추천시스템 구현 프로젝트
Recommender System

Contents

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

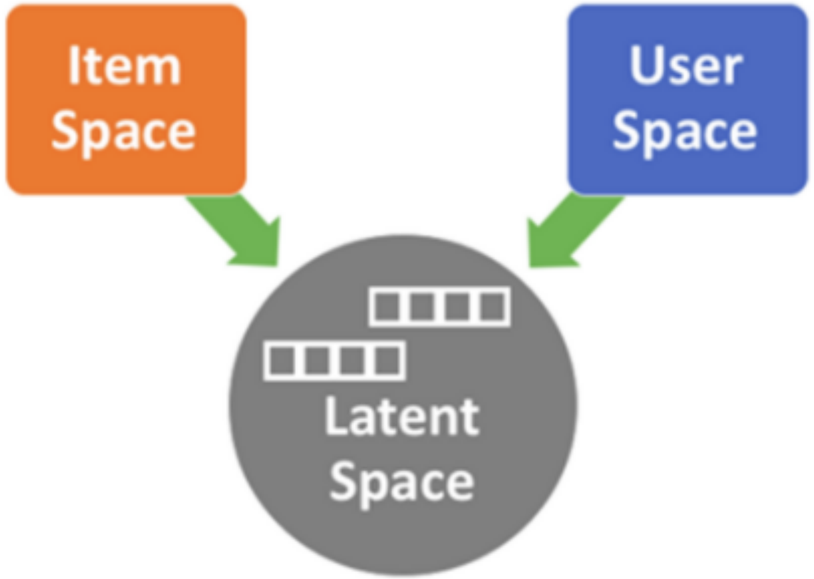
- 협업필터링 기반 추천시스템 분류

	메모리 기반 알고리즘	모델 기반 알고리즘
설명	메모리에 있는 데이터를 계산해서 추천하는 방식	데이터로부터 미리 모델을 구성 후 필요 시 추천하는 방식
특징	개별 사용자 데이터 집중	전체 사용자 패턴 집중
장점	원래 데이터에 충실하게 사용	대규모 데이터에 빠르게 반응
단점	대규모 데이터에 느리게 반응	모델 생성 과정 오래 걸림

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- Matrix Factorization(MF) 방식의 원리
  - **'사용자와 아이템 사이에는 사용자의 행동과 평점에 영향을 끼치는 잠재된 특성이 있을 것이다'**
  - User-Item Matrix를 F차원의 User와 Item의 latent factor 행렬곱으로 분해하는 방법
  - SGD, A

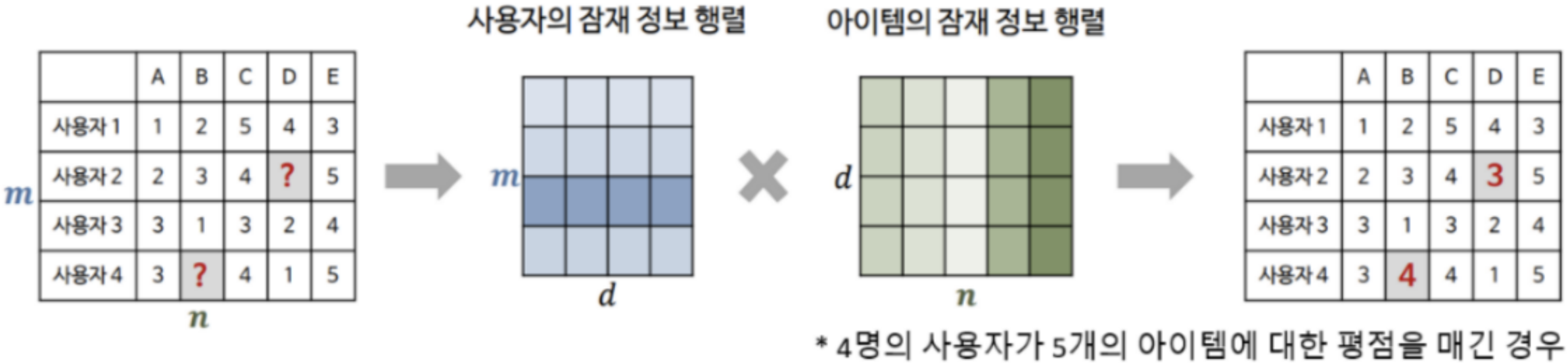
Neighborhood Model



Latent Model

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- Matrix Factorization(MF) 방식의 원리
  - 잠재 요인 협업 필터링(Latent Factor Collaborative Filtering) 방식
  - **Rating Matrix에서 빈 공간을 채우기 위해서 사용자와 상품을 잘 표현하는 차원을 찾는 방법**
  - **SVD, SGD, ALS 알고리즘(Latent Fator Collaborative Filtering) 사용**



	A	B	C	D	E
사용자 1	1	2	5	4	3
사용자 2	2	3	4	?	5
사용자 3	3	1	3	2	4
사용자 4	3	?	4	1	5

	A	B	C	D	E
사용자 1	1	2	5	4	3
사용자 2	2	3	4	3	5
사용자 3	3	1	3	2	4
사용자 4	3	4	4	1	5

* 4명의 사용자가 5개의 아이템에 대한 평점을 매긴 경우

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- Matrix Factorization(MF) 방식의 원리

사용자 요인 P

사용자 \ 잠재요인	액션 – 드라마 (-1 ~ 1)	판타지 – 사실주의 (-1 ~ 1)
User 1	-0.43	0.21
User 2	0.31	0.92
User 3	0.69	-0.03
User 4	0.46	-0.3

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

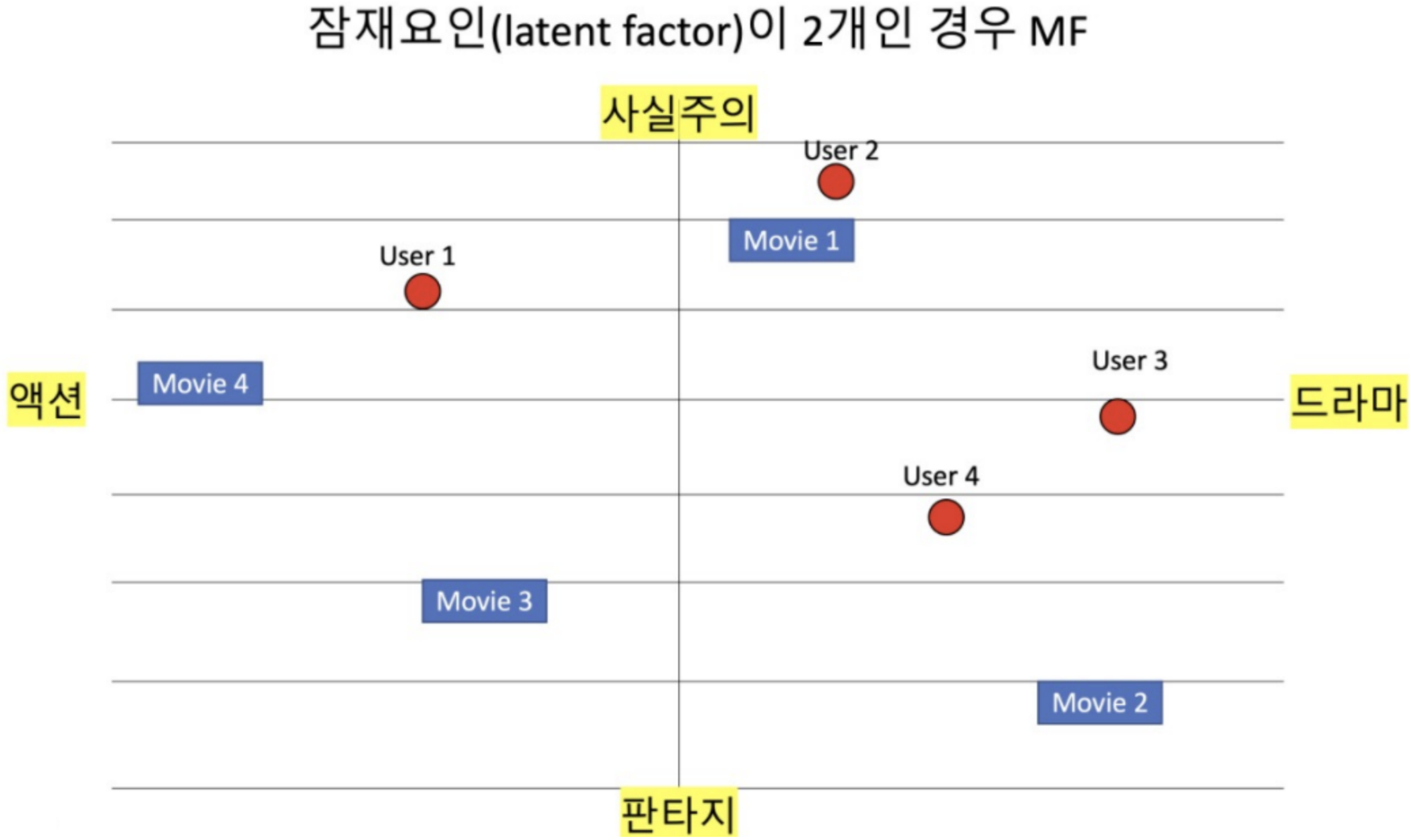
- Matrix Factorization(MF) 방식의 원리

아이템 요인 Q

영화 \ 잠재요인	액션 – 드라마 (-1 ~ 1)	판타지 – 사실주의 (-1 ~ 1)
Movie 1	0.31	0.6
Movie 2	0.61	-0.82
Movie 3	-0.38	-0.61
Movie 4	-0.79	0.08

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- Matrix Factorization(MF) 방식의 원리



추천시스템 구현 - Matrix Factorization(MF) 기반 추천

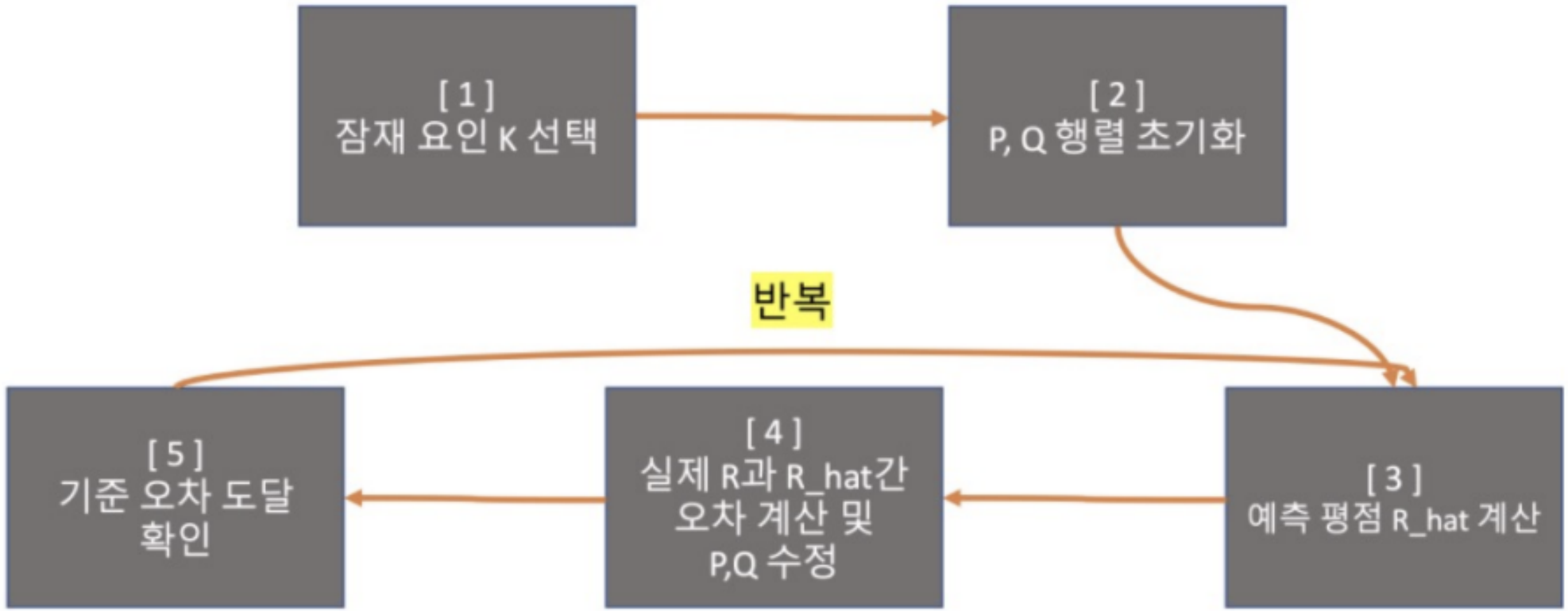
- Matrix Factorization(MF) 방식의 원리

사용자의 영화별 예측 평점 $R \approx P \times Q^T = \hat{R}$

사용자 \ 영화	Movie 1	Movie 2	Movie 3	Movie 4
User 1	-0.0073	-0.4345	0.0353	0.3565
User 2	0.6481	-0.5653	-0.679	-0.1713
User 3	0.1959	0.4455	-0.2439	-0.5475
User 4	-0.0374	0.5266	0.0082	-0.3874

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘
  - MF 알고리즘 개념적 설명



추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘

1. User Latent와 Item Latent의 임의로 초기화

?	3	2
5	1	2
4	2	1
2	?	4

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

=

User Latent(U)

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

np.dot

Item Latent(V)의 Transpose

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

*고유값 분해(eigen value Decomposition)와 같은 행렬을 대각화하는 방법

$$\text{Minimize } J = \frac{1}{2}||R - UV^T||^2$$

subject to:

No constraints on U and V

$$S = \{(i,j) : r_{ij} \text{ is observed}\}$$

$$\text{Minimize } J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$$

subject to:

No constraints on U and V

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘

2. 순차적으로 값이 존재하는 모든 평점에 대해 Gradient Descent 과정진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

*Desent

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$= \sum_{j:(i,j)\in S} (e_{ij})(-v_{jq}) + \lambda u_{iq} \quad \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right) (-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

$$= \sum_{i:(i,j)\in S} (e_{ij})(-u_{iq}) + \lambda v_{jq} \quad \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

Error : 3-0.6663 = 2.3337

descentUser = -2.3337 * [-1.1078, 0.8972] + 0.01 * [0.5756, 1.4534]

descentItem = -2.3337 * [0.5756, 1.4534] + 0.01 * [-1.1078, 0.8972]

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘

3. New Latent Update

?	3	2
5	1	2
4	2	1
2	?	4

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

*Desent

Error : 3-0.6663 = 2.3337

descentUser = -2.3337 * [-1.1078, 0.8972] + 0.01 * [0.5756, 1.4534] = [2.591, -2.0793]

descentItem = -2.3337 * [0.5756, 1.4534] + 0.01 * [-1.1078, 0.8972] = [-1.3544, -3.3828]

*New Latent = Latent - learning rate * Descent

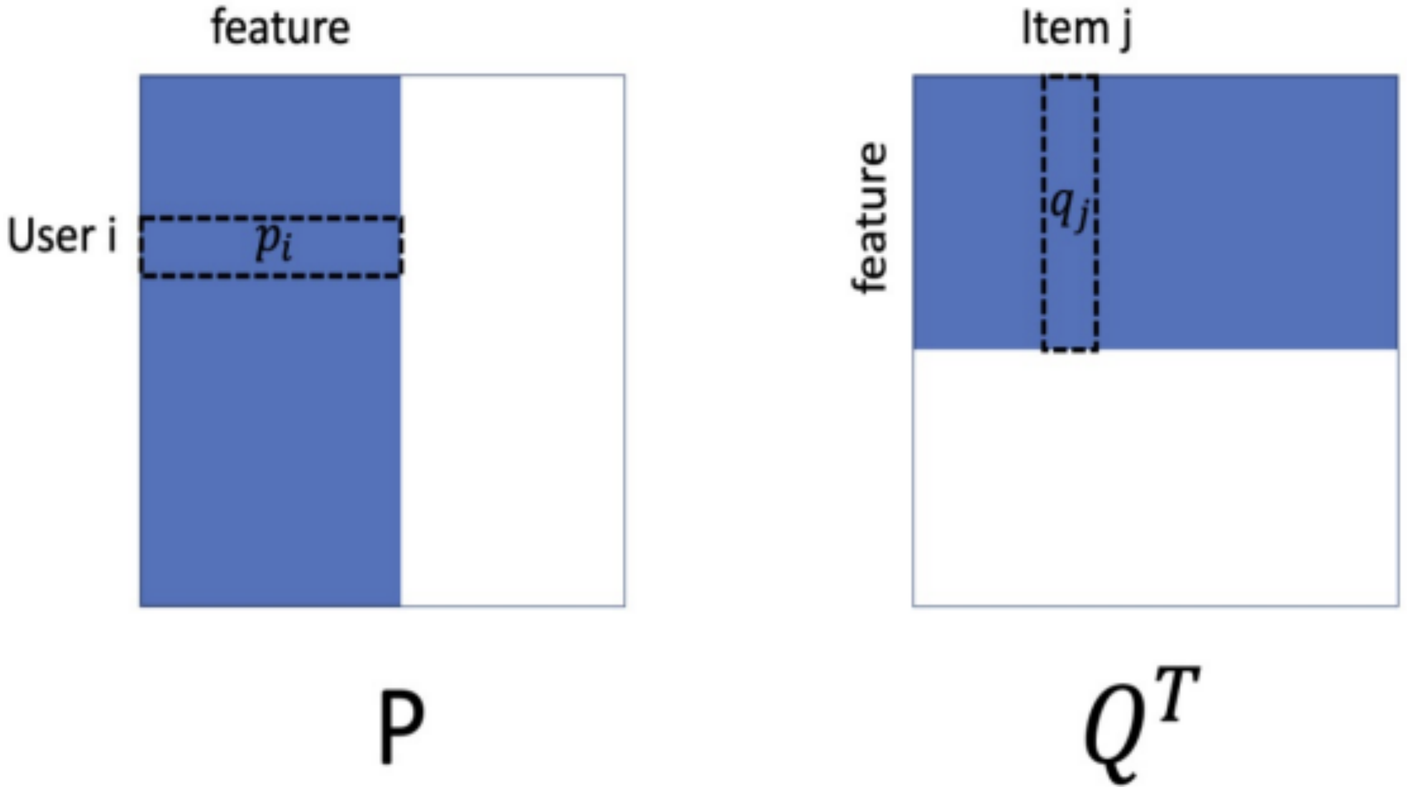
New User Latent = [0.5756, 1.4534] - 0.05 * [2.591, -2.0793] = [0.446, 1.5574]

New Item Latent = [-1.1078, 0.8972] - 0.05 * [-1.3544, -3.3828] = [-1.0401, 1.0663]

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘

P, Q 행렬에서 사용자 i와 아이템 j의 잠재요인



$$\hat{r}_{ij} = p_j^T q_j = \sum_{k=1}^{K} p_{ik}\, q_{kj}$$

$$e_{ij} = (r_{ij} - \hat{r}_{ij})$$

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik}\, q_{kj}\right)^2$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2\,(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2 e_{ij} q_{kj}$$

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2\,(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2 e_{ij} p_{ik}$$

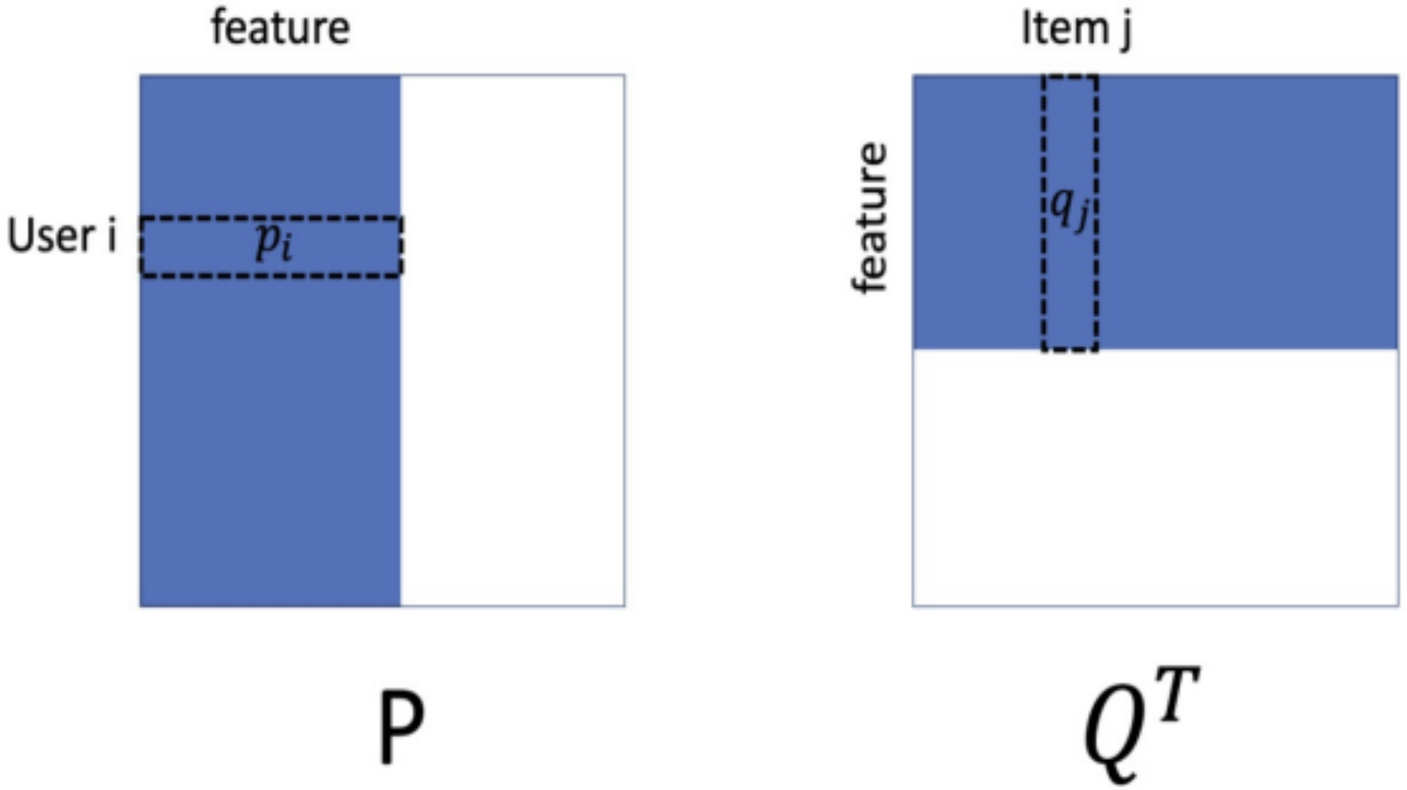
$$p'_{ik} = p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\,\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\,\alpha e_{ij} p_{ik}$$

추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- SGD 알고리즘

P, Q 행렬에서 사용자 i와 아이템 j의 잠재요인



$$e_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)$$

$$p'_{ik} = p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2 e_{ij} q_{kj} - \beta p_{ik})$$

$$q'_{kj} = q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2 e_{ij} p_{ik} - \beta q_{kj})$$

$$\hat{r}_{ij} = \mathrm{b} + \mathrm{bu_i} + \mathrm{bd_j} + \sum_{k=1}^{K} p_{ik} q_{kj}$$

$$bu'_i = bu_i + \alpha(e_{ij} - \beta bu_i)$$

$$bd'_j = bd_j + \alpha(e_{ij} - \beta bd_j)$$

실 습 - SGD를 사용한 MF 기본 알고리즘

- MF 클래스 생성

```
class MF():
  def __init__(self, ratings, hyper_params):
    self.R = np.array(ratings)
    self.num_users, self.num_items = np.shape(self.R)
    self.x = hyper_params['x']
    self.alpha = hyper_params['alpha']
    self.beta = hyper_params['beta']
    self.iteration = hyper_params['iteration']
    self.verbose = hyper_params['verbose']
```

실 습 - SGD를 사용한 MF 기본 알고리즘

- rmse( )

```
def rmse(self) :
  xs, ys = self.R.nonzero()
  self.predictions = []
  self.errors = []
  for x,y in zip(xs, ys):
    prediction = self.get_prediction(x,y)
    self.predictions.append(prediction)
    self.errors.append(self.R[x,y]-prediction)
  self.predictions = np.array(self.predictions)
  self.errors = np.array(self.errors)
  return np.sqrt(np.mean(self.errors**2))
```

실 습 - SGD를 사용한 MF 기본 알고리즘

- train( )

```
def train(self) :
self.P = np.random.normal(scale=1./self.x, size = (self.num_users, self.x))
self.Q = np.random.normal(scale=1./self.x, size = (self.num_items, self.x))

self.b_u = np.zeros(self.num_users)
self.b_d = np.zeros(self.num_items)
self.b = np.mean(self.R[self.R.nonzero()])

rows, columns = self.R.nonzero()
self.samples = [(i, j, self.R[i,j]) for i,j in zip(rows, columns)]
```

실 습 - SGD를 사용한 MF 기본 알고리즘

- train( )

```
training_process = [] #매 반복시 rmse를 기록할 리스트
for i in range(self.iteration):
  np.random.shuffle(self.samples)
  self.sgd()
  rmse = self.rmse()
  training_process.append((i+1, rmse))
  if self.verbose:
    if(i+1)%10 == 0 :
      print("Iteration:%d ; Train RMSE = %.4f" %(i+1,rmse))
return training_process
```

실 습 - SGD를 사용한 MF 기본 알고리즘

- get_prediction( ) , sgd()

```
def get_prediction(self,i,j) :
  prediction = self.b+self.b_u[i]+self.b_d[j]+self.P[i,:].dot(self.Q[j,:].T)
  return prediction

def sgd(self):
  for i, j, r in self.samples :
    prediction = self.get_prediction(i,j)
    e = (r-prediction)

    self.b_u[i] +=self.alpha * (e - self.beta * self.b_u[i])
    self.b_d[j] +=self.alpha * (e - self.beta * self.b_d[j])

    self.P[i,:] += self.alpha * (e * self.Q[j,:]-self.beta * self.P[i,:])
    self.Q[j,:] += self.alpha * (e * self.P[i,:]-self.beta * self.Q[j,:])
```

실 습 - SGD를 사용한 MF 기본 알고리즘

- 전체 데이터를 사용한 MF

```
# 전체 데이터 사용 MF
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
hyper_params = {
    'x' : 30,
    'alpha' : 0.001,
    'beta' : 0.02,
    'iteration' : 100,
    'verbose' : True
}
mf = MF(R_temp, hyper_params)
train_process = mf.train()
```

실 습 - train/test 분리 MF 알고리즘

- train/test set 분리

```
 # train/test set 분리
from sklearn.utils import shuffle
TRAIN_SIZE = 0.75

# (사용자 - 영화 - 평점)
ratings = shuffle(ratings, random_state=2021)
cutoff = int(TRAIN_SIZE * len(ratings))
ratings_train = ratings.iloc[:cutoff]
ratings_test = ratings.iloc[cutoff:]
```

실 습 - train/test 분리 MF 알고리즘

- Test set 선정

```
# Test set 선정
def set_test(self, ratings_test) :
  test_set = []
  for i in range(len(ratings_test)):
    x = self.user_id_index[ratings_test.iloc[i,0]]
    y = self.item_id_index[ratings_test.iloc[i,1]]
    z = ratings_test.iloc[i,2]
    test_set.append([x,y,z])
    self.R[x,y]=0
  self.test_set = test_set
  return test_set
```

실 습 - train/test 분리 MF 알고리즘

- Test set RMSE 계산

```
def test_rmse(self):
  error = 0
  for one_set in self.test_set :
    predicted = self.get_prediction(one_set[0], one_set[1])
    error +=pow(one_set[2]-predicted, 2)
  return np.sqrt(error/len(self.test_set))
```

실 습 - train/test 분리 MF 알고리즘

- test( )

```
def test(self) :
self.P = np.random.normal(scale=1./self.x, size = (self.num_users, self.x))
self.Q = np.random.normal(scale=1./self.x, size = (self.num_items, self.x))

self.b_u = np.zeros(self.num_users)
self.b_d = np.zeros(self.num_items)
self.b = np.mean(self.R[self.R.nonzero()])

rows, columns = self.R.nonzero()
self.samples = [(i, j, self.R[i,j]) for i,j in zip(rows, columns)]
```

실 습 - train/test 분리 MF 알고리즘

- test( )

```
training_process = [] #매 반복시 rmse를 기록할 리스트
for i in range(self.iteration):
  np.random.shuffle(self.samples)
  self.sgd()
  rmse1 = self.rmse()
  rmse2 = self.test_rmse()
  training_process.append((i+1, rmse1, rmse2))
  if self.verbose:
    if(i+1)%10 == 0 :
      print("Iteration:%d ; Train RMSE = %.4f ; Test RMSE= %.4f" %(i+1,rmse1, rmse2))
return training_process
```

실 습 - train/test 분리 MF 알고리즘

- 전체 데이터 사용 MF

```
# 전체 데이터 사용 MF
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
hyper_params = {
    'x' : 30,
    'alpha' : 0.001,
    'beta' : 0.02,
    'iteration' : 100,
    'verbose' : True
}
mf = NEW_MF(R_temp, hyper_params)
test_set = mf.set_test(ratings_test)
result = mf.test()
```

실 습 - train/test 분리 MF 알고리즘

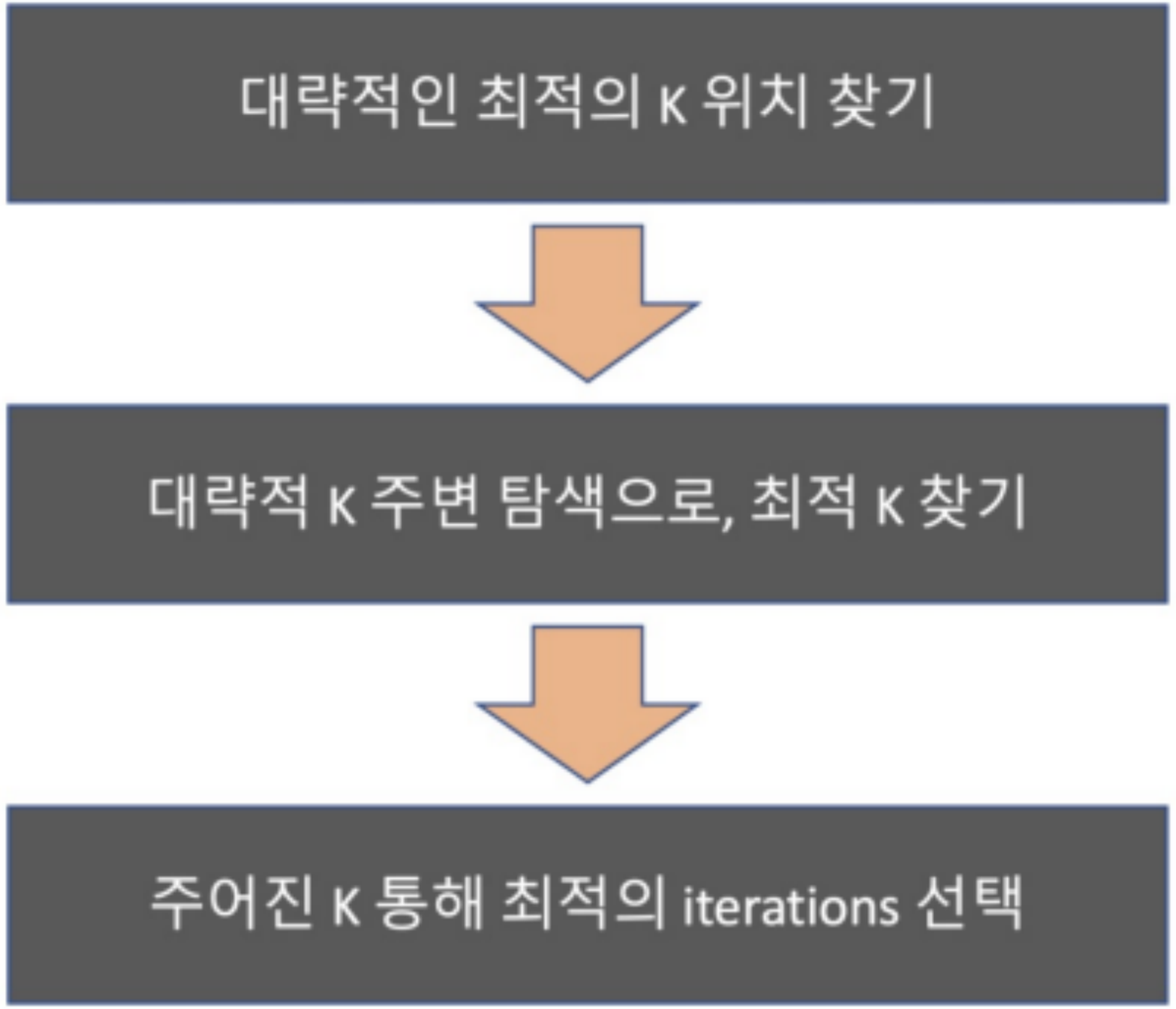
- 예측

```
def get_one_prediction(self, user_id, item_id):
  return self.get_prediction(self.user_id_index[user_id],
                             self.item_id_index[item_id])
def full_prediction(self):
  return self.b+self.b_u[:,np.newaxis]+self.b_d[np.newaxis,:]+self.P.dot(self.Q.T)
```

```
print(mf.full_prediction())
print(mf.get_one_prediction(1,2))
```

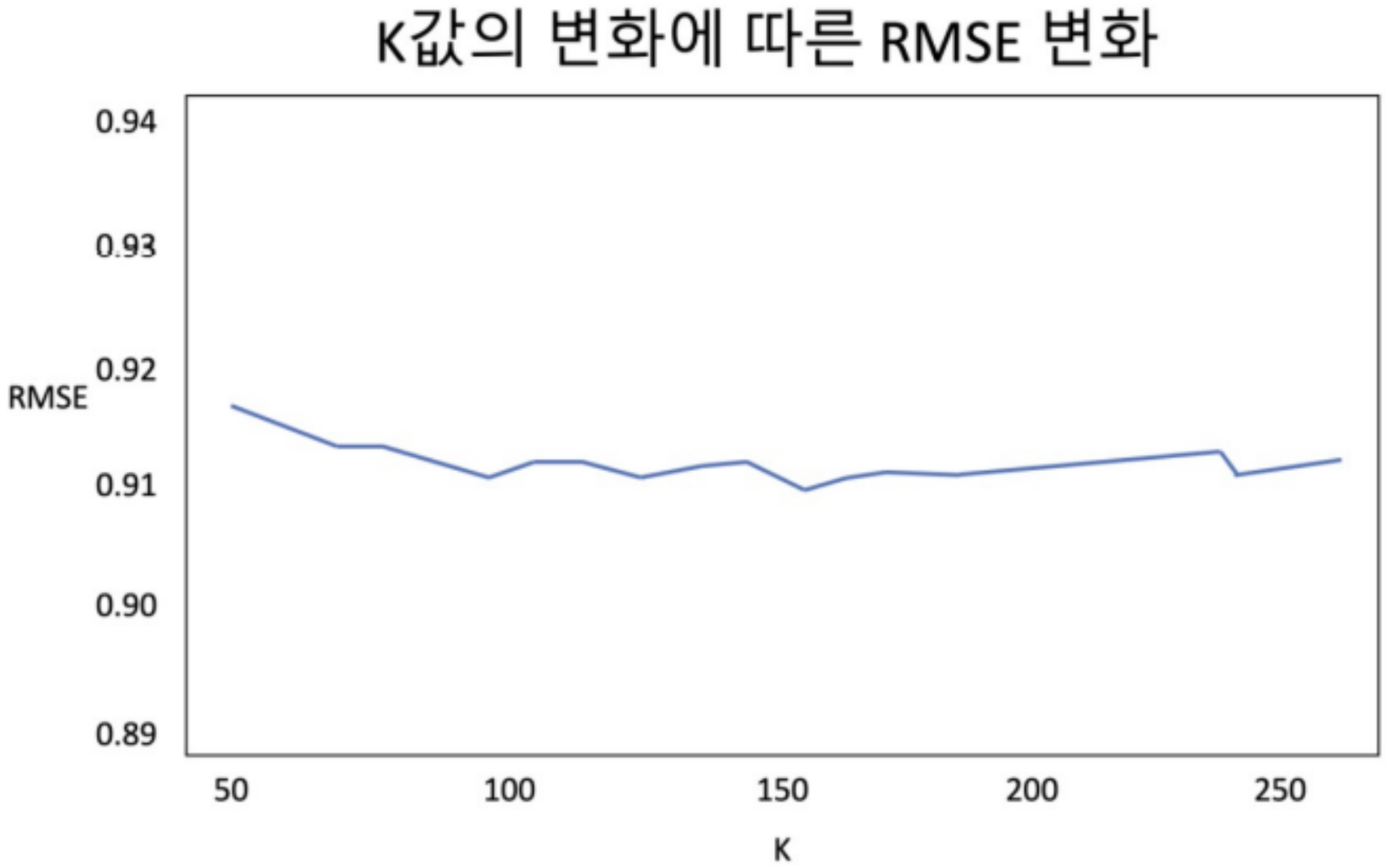
추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- 최적의 파라미터 찾기



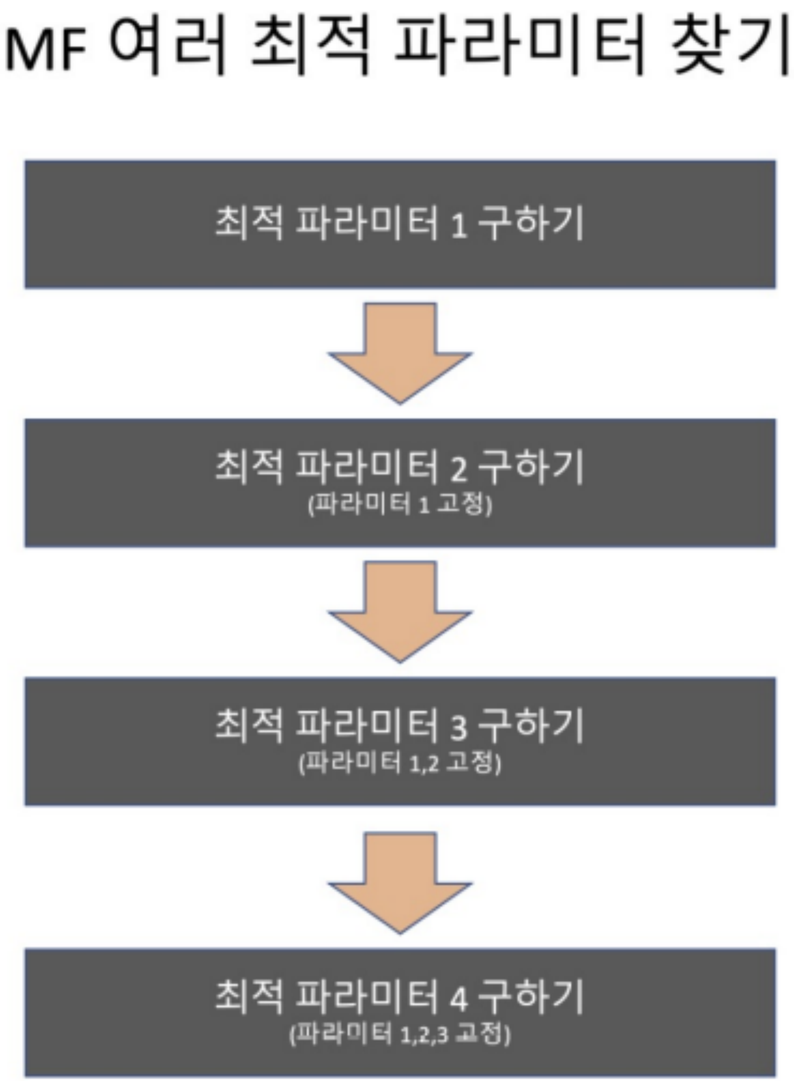
추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- 최적의 파라미터 찾기



추천시스템 구현 - Matrix Factorization(MF) 기반 추천

- 최적의 파라미터 찾기



실 습 - MF의 최적 파라미터 찾기

- 최적의 k 찾기

```
results=[]
index=[]

R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
for x in range(50, 261, 10):
  print(f'x: {x}')
  hyper_parmas={
    'x' : x, 'alpha' : 0.001, 'beta' : 0.02, 'iteration' : 300, 'verbose' : True }

  mf = NEW_MF(R_temp, hyper_params)
  test_set = mf.set_test(ratings_test)
  result = mf.test()
  index.append(x)
  results.append(result)
```

실 습 - MF의 최적 파라미터 찾기

- 최적의 iteratins값 찾기

```
# 최적의 iteratins값 찾기
summary=[]
for i in range(len(results)) :
  RMSE = []
  for result in results[i]:
    RMSE.append(result[2])
  min = np.min(RMSE)
  j= RMSE.index(min)
  summary.append(index[i], j+1, RMSE[j])
```

감/사/합/니/다