2023/07/11

DSAIL summer-internship

STUDENT

유혜원

Hyewon Ryu

Collaborative Filtering for Implicit Feedback Datasets

Table of Contents

I Research Background & Motivation

- Recommender systems provide users with personalized recommendations for products or services, which hopefully suit their unique taste and needs.

Content based approach

User profile

Item profile

Recommand Similar Items using User profile and Item profile

✓ Collaborative Filtering

Explicit feedback

Implicit feedback

<u>Assumption</u>
Similar users will show similar preference

Neighborhood models

Latent factor models

I Research Background & Motivation

Explicit feedback

Most convenient input in recommend system is the high quality explicit feedback which includes explicit input by users regarding their interest in products.

<u>Example</u>

Netfilix star rating,

TV program thumbs-up/down

Implicit feedback

Explicit feedback is not always available.

Thus, recommenders can infer user preferences from the more abundant implicit feedback, which indirectly reflect opinion through observing user behavior.

<u>Example</u>

Purchase history, Browsing history, Search patterns

II Characteristics of Implicit Feedback

Cannot use algorithms that were designed with explicit feedback directly

- <u>No negative feedback</u>

  A user that did not watch a certain show might have done so because she dislikes the show or just because she did not know about the show or was not available to watch it

- <u>Numerical value of implicit feedback does not indicate preference</u>

  Numerical values of implicit feedback describe the frequency of actions. A larger value is not indicationg a higher preference but higher confidence level

- <u>Implicit feedback is inherently noisy</u>

  Purchased items can be gifts for others, or the user can be disappointed with the product

- <u>Evaluation of implicit feedback recommender requires appropriate measures</u>

  If we gather data on television show, it is unclear how to evaluate a show that has been watched more than once, or how to compare two shows that are on at the same time, and hence cannot both be watched by user

III Model Description

Notation

for users: u, v

for items: i, j

⟶ observation: r_{ui}

Ex) The number of times u perchased i or the time u spent on webpage i

⟶ If no action was observed, r_{ui} is set to zero.

- similarity of item i and j: s_{ij}

in Latent Factor models,

- user–factors vector: $x_u \in R^f$
- item–factors vector: $y_i \in R^f$

- cost function

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

III Model Description

Notation

in suggesting model,

- preference of user u to item i

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

- confidence in observing p_{ui}

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

- cost function

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

$\longrightarrow$ Alternating Least Squares (ALS) optimization

(user-item matrix is too large to SGD optimization)

III Model Description

ALS optimization

when two parameter exist, ALS minimizes two loss functions alternatively until they converge

- cost function

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

$$\longrightarrow \left.\begin{array}{l} \text{1) fix item-factor } y_i \\ \text{2) optimize user-factor } x_u \\ \text{3) fix user-factor } x_u \\ \text{4) optimize item-factor } y_i \end{array}\right\} \text{repeat until converge}$$

III Model Description

ALS optimization

- cost function

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

when one of the factors is fixed, the problem is turned into linear regression optimization

$\longrightarrow$ closed form solution exists

when calculate differentiation on x_u

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

$\longrightarrow$ $Y^T C^u Y = \underline{Y^T Y} + Y^T \underline{(C^u - I)} Y$

independent of u

only has n_u non-zero elements

when calculate differentiation on y_i

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

IV Experiment settings & results in paper

Data description

- collected from a digital television service
- about 300,000 set top boxes
- about 17,000 unique programs which aired during a four week period
- r_{ui} : for each user u, and program i, represent how many times user u watched program i (show length was used as base units)
- train data: collected over 4 weeks
- test data: 1 week after train data is collected $\longrightarrow r_{ui}^t$

IV Experiment settings & results in paper

Reflection of data characteristics

- Remove "easy" predictions from the test set corresponding to the shows that had been watched by that user during the training period.
- To make the test set more accurate, toggle to zero all entries with $r^t_{ui} < 0.5$
- Employ the log scaling scheme with $\epsilon = 10^{-8}$

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

- Down-weight the second and subsequent shows after a channel tuning event.

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

Experimentally authors found $a = 2, b = 6$

IV Experiment settings & results in paper

Evaluation methodology

- Precision based metrics are not appropriate, as they require knowing which programs are undesired to a user.
- Recall based metrics are applicable.

$rank_{ui}$ percentile–ranking of program i within the ordered list of all programs prepared for user u

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

expected percentile ranking of a watching unit in the test period

IV Experiment settings & results in paper

Evaluation results

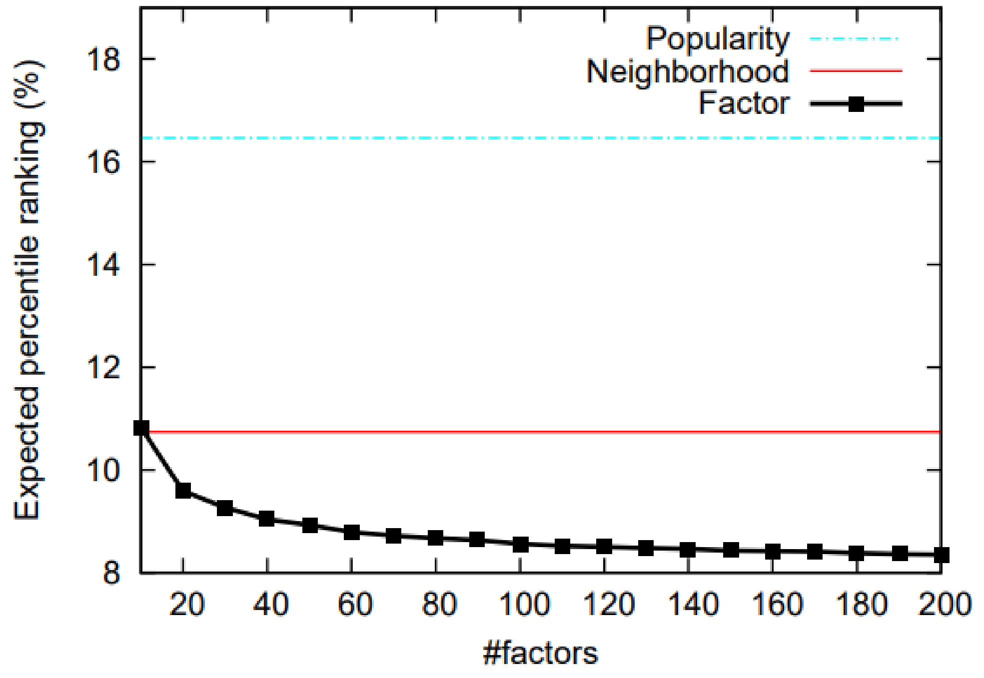


Figure 1. Comparing factor model with popularity ranking and neighborhood model.

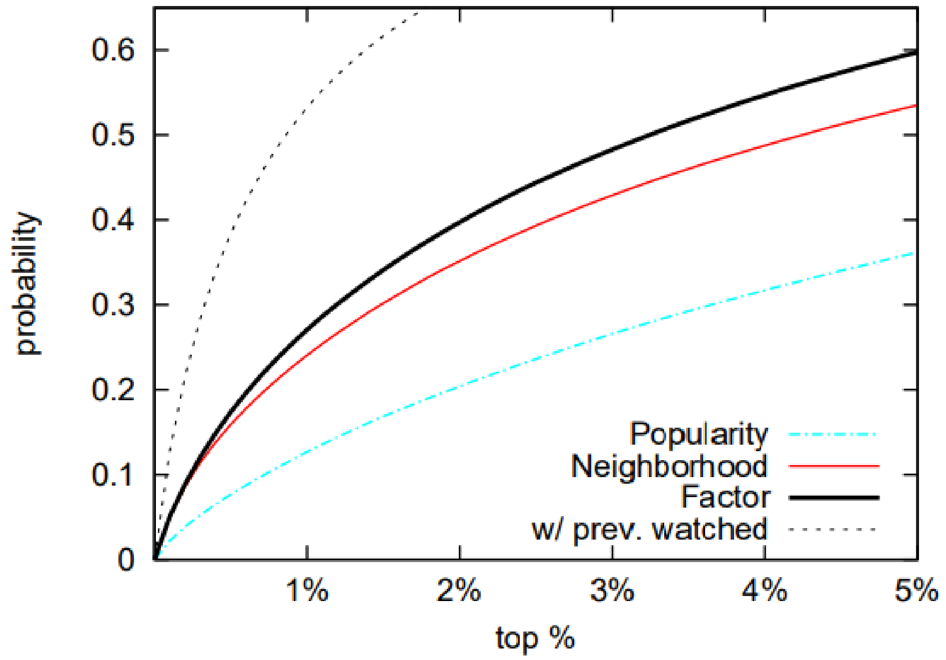


Figure 2. Cumulative distribution function of the probability that a show watched in the test set falls within top x% of recommended shows.

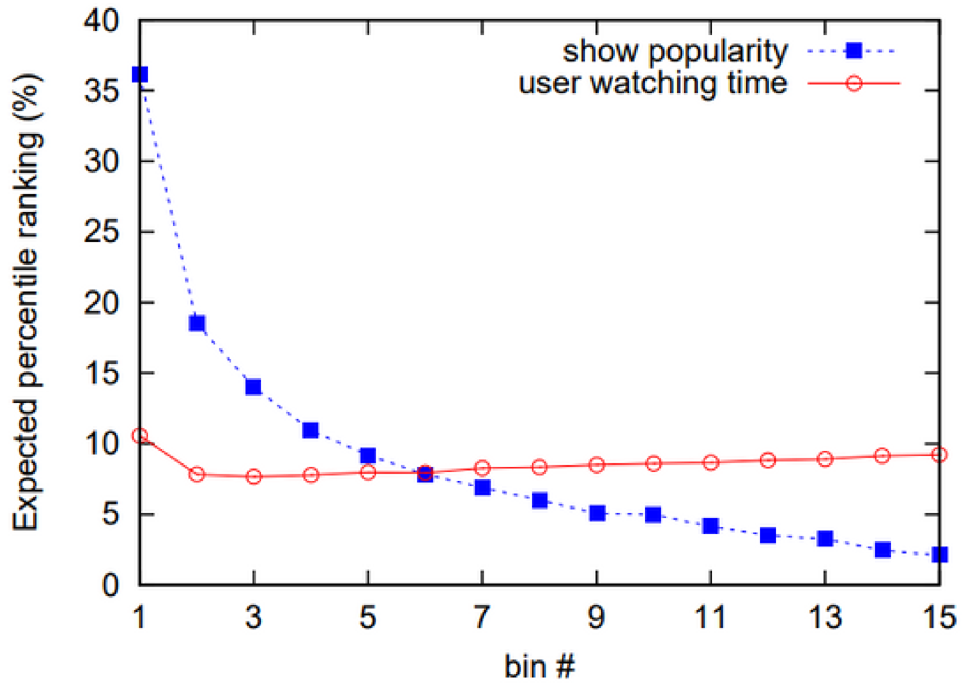


Figure 3. Analyzing the performance of the factor model by segregating users/shows based on different criteria.

- number of factors : 100

V Implementation

Dataset

Original Dataset ⟶ Reduced Dataset

Music Recommendation Dataset

- Number of Users: 358868
- Number of Artists: 292363
- Observation Type: plays

Reduced Dataset

- Number of Users: 100
- Number of Artists: 7257
- Observation Type: plays

userID	artistNM	plays
1	bloc party	1677
1	radiohead	1287
1	the mars volta	1024

Train/Test set Split

- Randomly seperate train and test datasets within each user
- 80 : 20

V Implementation

Matrix construction

R matrix

n (artist)

m (user)

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

P matrix

n (artist)

m (user)

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

use log scaling scheme
(number of music plays
varies over a large range)

C matrix

n (artist)

m (user)

V Implementation

Matrix construction

R matrix

```
def create_R_matrix(df, train_set):
    # 모든 고유한 userID와 artistNM을 추출
    all_userIDs = np.unique((df['userID'].unique()))
    all_artistNMs = np.unique((df['artistNM'].unique()))

    # 행렬 초기화
    plays_matrix = pd.DataFrame(0, index=all_userIDs, columns=all_artistNMs)

    # train set에 있는 데이터로 행렬 채우기
    for _, row in train_set.iterrows():
        userID = row['userID']
        artistNM = row['artistNM']
        plays = row['plays']
        plays_matrix.loc[userID, artistNM] = plays

    return plays_matrix
```

P matrix

```
def create_P_matrix(R_matrix):
    P_matrix = np.copy(R_matrix)
    P_matrix[R_matrix>0] = 1
    return P_matrix
```

C matrix

```
def create_C_matrix(R_matrix):
    C_matrix = 1 + alpha_*np.log(1+R_matrix/epsilon_)
    C_matrix = np.array(C_matrix)
    return C_matrix
```

V Implementation

Matrix construction

User-factor matrix (X)

f (factor)

m (user)

Item-factor matrix (Y)

f (factor)

n (item)

```
# number of user and item
m = R_matrix.shape[0]
n = R_matrix.shape[1]

# user-factor matrix initialize
X = np.random.rand(m, n_factor) * 0.01
print(X.shape)

# item-factor matrix initialize
Y = np.random.rand(n, n_factor) * 0.01
print(Y.shape)
```

V Implementation

Alternating Least Square (ALS) training

Loss function

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

```python
def loss_function(C, P, pred, X, Y, lambda_):
    predict_error = np.square(P-pred)
    confidence_error = np.sum(C*predict_error)
    regularization_error = lambda_*(np.sum(np.square(X))+np.sum(np.square(Y)))
    total_loss = confidence_error + regularization_error
    return np.sum(predict_error), confidence_error, regularization_error, total_loss
```

Optimizer

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \quad y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \longrightarrow Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$$

```python
def user_optimization(X, Y, C_matrix, P_matrix, m, n, n_factor, lambda_):
    yT = np.transpose(Y)
    yTy = np.matmul(yT, Y)
    for user in range(m):
        Cu = np.diag(C_matrix[user])
        yT_Cu_y = yTy + np.matmul(np.matmul(yT, Cu - np.identity(n)),Y)
        lambda_I = lambda_ * np.identity(n_factor)
        yT_Cu_pu = np.matmul(np.matmul(yT,Cu), np.transpose(P_matrix[user]))
        X[user] = np.linalg.solve(yT_Cu_y+lambda_I, yT_Cu_pu)
```

```python
def item_optimization(X, Y, C_matrix, P_matrix, m, n, n_factor, lambda_):
    xT = np.transpose(X)
    xTx = np.matmul(xT, X)
    for item in range(n):
        Ci = np.diag(C_matrix[:,item])
        xT_Ci_x = xTx + np.matmul(np.matmul(xT, Ci - np.identity(m)),X)
        lambda_I = lambda_ * np.identity(n_factor)
        xT_Ci_pi = np.matmul(np.matmul(xT,Ci), np.transpose(P_matrix[:,item]))
        Y[item] = np.linalg.solve(xT_Ci_x+lambda_I, xT_Ci_pi)
```

V Implementation

Alternating Least Square (ALS) training

training function

```
def train(n_iter,X, Y, C_matrix, P_matrix, m, n, n_factor, lambda_):
    # result save
    predict_errors = []
    confidence_errors = []
    regularization_list = []
    total_losses = []

    for i in range(n_iter):
        print(f'processing on {i+1}th iteration')
        if i != 0:
            user_optimization(X, Y, C_matrix, P_matrix, m, n, n_factor, lambda_)
            item_optimization(X, Y, C_matrix, P_matrix, m, n, n_factor, lambda_)
        predict = np.matmul(X, np.transpose(Y))
        predict_error, confidence_error, regularization, total_loss = loss_function(C_matrix, P_matrix, predict, X, Y, lambda_)
        predict_errors.append(predict_error)
        confidence_errors.append(confidence_error)
        regularization_list.append(regularization)
        total_losses.append(total_loss)

        print(f'{i+1}th iteration is done')

    fin_predict = np.matmul(X, np.transpose(Y))
    print('final predict')
    print([fin_predict])
    return fin_predict, predict_errors, confidence_errors, regularization_list, total_losses
```

V Implementation

Evaluation Metric

Expected Percentile Ranking

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

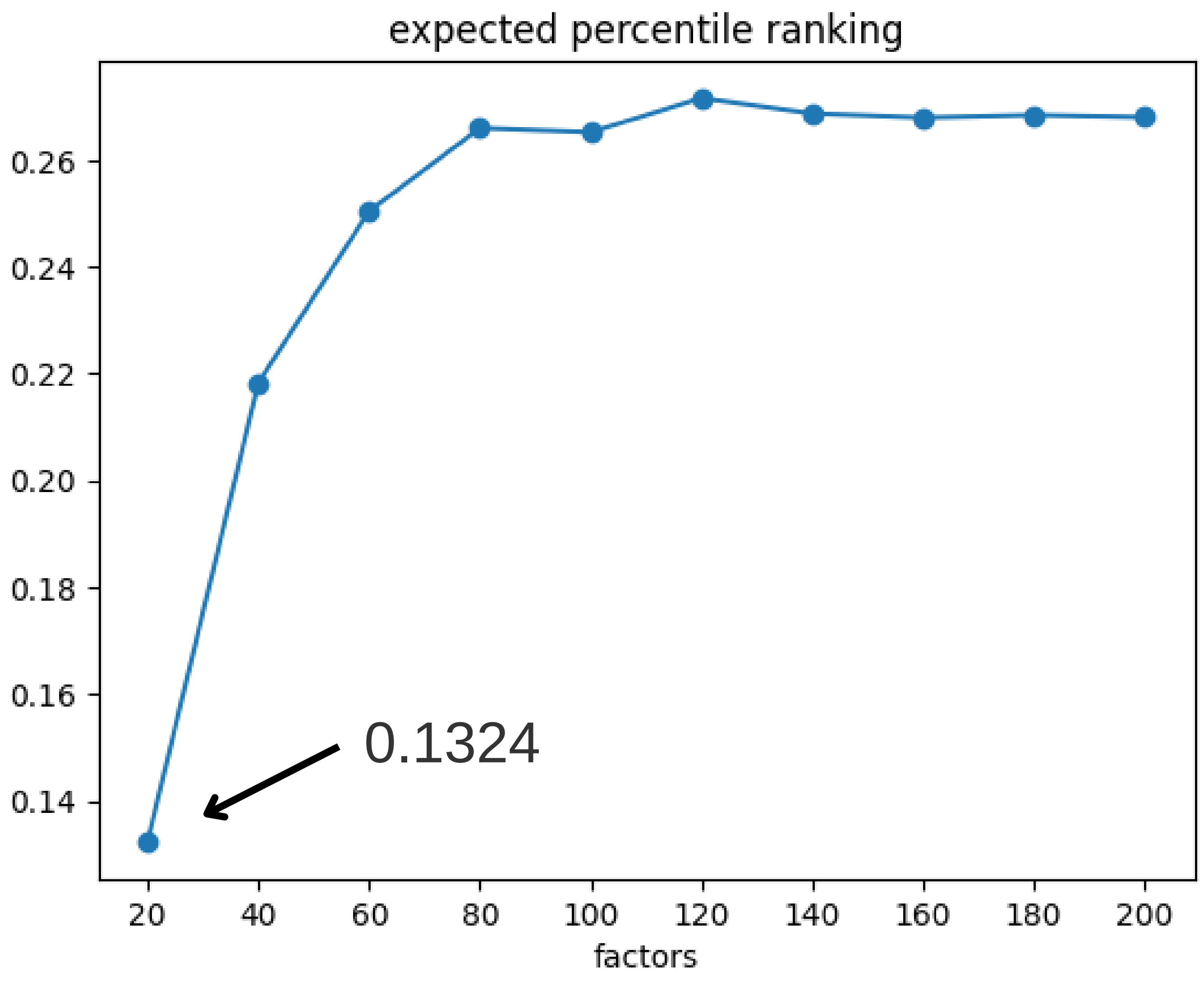
```
def calculate_rank(predict_df, R_matrix, train_set, test_set):

  # 1. predict_df에서 0.5 미만 값을 갖는 경우 모두 0으로 변환
  predict_df[predict_df < 0.5] = 0

  # 2. rank_matrix 구성 (size는 predict_df와 동일)
  rank_matrix = predict_df.copy()

  # predict_df에서 train에 포함된 데이터의 경우 값을 모두 0으로 처리
  train_indices = [(row, col) for row, col in train_set[['userID', 'artistNM']].values]
  rank_matrix[train_indices] = 0

  # user(row)별로 값이 0이 아닌 데이터들에 대해 percentile rank를 계산하여 값 변환
  for row in rank_matrix.index:
    nonzero_indices = rank_matrix.columns[rank_matrix.loc[row] != 0].tolist()
    if len(nonzero_indices) > 0:
      tmp_col_list = rank_matrix.loc[row, nonzero_indices].tolist()
      tmp_qua_list = calculate_percentile_ranks(tmp_col_list)
      rank_matrix.loc[row, nonzero_indices] = np.array(tmp_qua_list)
```

```
# 3. R_test_matrix 구성
R_test_matrix = R_matrix.copy()

# R_matrix에서 train에 포함된 데이터의 경우 값을 모두 0으로 처리
train_indices = [(row, col) for row, col in train_set[['userID', 'artistNM']].values]
R_test_matrix[train_indices] = 0

# 4. rank_bar 계산 (expected percentile ranking)
rt = np.array(R_test_matrix)
rank = np.array(rank_matrix)
rt_rank = rt*rank
numerator = np.sum(rt_rank)
denominator = np.sum(rt)
rank_bar = numerator/denominator
print(rank_bar)
return rank_bar
```

V Implementation

Experiment Results

Expected Percentile Ranking change by number of factors



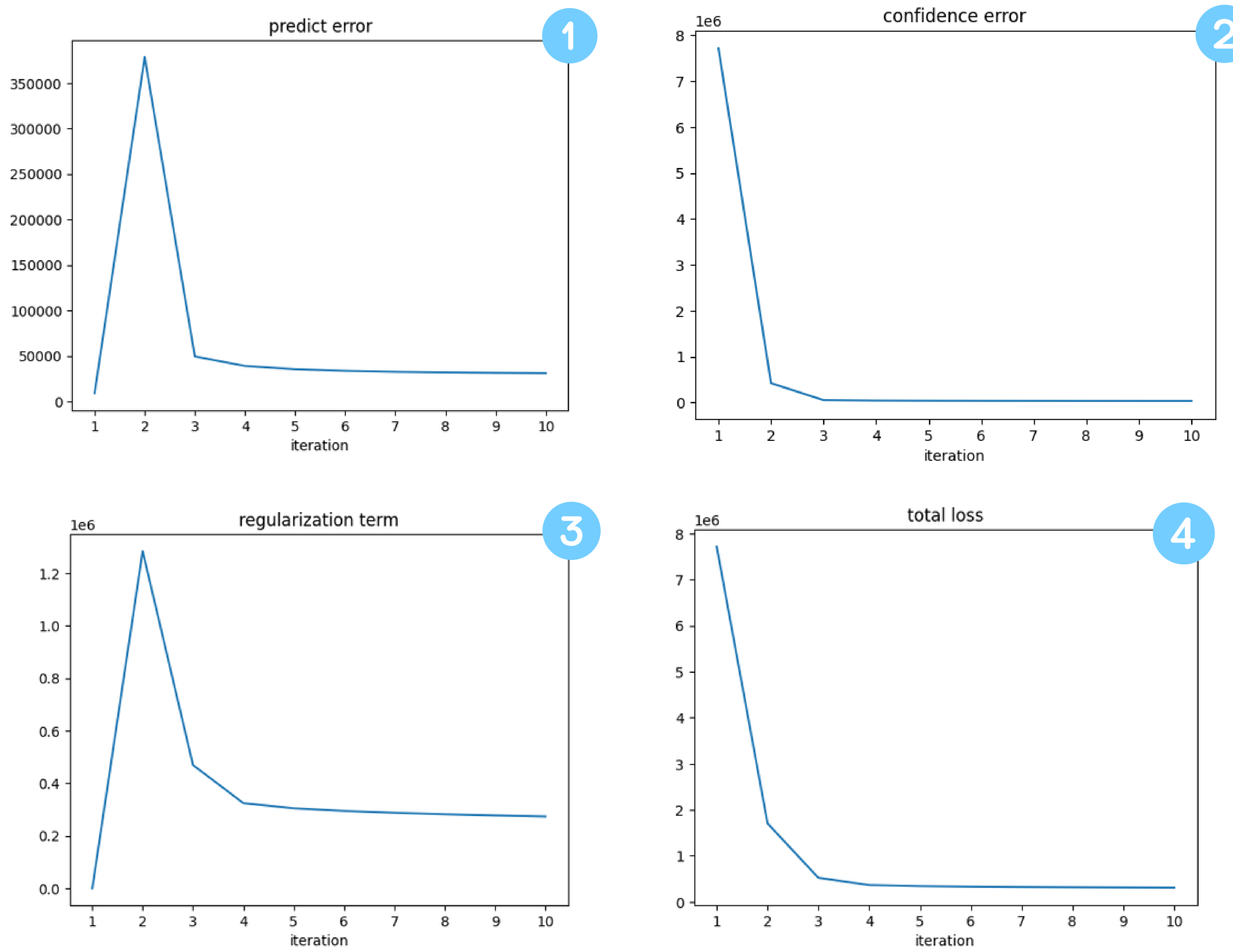
- Setting

  n_factor : range(20,201,20)
  n_iter : 10
  lambda : 150
  alpha: 40
  epsilon : 10**(-8)

- Ranking increases as the number of factors increases
- Contrary to the original paper
- Due to the decrease in the number of data, it can be interpreted that a small number of factors are sufficient to represent the data

V Implementation

Experiment Results



$$\min_{x_\star,y_\star} \sum_{u,i} \underbrace{\underbrace{c_{ui}(\underbrace{p_{ui}-x_u^T y_i)^2}_{1}}_{2} + \lambda\underbrace{\left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\right)}_{3}}_{4}$$

- Fast converge on this dataset

  -> when iteration 3~4

STUDENT

유혜원

Hyewon Ryu

Thank you for listening