors vector $y_i \in R^f$.
    - The prediction is done by taking an inner product, i.e., $r_{ui} = \boldsymbol{x_u^T y_i}$.

$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \quad (2)$$

Paper Review

- Previous work (CF)
  - Neighborhood models
    - Central to most item-oriented approaches is a **similarity measure** between items, where s_{ij} denotes the similarity of i and j.
    - Our goal is to **predict r_{ui} the unobserved value** by user u for item i.
    - The predicted value of r_{ui} is taken as a weighted average of the ratings for neighboring items.

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i;u)} s_{ij}} \quad (1)$$

  - Latent factor models
    - A typical model associates each user u with a **user-factors vector** $x_u \in R^f$, and each item i with an item-factors vector $y_i \in R^f$.
    - The prediction is done by taking an inner product, i.e., r_{ui} **=** $\boldsymbol{x_u^T y_i}$.

$$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \quad (2)$$

-> The results tend to be consistently superior to those achieved by neighborhood models.

Paper Review

- Our model
  - Formalize the notions (Preference, Confidence)
  - Cost function
  - Optimization Techniques
  - Trick for reducing time complexity
  - Predict Preference

$$\boxed{\hat{p}_{ui} = y_i^T x_u}$$

Paper Review

- Our model
  - Formalize the notions (Preference, Confidence)
    - Preference
      - Consuming an item can also be the result of factors different from preferring it.

$$\boxed{p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}}$$

    - Confidence
      - We will have different confidence levels also among items that are indicated to be preferred by the user.
      - In general, as r_{ui} grows, we have a stronger indication that the user indeed likes the item.

$$c_{ui} = 1 + \alpha r_{ui}$$

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon).$$

Paper Review

- Our model
  - Cost function
    - Our goal is to **find a vector $x_u \in \mathbf{R}^f$ for each user u, and a vector $y_i \in \mathbf{R}^f$** for each item i that will factor user preferences.
    - In other words, preferences are assumed to be the inner products: $p_{ui} = x_u^T y_i$.

$$\min_{x_\star, y_\star} \sum_{u,i} \underbrace{c_{ui}(p_{ui} - x_u^T y_i)^2}_{\text{-> SSE}} + \underbrace{\lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)}_{\text{-> Regularizing}}$$

Paper Review

- Our model
  - Optimization Techniques
    - For typical datasets m · n can easily reach a few billion. This huge number of terms **prevents** most direct optimization techniques such as **stochastic gradient descent**, which was widely used for explicit feedback datasets.
    - Observe that when either the user-factors or the item- factors are **fixed**, the cost function becomes **quadratic** so its global minimum can be readily computed.

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - \underline{x_u^T y_i})^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

Paper Review

- Our model
  - Trick for reducing time complexity
    - A **computational bottleneck** here is computing $Y^T C_u Y$, whose naive calculation will require time O(f^2 n) (for each of the m users).
    - Use $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y.$

① $x_u = \underline{(Y^T C^u Y + \lambda I)}^{-1} Y^T C^u p(u)$

-> Computational Bottleneck

② So Use $Y^T C^u Y = Y^T Y + Y^T \underline{(C^u - I)} Y.$

-> Only non-zero elements(n_u)

$n_u << n$

① m * {O($f^2 n$) + matrix inversion time}
≅ m * O($f^2 n$ + f^3)

② O($f^2 N + f^3 m$)
N: overall number of non-zero observations

-> ② Is better and Scalable

Paper Review

- Explaining Recommendations
  - A good recommendation should be accompanied with **an explanation**, which is a short description to why a specific product was recommended to the user.
  - Use concepts of Neighborhood methods

$$\boxed{\begin{aligned} \hat{p}_{ui} &= y_i^T x_u \\ x_u &= (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \end{aligned}} \Rightarrow \hat{p}_{ui} = y_i^T \underline{(Y^T C^u Y + \lambda I)^{-1}} Y^T C^u p(u)$$

$$\Downarrow$$

D X D Matrix

D: dimension for latent factor

$$\hat{p}_{ui} = \sum_{j: r_{uj} > 0} s_{ij}^u c_{uj} \Leftarrow \boxed{\begin{aligned} &f \times f \text{ matrix } (Y^T C^u Y + \lambda I)^{-1} \text{ as } W^u \\ &s_{ij}^u = y_i^T W^u y_j \end{aligned}}$$

Paper Review

- Explaining Recommendations
  - A good recommendation should be accompanied with **an explanation**, which is a short description to why a specific product was recommended to the user.
  - Use concepts of Neighborhood methods

Perspective of user u, we can get each term of item which is associated with p_{ui}.

$$\hat{p}_{ui} = \sum_{j:r_{uj}>0} s_{ij}^{u} c_{uj} = \underline{s_{i1} * c_{u1}} + \underline{s_{i2} * c_{u2}} + \underline{s_{i3} * c_{u3}} + \underline{s_{i4} * c_{u4}} + \dots$$

By comparing the values of each term, the effect of item j on the predicted p_{ui} value can be quantified.

-> Explaining Recommendations

Paper Review

- Experiment study
  - Define Task
  - Data description & Preprocessing
  - Evaluating methodology
  - Evaluating results

Paper Review

- Experiment study
  - Define Task
    - Our system is **trained using the recent 4 weeks** of data in order to generate **predictions of what users will watch in the ensuing week**.
  - Data description
    - 300,000 set top boxes, 17,000 unique programs
    - **No personally identifiable information** was collected in connection with this research.
  - Data Preprocessing
    - We remove the **"easy" predictions** from the test set corresponding to the shows that had been watched by that user during the training period.
    - We toggle to zero all entries with $r_{ui}^{t} < 0.5$, as watching less than half of a program is not a strong indication that a user likes the program.
    - We employ the **log scaling** scheme
    - Consider Momentum effect

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon).$$

$$\frac{e^{-(at-b)}}{1+e^{-(at-b)}}$$

$$\frac{e^{-(2\cdot1-6)}}{1+e^{-(2\cdot1-6)}} = 0.98201379$$

$$\frac{e^{-(2\cdot2-6)}}{1+e^{-(2\cdot2-6)}} = 0.880797078$$

$$\frac{e^{-(2\cdot3-6)}}{1+e^{-(2\cdot3-6)}} = 0.5$$

Paper Review

- Experiment study
  - Evaluating methodology
    - $\mathrm{rank}_{\mathrm{ui}}$ = 0% would mean that program i is predicted to be the most desirable for user u
    - We opted for using **percentile-ranks** rather than absolute ranks in order to make our discussion general and independent of the number of programs.

$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}}$$

: Do the actual view records and Rank results match?

- **Lower values of rank_bar are more desirable**.

Paper Review

- Experiment study
  - Evaluating results
    - We recommend working with the highest number of factors feasible within computational limitations.

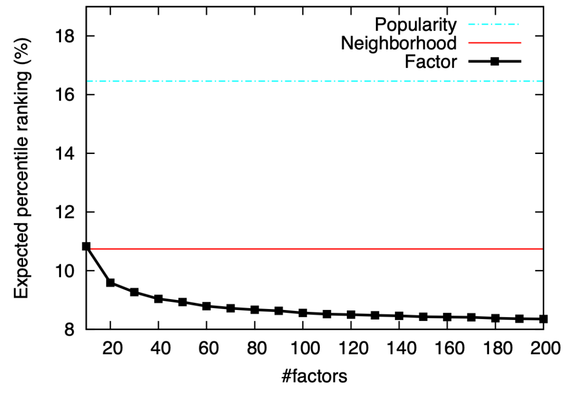


Figure 1. Comparing factor model with popularity ranking and neighborhood model.

$$\min_{x_\star, y_\star} \sum_{u,i} (r_{ui} - x_u^T y_i)^2 + \lambda_1 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{9}$$

$$\min_{x_\star, y_\star} \sum_{u,i} (p_{ui} - x_u^T y_i)^2 + \lambda_2 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{10}$$

$$<$$

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

Paper Review

- Experiment study
  - Evaluating results
    - **Predicting rewatching a program** is much easier than predicting a first time view of a program.
    - It becomes much **easier to predict popular programs**, while it is increasingly difficult to predict watching a non popular show.
    - This was somewhat unexpected, as our experience with explicit feedback datasets was that as we gather more information on users, prediction quality significantly increases.

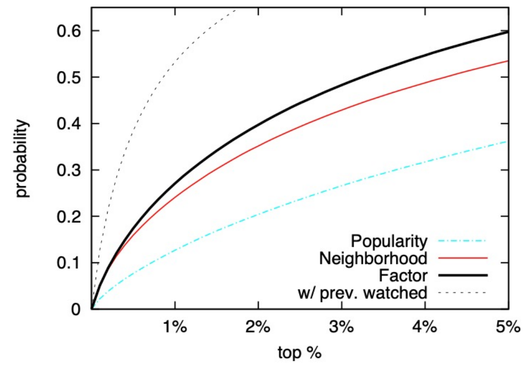


Figure 2. Cumulative distribution function of the probability that a show watched in the test set falls within top x% of recommended shows.

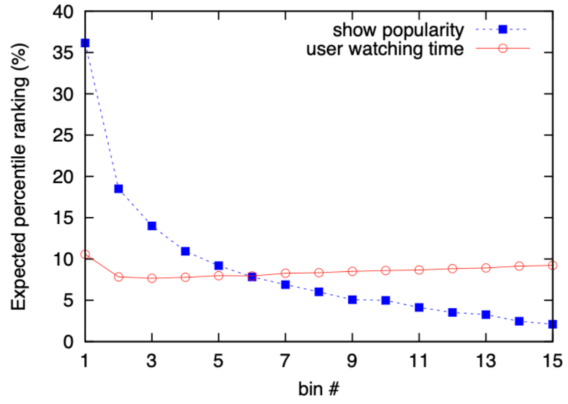
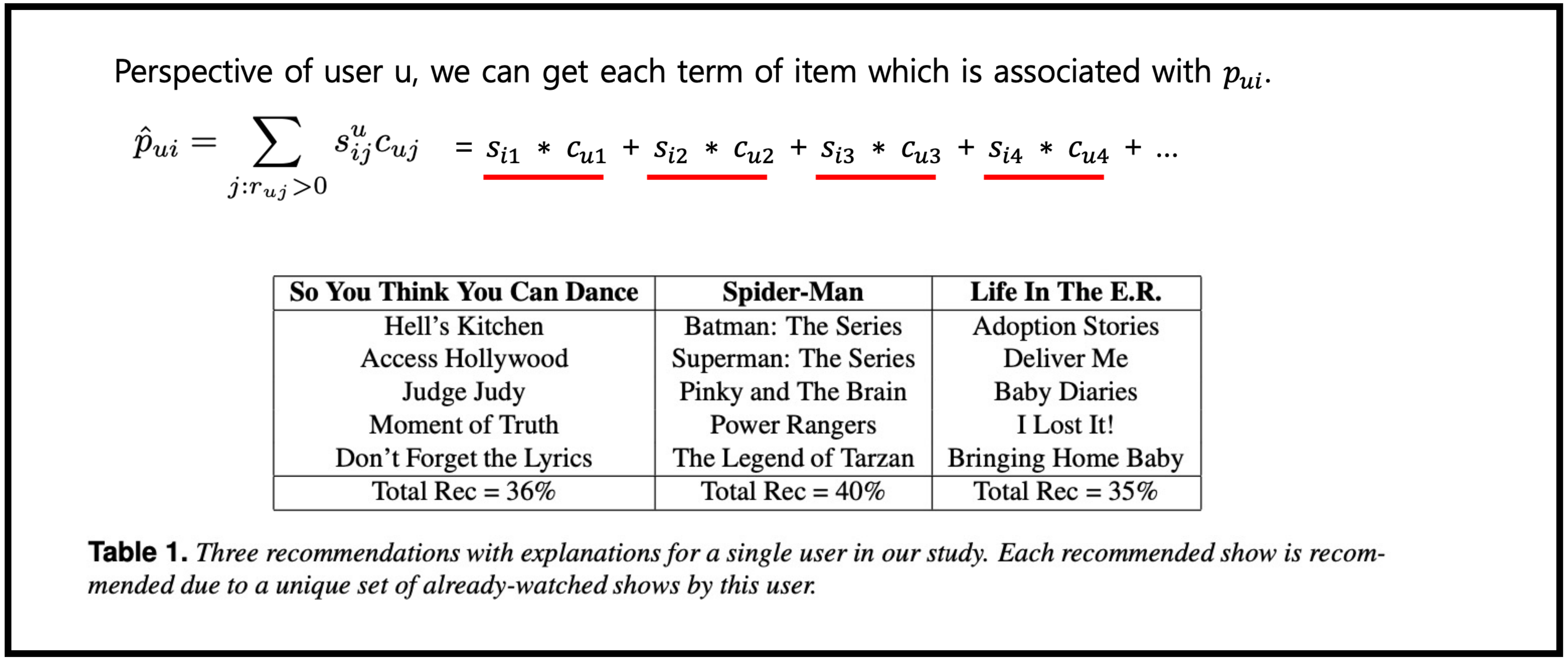


Figure 3. Analyzing the performance of the factor model by segregating users/shows based on different criteria.

Paper Review

- Experiment study
  - Evaluating results
    - These **common-sense explanations** help the user understand why certain shows are recommended and are similar to explanations returned by neighbor methods.



Perspective of user u, we can get each term of item which is associated with p_{ui}.

$$\hat{p}_{ui} = \sum_{j:r_{uj}>0} s^u_{ij} c_{uj} = \underline{s_{i1} * c_{u1}} + \underline{s_{i2} * c_{u2}} + \underline{s_{i3} * c_{u3}} + \underline{s_{i4} * c_{u4}} + \dots$$

So You Think You Can Dance	Spider-Man	Life In The E.R.
Hell's Kitchen	Batman: The Series	Adoption Stories
Access Hollywood	Superman: The Series	Deliver Me
Judge Judy	Pinky and The Brain	Baby Diaries
Moment of Truth	Power Rangers	I Lost It!
Don't Forget the Lyrics	The Legend of Tarzan	Bringing Home Baby
Total Rec = 36%	Total Rec = 40%	Total Rec = 35%

Table 1. *Three recommendations with explanations for a single user in our study. Each recommended show is recommended due to a unique set of already-watched shows by this user.*

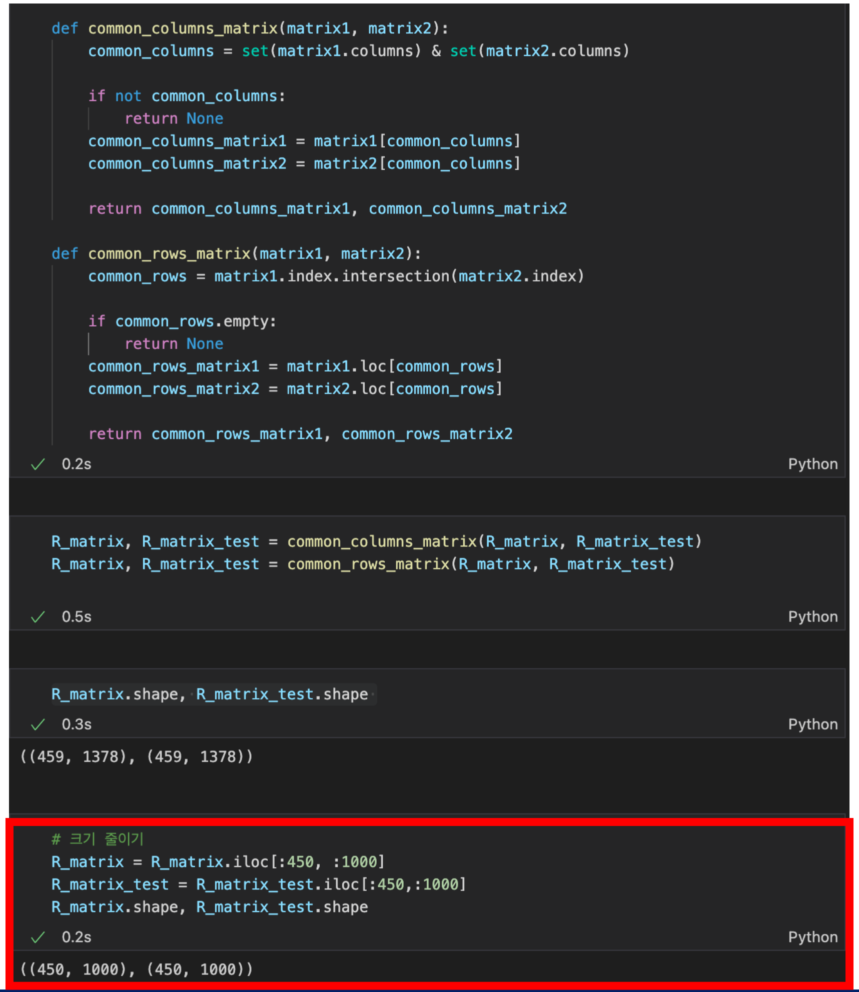
Paper Review

- Discussion
  - For Implicit feedback datasets
    - Preference – Confidence partition
    - Taking all user – item values as an input
    - Exploit the algebraic structure for scalability Issue
  - Explaining Recommendations
  - Future works
    - A more careful analysis would split **those zero values into different confidence levels**, perhaps based on availability of the item.
  - Limitation
    - Setup is designed to evaluate how well a model can predict future **user behavior**.

Code Review

Data Description → Our Model → Explaining Recommendations → Experiment

- Data Description
  - MovieLens 100K Dataset (Rating)



```python
def common_columns_matrix(matrix1, matrix2):
    common_columns = set(matrix1.columns) & set(matrix2.columns)

    if not common_columns:
        return None
    common_columns_matrix1 = matrix1[common_columns]
    common_columns_matrix2 = matrix2[common_columns]

    return common_columns_matrix1, common_columns_matrix2

def common_rows_matrix(matrix1, matrix2):
    common_rows = matrix1.index.intersection(matrix2.index)

    if common_rows.empty:
        return None
    common_rows_matrix1 = matrix1.loc[common_rows]
    common_rows_matrix2 = matrix2.loc[common_rows]

    return common_rows_matrix1, common_rows_matrix2
```
✓ 0.2s Python

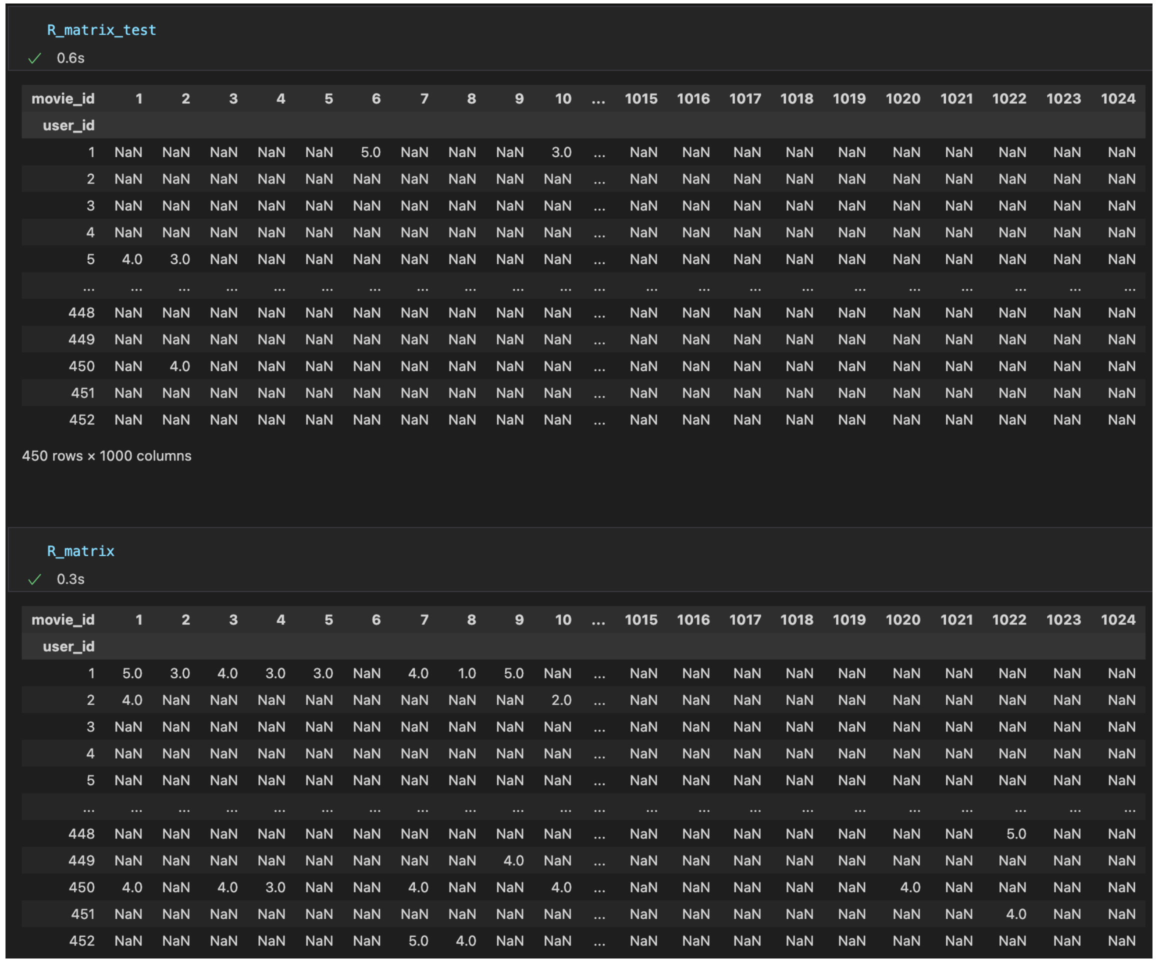
```python
R_matrix, R_matrix_test = common_columns_matrix(R_matrix, R_matrix_test)
R_matrix, R_matrix_test = common_rows_matrix(R_matrix, R_matrix_test)
```
✓ 0.5s Python

```python
R_matrix.shape, R_matrix_test.shape
```
✓ 0.3s Python

```
((459, 1378), (459, 1378))
```

```python
# 크기 줄이기
R_matrix = R_matrix.iloc[:450, :1000]
R_matrix_test = R_matrix_test.iloc[:450,:1000]
R_matrix.shape, R_matrix_test.shape
```
✓ 0.2s Python

```
((450, 1000), (450, 1000))
```



```python
R_matrix_test
```
✓ 0.6s

movie_id	1	2	3	4	5	6	7	8	9	10	...	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024
user_id																					
1	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
448	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
449	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
450	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
451	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
452	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

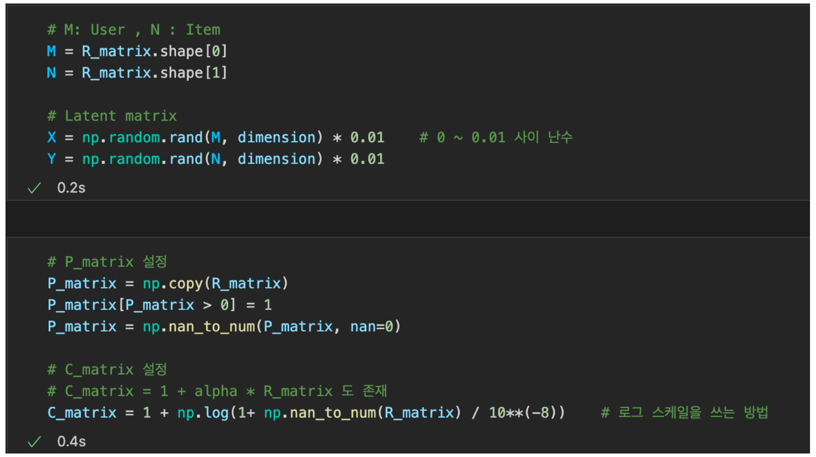
450 rows × 1000 columns

```python
R_matrix
```
✓ 0.3s

movie_id	1	2	3	4	5	6	7	8	9	10	...	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024
user_id																					
1	5.0	3.0	4.0	3.0	3.0	NaN	4.0	1.0	5.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
448	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN
449	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
450	4.0	NaN	4.0	3.0	NaN	NaN	4.0	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN
451	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN
452	NaN	NaN	NaN	NaN	NaN	NaN	5.0	4.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Code Review

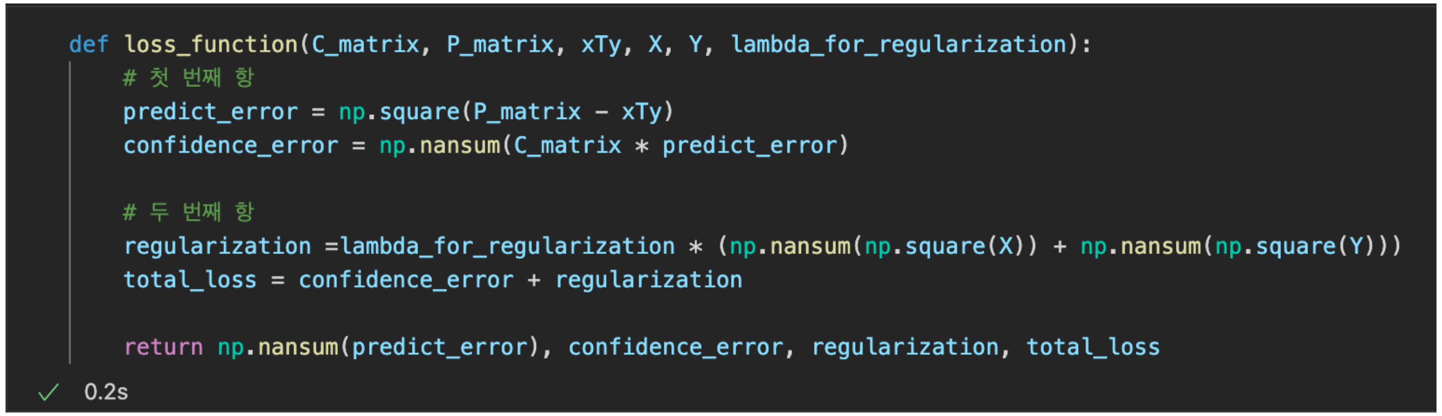
- Our model
  - Preference matrix & Confidence matrix



```
# M: User , N : Item
M = R_matrix.shape[0]
N = R_matrix.shape[1]

# Latent matrix
X = np.random.rand(M, dimension) * 0.01    # 0 ~ 0.01 사이 난수
Y = np.random.rand(N, dimension) * 0.01
✓ 0.2s
```

```
# P_matrix 설정
P_matrix = np.copy(R_matrix)
P_matrix[P_matrix > 0] = 1
P_matrix = np.nan_to_num(P_matrix, nan=0)

# C_matrix 설정
# C_matrix = 1 + alpha * R_matrix 도 존재
C_matrix = 1 + np.log(1+ np.nan_to_num(R_matrix) / 10**(-8))    # 로그 스케일을 쓰는 방법
✓ 0.4s
```

  - Cost Function

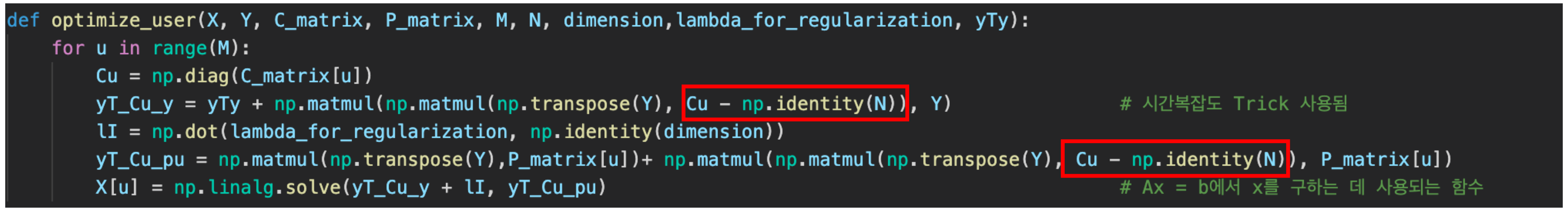


```
def loss_function(C_matrix, P_matrix, xTy, X, Y, lambda_for_regularization):
    # 첫 번째 항
    predict_error = np.square(P_matrix - xTy)
    confidence_error = np.nansum(C_matrix * predict_error)

    # 두 번째 항
    regularization =lambda_for_regularization * (np.nansum(np.square(X)) + np.nansum(np.square(Y)))
    total_loss = confidence_error + regularization

    return np.nansum(predict_error), confidence_error, regularization, total_loss
✓ 0.2s
```

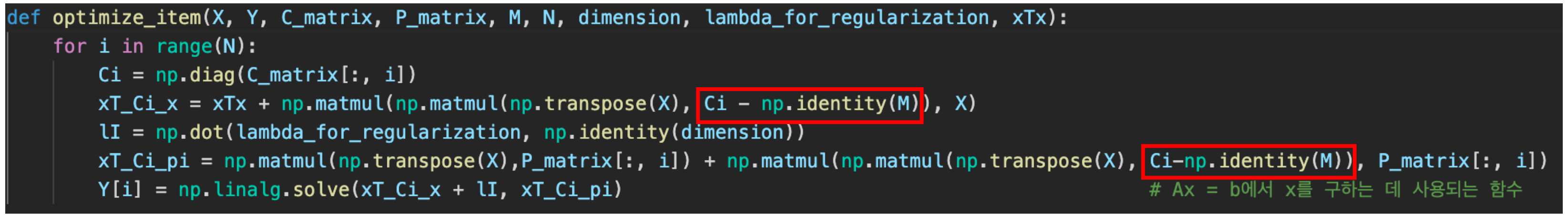
$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

Code Review

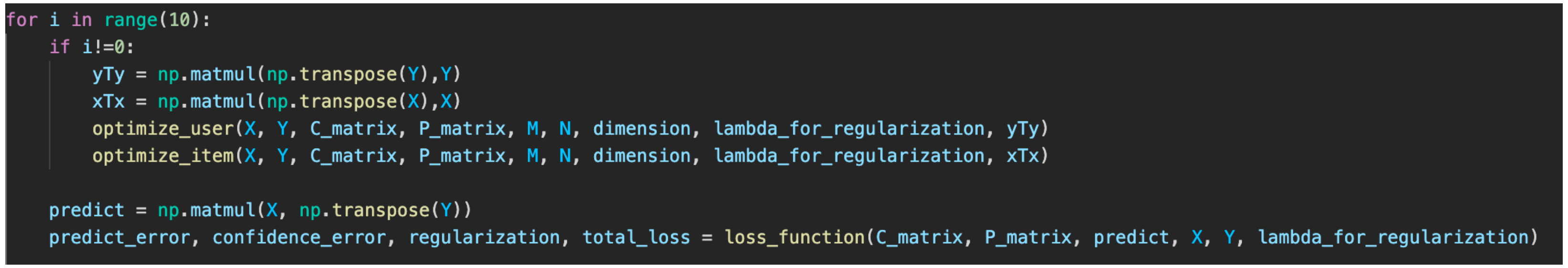
- Our model
  - Optimization



```python
def optimize_user(X, Y, C_matrix, P_matrix, M, N, dimension,lambda_for_regularization, yTy):
    for u in range(M):
        Cu = np.diag(C_matrix[u])
        yT_Cu_y = yTy + np.matmul(np.matmul(np.transpose(Y), Cu - np.identity(N)), Y)              # 시간복잡도 Trick 사용됨
        lI = np.dot(lambda_for_regularization, np.identity(dimension))
        yT_Cu_pu = np.matmul(np.transpose(Y),P_matrix[u])+ np.matmul(np.matmul(np.transpose(Y), Cu - np.identity(N)), P_matrix[u])
        X[u] = np.linalg.solve(yT_Cu_y + lI, yT_Cu_pu)                                              # Ax = b에서 x를 구하는 데 사용되는 함수
```



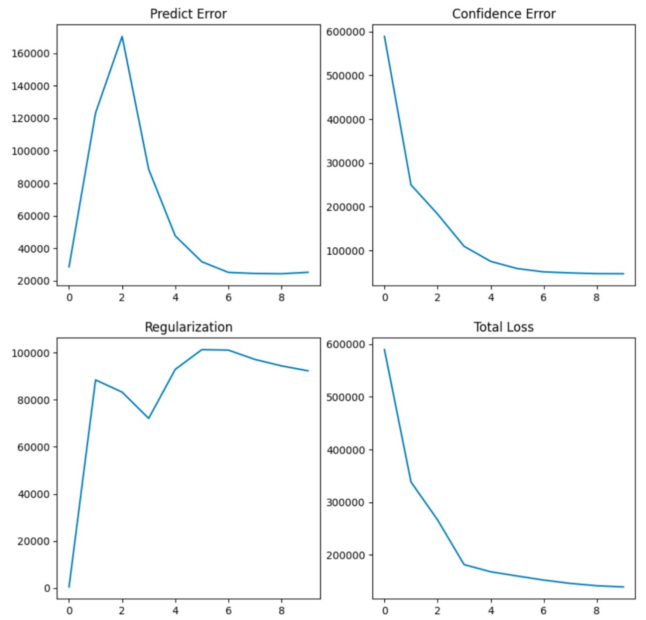
```python
def optimize_item(X, Y, C_matrix, P_matrix, M, N, dimension, lambda_for_regularization, xTx):
    for i in range(N):
        Ci = np.diag(C_matrix[:, i])
        xT_Ci_x = xTx + np.matmul(np.matmul(np.transpose(X), Ci - np.identity(M)), X)
        lI = np.dot(lambda_for_regularization, np.identity(dimension))
        xT_Ci_pi = np.matmul(np.transpose(X),P_matrix[:, i]) + np.matmul(np.matmul(np.transpose(X), Ci-np.identity(M)), P_matrix[:, i])
        Y[i] = np.linalg.solve(xT_Ci_x + lI, xT_Ci_pi)                                              # Ax = b에서 x를 구하는 데 사용되는 함수
```



```python
for i in range(10):
    if i!=0:
        yTy = np.matmul(np.transpose(Y),Y)
        xTx = np.matmul(np.transpose(X),X)
        optimize_user(X, Y, C_matrix, P_matrix, M, N, dimension, lambda_for_regularization, yTy)
        optimize_item(X, Y, C_matrix, P_matrix, M, N, dimension, lambda_for_regularization, xTx)

    predict = np.matmul(X, np.transpose(Y))
    predict_error, confidence_error, regularization, total_loss = loss_function(C_matrix, P_matrix, predict, X, Y, lambda_for_regularization)
```

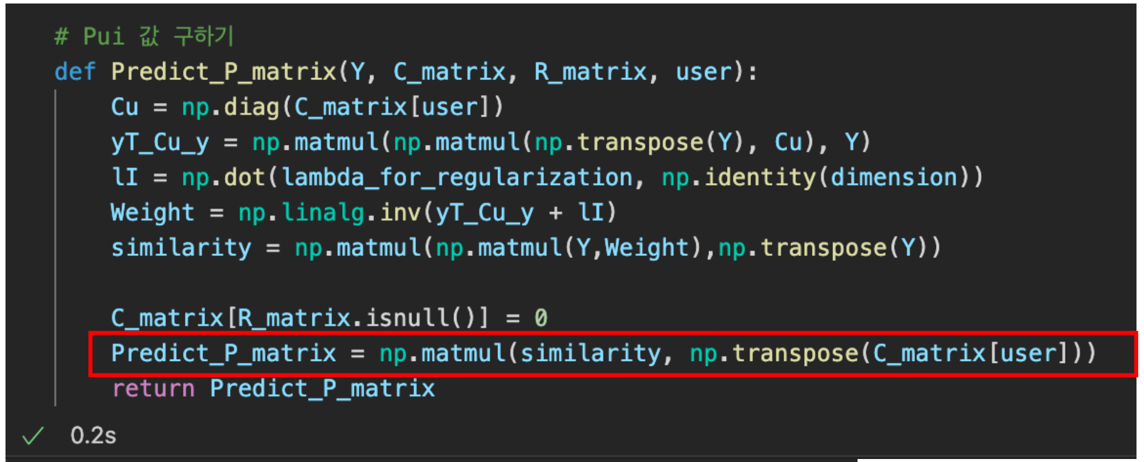
$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

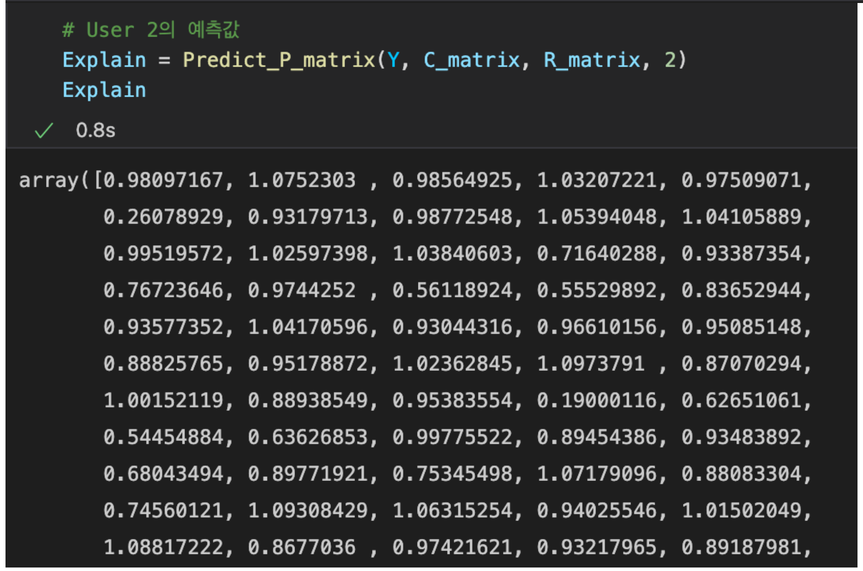


Code Review

- Explaining Recommendations

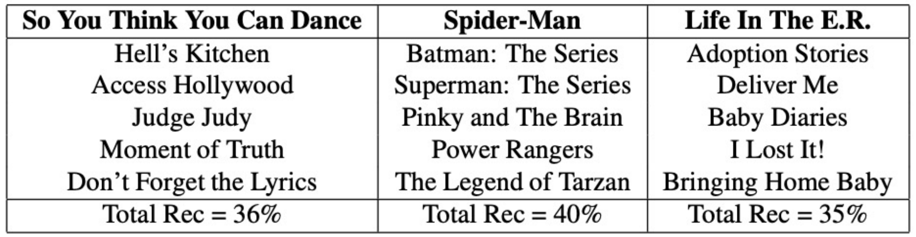


```
# Pui 값 구하기
def Predict_P_matrix(Y, C_matrix, R_matrix, user):
    Cu = np.diag(C_matrix[user])
    yT_Cu_y = np.matmul(np.matmul(np.transpose(Y), Cu), Y)
    lI = np.dot(lambda_for_regularization, np.identity(dimension))
    Weight = np.linalg.inv(yT_Cu_y + lI)
    similarity = np.matmul(np.matmul(Y,Weight),np.transpose(Y))

    C_matrix[R_matrix.isnull()] = 0
    Predict_P_matrix = np.matmul(similarity, np.transpose(C_matrix[user]))
    return Predict_P_matrix
✓ 0.2s
```



```
# User 2의 예측값
Explain = Predict_P_matrix(Y, C_matrix, R_matrix, 2)
Explain
✓ 0.8s

array([0.98097167, 1.0752303 , 0.98564925, 1.03207221, 0.97509071,
       0.26078929, 0.93179713, 0.98772548, 1.05394048, 1.04105889,
       0.99519572, 1.02597398, 1.03840603, 0.71640288, 0.93387354,
       0.76723646, 0.9744252 , 0.56118924, 0.55529892, 0.83652944,
       0.93577352, 1.04170596, 0.93044316, 0.96610156, 0.95085148,
       0.88825765, 0.95178872, 1.02362845, 1.0973791 , 0.87070294,
       1.00152119, 0.88938549, 0.95383554, 0.19000116, 0.62651061,
       0.54454884, 0.63626853, 0.99775522, 0.89454386, 0.93483892,
       0.68043494, 0.89771921, 0.75345498, 1.07179096, 0.88083304,
       0.74560121, 1.09308429, 1.06315254, 0.94025546, 1.01502049,
       1.08817222, 0.8677036 , 0.97421621, 0.93217965, 0.89187981,
```

$$\hat{p}_{ui} = \sum_{j:r_{uj}>0} s^u_{ij} c_{uj} = \underline{s_{i1} * c_{u1}} + \underline{s_{i2} * c_{u2}} + \underline{s_{i3} * c_{u3}} + \underline{s_{i4} * c_{u4}} + \dots$$

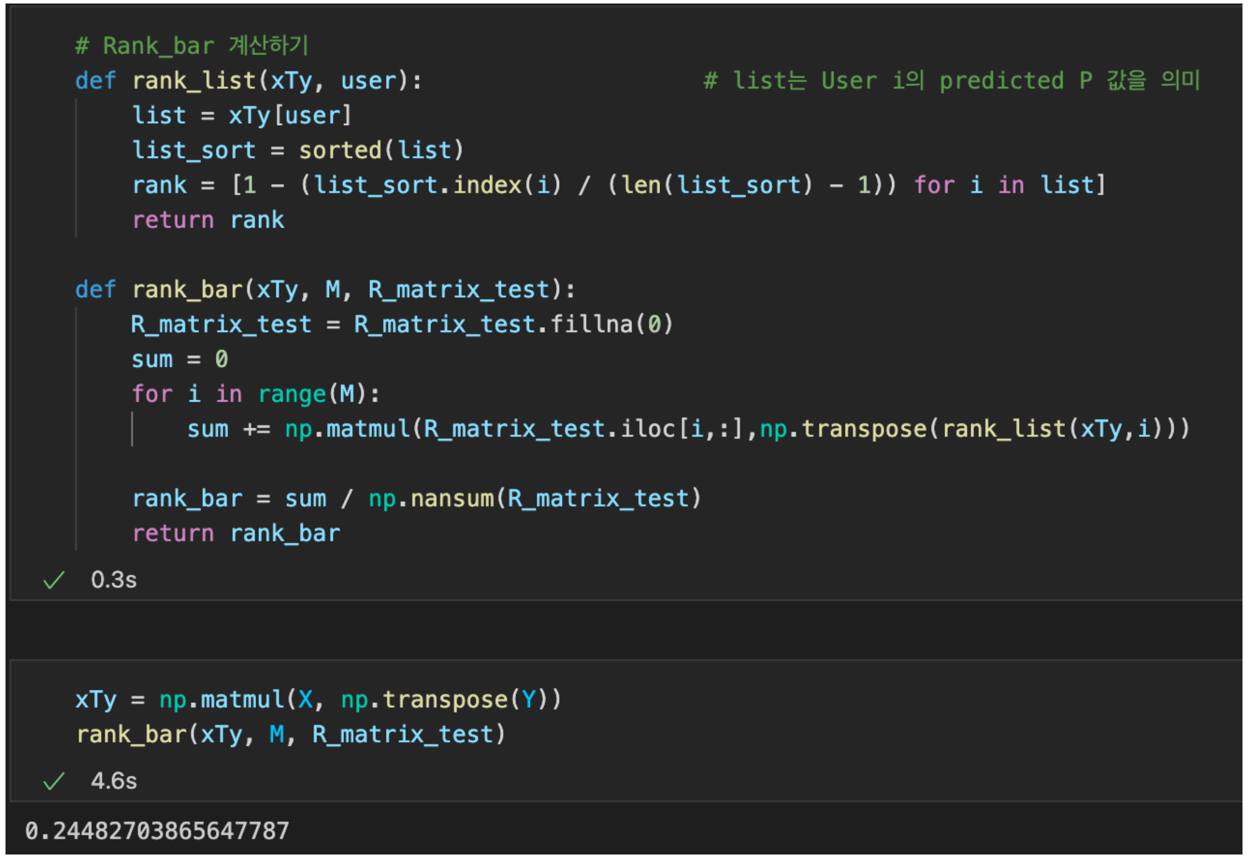


So You Think You Can Dance	Spider-Man	Life In The E.R.
Hell's Kitchen Access Hollywood Judge Judy Moment of Truth Don't Forget the Lyrics	Batman: The Series Superman: The Series Pinky and The Brain Power Rangers The Legend of Tarzan	Adoption Stories Deliver Me Baby Diaries I Lost It! Bringing Home Baby
Total Rec = 36%	Total Rec = 40%	Total Rec = 35%

Table 1. *Three recommendations with explanations for a single user in our study. Each recommended show is recommended due to a unique set of already-watched shows by this user.*

Code Review

- Experiment



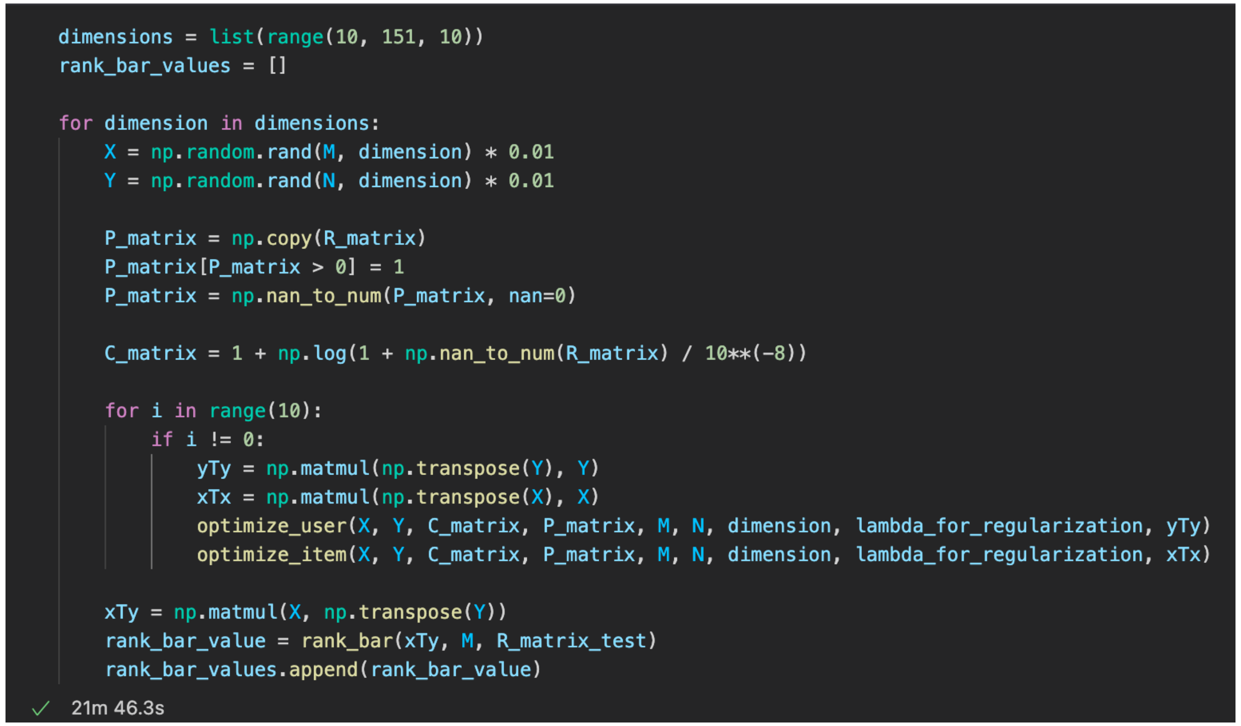
```
# Rank_bar 계산하기
def rank_list(xTy, user):                    # list는 User i의 predicted P 값을 의미
    list = xTy[user]
    list_sort = sorted(list)
    rank = [1 - (list_sort.index(i) / (len(list_sort) - 1)) for i in list]
    return rank

def rank_bar(xTy, M, R_matrix_test):
    R_matrix_test = R_matrix_test.fillna(0)
    sum = 0
    for i in range(M):
        sum += np.matmul(R_matrix_test.iloc[i,:],np.transpose(rank_list(xTy,i)))

    rank_bar = sum / np.nansum(R_matrix_test)
    return rank_bar
✓ 0.3s
```

```
xTy = np.matmul(X, np.transpose(Y))
rank_bar(xTy, M, R_matrix_test)
✓ 4.6s
0.24482703865647787
```

$$\overline{rank} = \frac{\sum_{u,i} r_{ui}^t rank_{ui}}{\sum_{u,i} r_{ui}^t}$$

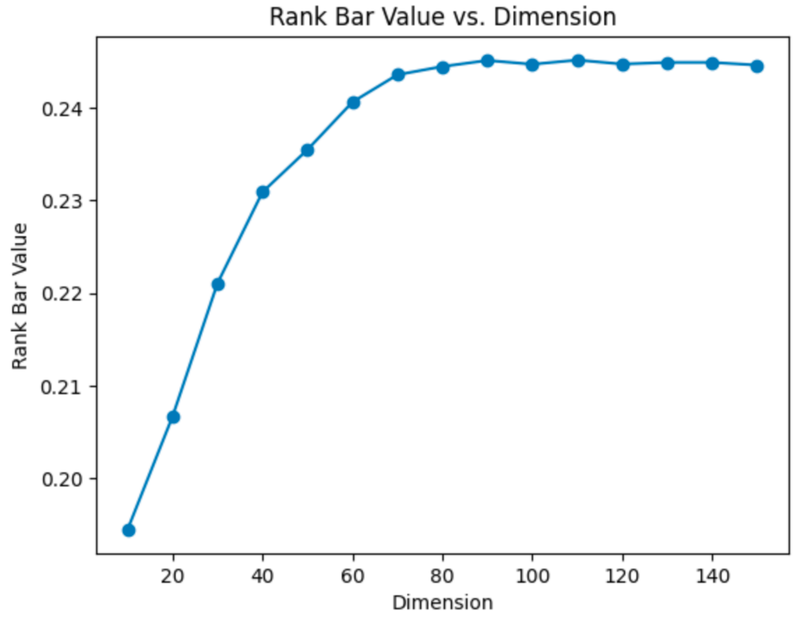
Code Review

- Experiment



```
dimensions = list(range(10, 151, 10))
rank_bar_values = []

for dimension in dimensions:
    X = np.random.rand(M, dimension) * 0.01
    Y = np.random.rand(N, dimension) * 0.01

    P_matrix = np.copy(R_matrix)
    P_matrix[P_matrix > 0] = 1
    P_matrix = np.nan_to_num(P_matrix, nan=0)

    C_matrix = 1 + np.log(1 + np.nan_to_num(R_matrix) / 10**(-8))

    for i in range(10):
        if i != 0:
            yTy = np.matmul(np.transpose(Y), Y)
            xTx = np.matmul(np.transpose(X), X)
            optimize_user(X, Y, C_matrix, P_matrix, M, N, dimension, lambda_for_regularization, yTy)
            optimize_item(X, Y, C_matrix, P_matrix, M, N, dimension, lambda_for_regularization, xTx)

    xTy = np.matmul(X, np.transpose(Y))
    rank_bar_value = rank_bar(xTy, M, R_matrix_test)
    rank_bar_values.append(rank_bar_value)
```

✓ 21m 46.3s



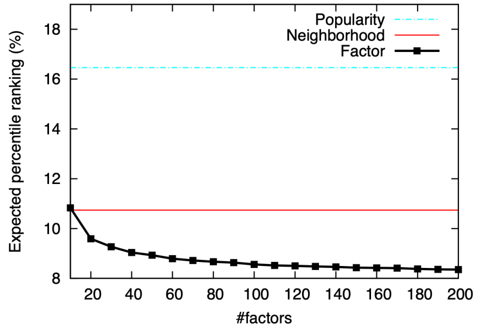


Figure 1. Comparing factor model with popularity ranking and neighborhood model.

Thank you for watching!

- 코드 깃허브 업로드 완료