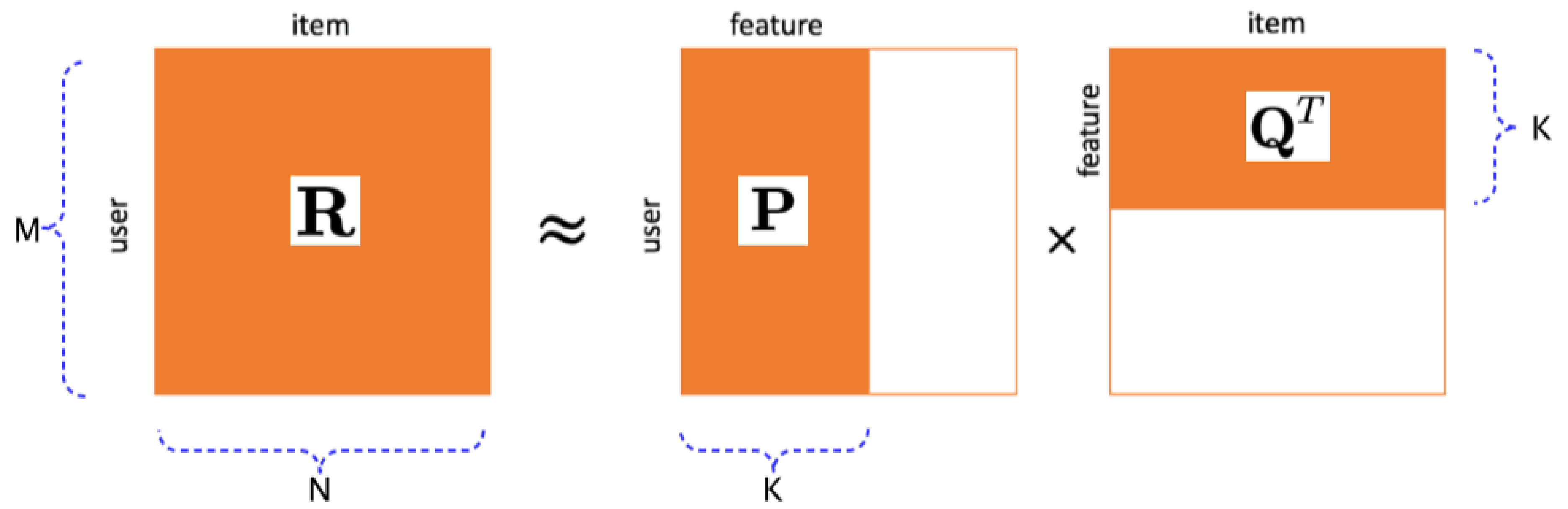
Recommend System Study

Section 5
Matrix factorization and Deep Learning

2022.07.06

miju Kim

Matrix Factorization



$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

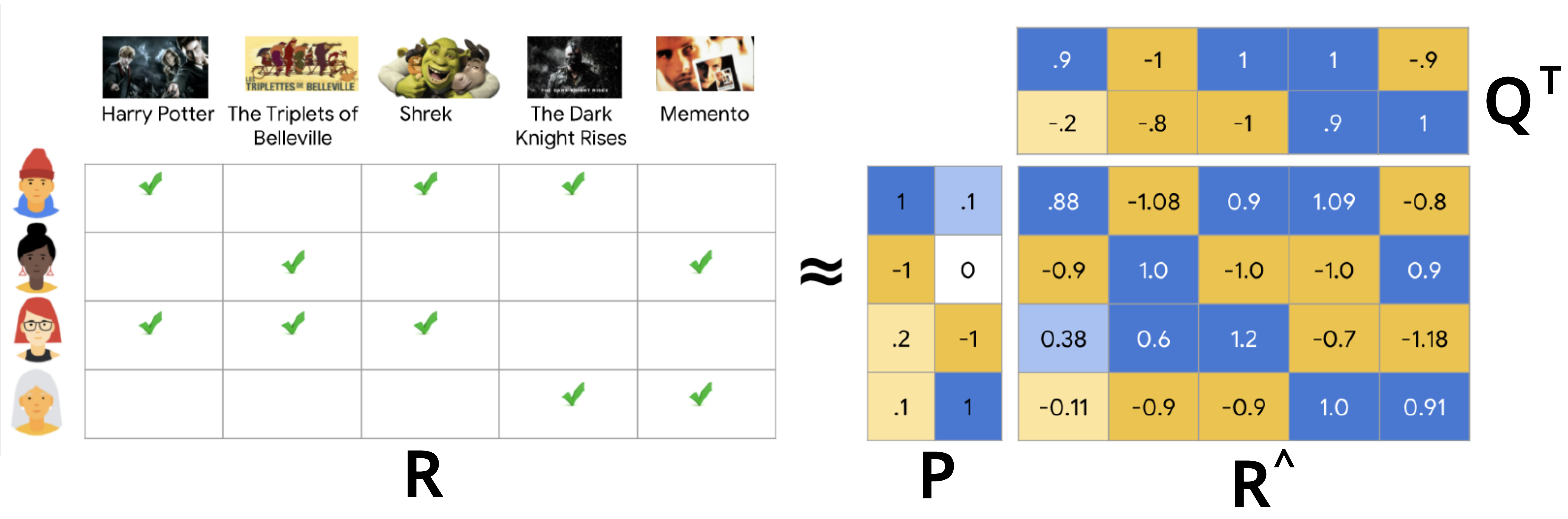
R: Rating matrix P: User latent matrix Q: Item latent matrix

User와 Item 간의 평가 정보를 나타내는 Rating Matrix를 User Latent Matrix와 Item Latent Matrix로 분해하는 기법

User Latent 매트릭스인 P와 Item Latent 매트릭스인 Q의 전치행렬을 곱해서 R^을 만들면,
원래 Rating Matrix의 근사값을 구할 수 있다는 아이디어에 기반한 방식

- User Latent Matrix(P) = K * (User의수)
- Item Latent Matrix(Q^T) = (Item의 수) * K
- Rating Matrix(R) = (Item * K) X (K *User) => User * Item

Matrix Factorization



Rating Matrix = (User의 수) * (Item의 수)
각 칸에는 각 유저가 기록한 해당 아이템에 대한 평가가 수치로써 기록

-> *Sparse Matrix*
R을 실제로 메모리에 배열로 저장하지 않음, 용량을 많이 차지하기 때문

MF는 행렬 분해 과정에서 이러한 빈칸을 채울만한 평점을 예측하는 과정이라고 볼 수 있음.

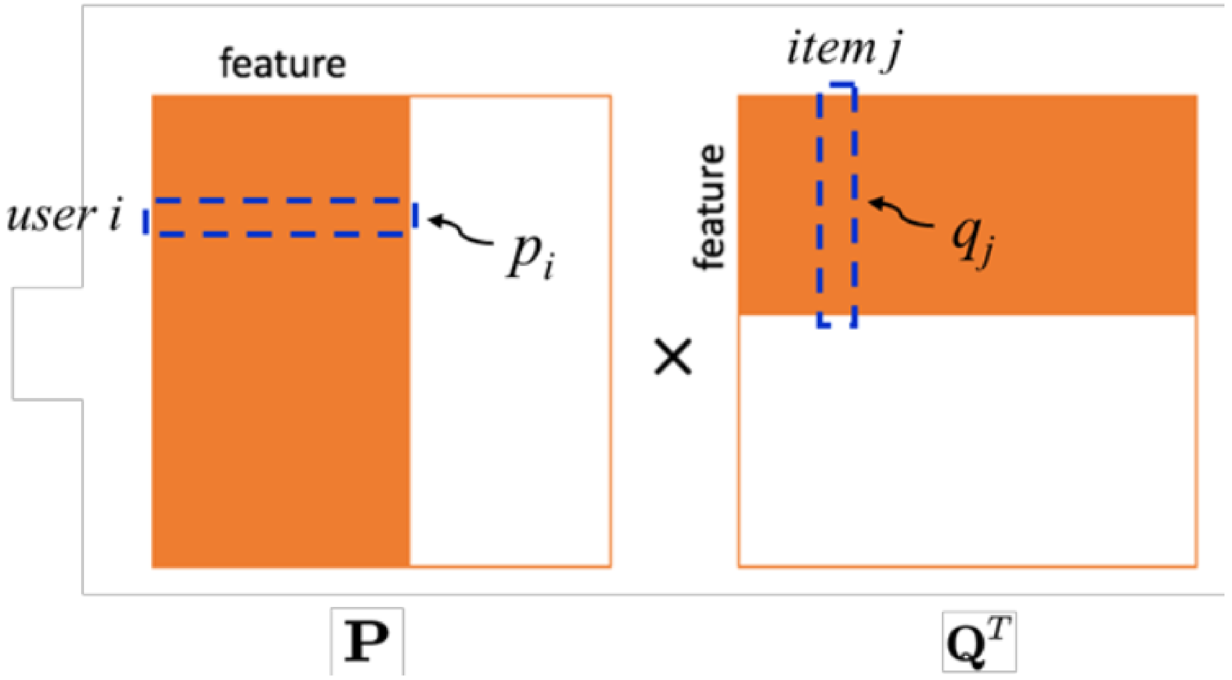
Matrix Factorization

- User Latent Matrix(P) = K * (User의수)
- Item Latent Matrix(Q^T) = (Item의 수) * K

K = 학습시에 정하는 임의의 차원 수인 Latent Factor의 크기로 사용자에 의해 Hyperparamter로 입력받는다.

K를 크게 잡으면 기존의 Rating Matrix로부터 다양한 정보를 가져갈 수 있지만,
K를 작게 잡아야만 핵심적인 정보외의 노이즈를 제거할 수 있다.

Matrix Factorization



Hypothesis

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{ik}$$

i 사용자의 j번째 아이템의 예측치의 계산은 P에서의 사용자에 해당하는 행을 고르고, q에서의 아이템을 고르면 된다.

cost function

$$\min_{P,Q} \sum_{observed\ r_{u,i}} \left(r_{u,i} - \hat{r}_{u,i}\right)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

1) MSE　　　2) 정규화

-> 비용함수의 값를 최소로 만드는 P Matrix와 Q Matrix를 찾는게 목표

Matrix Factorization

cost function

$$\min_{P,Q} \sum_{observed\ r_{u,i}} (r_{u,i} - \hat{r}_{u,i})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

1) MSE 2) 정규화

1) MSE = 실제 평점과 예측된 평점 간의 차이

$$e_{ij}^2 = (r_{ij} - \hat{r})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{jk})^2$$

P : User Latent Matrix
Q : Item Latent Matrix
K : Latent Factor의 수
r_{ij} : 사용자 i 의 아이템 j 에대한 실제 평점 값
e_{ij} : 예측 오차

-> 학습 데이터에서 실제 평점이 있는 경우(observed)에 대해서만 오차를 계산

2) 정규화 = 과적합을 방지

λ = 정규화 텀의 영향력을 어느 정도로 줄 것인지 정하는 하이퍼 파라미터

파라미터 값이 커지면 p벡터의 제곱과 q벡터의 제곱 합도 커질 것이고
이는 cost 함수도 커지게 하므로 패널티를 주는 것

-> 학습 파라미터인 p와 q의 값이 너무 커지지 않도록 규제

Matrix Factorization

Optimazation (SGD), (ALS)

$$\min_{P,Q} \sum_{observed\ r_{u,i}} \left(r_{u,i} - \hat{r}_{u,i}\right)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2)$$

1) MSE 2) 정규화

1) cost함수(L)을 p와 q로 편미분

$$\frac{\partial L}{\partial p_u} = \frac{\partial\left(r_{u,i} - p_u^T q_i\right)^2}{\partial p_u} + \frac{\partial \lambda \|p_u\|_2^2}{\partial p_u} = -2\left(r_{u,i} - p_u^T q_i\right)q_i + 2\lambda p_u = -2(e_{u,i}q_i - \lambda p_u)$$

$$\frac{\partial L}{\partial q_i} = \frac{\partial\left(r_{u,i} - p_u^T q_i\right)^2}{\partial q_i} + \frac{\partial \lambda \|q_i\|_2^2}{\partial q_i} = -2\left(r_{u,i} - p_u^T q_i\right)p_u + 2\lambda q_i = -2(e_{u,i}p_u - \lambda q_i)$$

2) Gradient를 현재 상태의 파라미터 값에서 빼줌으로써 업데이트 진행

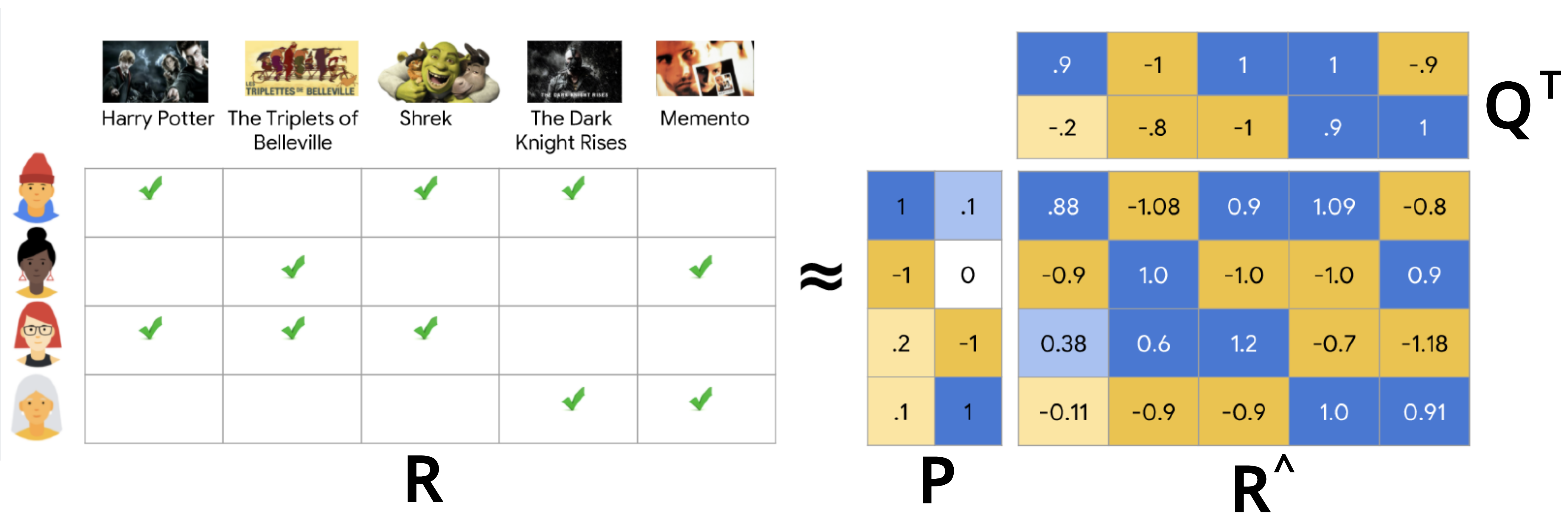
$$p_u \leftarrow p_u + \eta \cdot \left(e_{u,i}q_i - \lambda p_u\right)$$

$$q_i \leftarrow q_i + \eta \cdot \left(e_{u,i}p_u - \lambda q_i\right)$$

에타(η) = Learning Rate를 의미

Learning Rate = 파라미터를 업데이트 할 때
얼마나 크게 변화시킬지를 정하는 하이퍼 파라미터

Matrix Factorization



| training 과정

1. Latent Factor의 수 K를 정함. 위의 경우 k = 2
2. 주어진 k에 따라 P와 Q행렬을 만들고 초기화
-> 맨 처음엔 임의의 수로 채운다.
3. P와 Q를 사용해 예측 평점 R^을 구함.
4. R에 있는 실제 평점에 대해 예측 평점 R^의 예측과 비교해서
오차를 구하고, 이 오차를 줄이기 위해 P, Q값을 수정 → SGD 사용
5. 전체 오차가 미리 정해진 기준값 이하가 되거나, 미리 정해진 반복 횟수에
도달할 때 까지 3번으로 돌아가 반복

Matrix Factorization

Biases 고려

| 이유

사용자나 아이템별로 평점에 편향이 있을 수 있다.
예를 들어, 사용자 A는 점수를 후하게 주고 반면에 사용자 B는 점수를 짜게 준다던지, 아니면 어떤 영화는 유명한 명작이라 사용자의 취향과 별개로 점수가 높고, 어떤 영화는 그렇지 않을 수 있다.

1. Hypothesis = Adding Bias에서 예측 평점

$$\hat{r}_{u,i} = \mu + b_u + b_i + p_u^T q_i$$

μ : 학습 데이터 전체 평점의 평균
b_u :사용자가 가진 편향
b_i : 아이템이 가진 편향
$p_u^T q_i$: 예측 평점

μ를 더해주는 이유는 대부분의 아이템이 평균적으로 μ 정도의 값은 받는다는 것을 감안해주기 위함.

2. cost function

$$\min_{P,Q} \sum_{observed\ r_{u,i}} (r_{u,i} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

1) MSE 2) 정규화

* μ는 학습 데이터로부터 도출되는 상수값인 반면에, Bias들은 모두 학습 대상인 파라미터
-> 두번째 정규화 텀에도 Bias가 추가

(참고: p와 u는 벡터인 반면, bias들은 모두 스칼라값이므로 둘이 서로 표현법이 다르다.)

3. Optimazation (SGD)

$$b_u \leftarrow b_u + \eta \cdot (e_{u,i} - \lambda b_u)$$

$$b_i \leftarrow b_i + \eta \cdot (e_{u,i} - \lambda b_i)$$

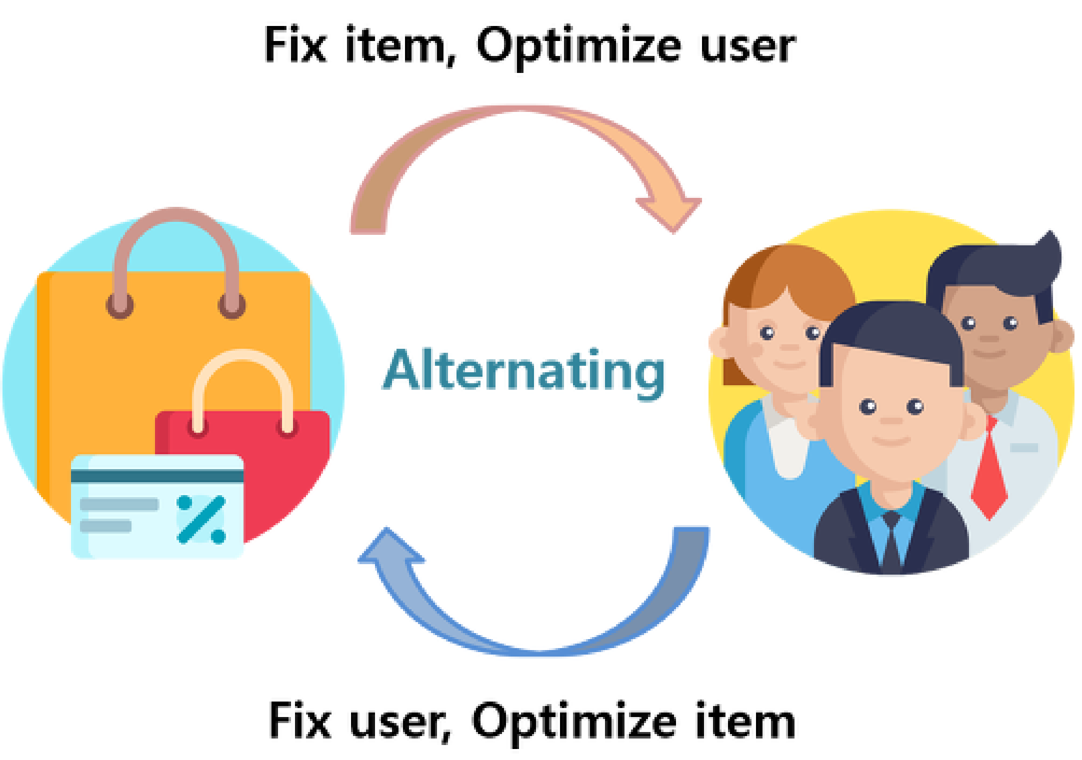
$$p_u \leftarrow p_u + \eta \cdot (e_{u,i} q_i - \lambda p_u)$$

$$q_i \leftarrow q_i + \eta \cdot (e_{u,i} p_u - \lambda q_i)$$

파라미터가 네 종류이므로, 4개의 각각 파라미터로 cost 함수를 편미분하여 Gradient를 구하고 이를 바탕으로 업데이트를 수행

Matrix Factorization

Optimazation: Alternating Least Squares (ALS)



두 개의 행렬 중 하나를 고정시키고 사용자와 아이템의 Latent Factor를 한번씩 번갈아가며 학습

아이템의 행렬을 상수로 놓고 사용자의 행렬을 학습시키고, 사용자 행렬을 상수로 놓고 아이템 행렬을 학습시키는 방식

이 과정을 계속 반복하면서 짧은 시간 내 최적의 사용자와 아이템 Latent Factor를 구함.

Gradient Descent 문제점

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{ik}$$

학습시켜야 하는 것은 P와 Q행렬 두 개이고, 이 둘은 곱셉으로 묶여있다.
이 둘을 동시에 최적화 시키는 문제는 Non-convex problem으로 귀결
-> global minimum 대신 Local minimum을 구할 수 있음.

Gradient Descent로 이를 최적화 시키는 것은 너무 느리고 많은 반복이 필요하다는 단점 존재

Reference

[1] https://yeong-jin-data-blog.tistory.com/entry/%EC%B6%94%EC%B2%9C-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-Matrix-Factorization

[2] https://sungkee-book.tistory.com/12

[3]https://yeomko.tistory.com/4?category=805638