

# **Ch 3. Model Based CF**

**I** Introduction

**II** Latent Factor Model

**III** Optimization

**IV** SVD & NMF

Chapter



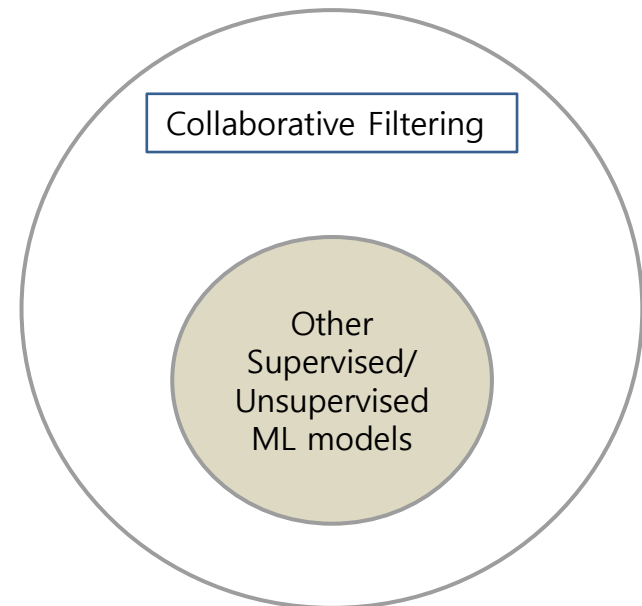
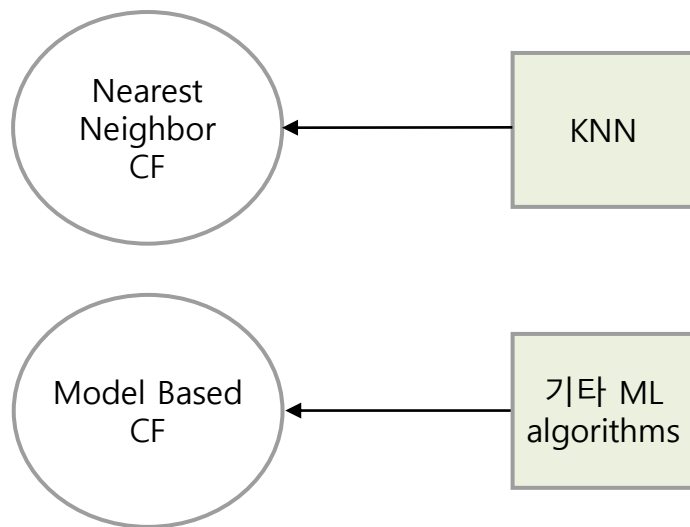
# Introduction

## 1. Model Based CF

# 01 Model Based Collaborative Filtering

모델 기반 협업필터링이란,  
유저의 선호를 예측하는 모델을 구축하여 추천을 제공하는 협업필터링 방법을 의미

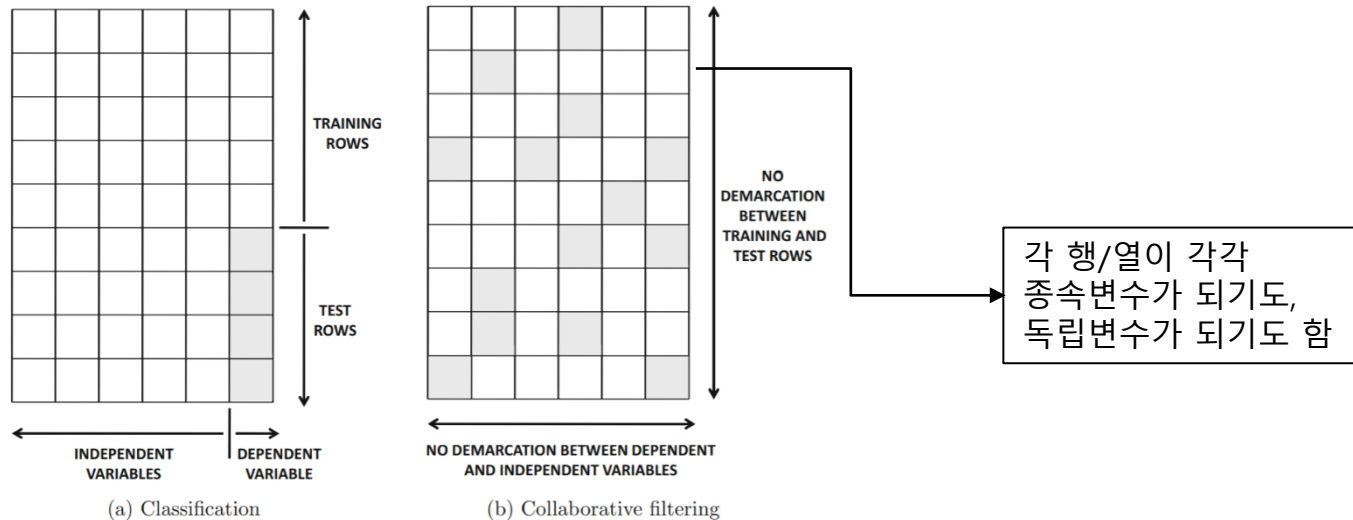
- 이웃기반 방식 협업필터링이 lazy learning method를 대표하는 KNN을 사용한 비모델적 방식이라면, 반대로 모델 기반 협업필터링은 대부분의 머신러닝 모델을 matrix completion 맥락으로 일반화 한 것으로 볼 수 있음 (즉, 이웃기반 CF -> KNN, 모델기반 CF -> 선형회귀모델)



# 01 Model Based Collaborative Filtering

## CF 문제와 전통적 분류/회귀 모델의 차이점 3가지

1. 독립변수와 종속변수간에 명확한 분리가 있다. Matrix Completion 문제에서는, 그런 게 없다.  
어떤 엔트리(entry, 입력값)가 예측 모델상에서 고려되냐에 따라,  
각 컬럼은 종속변수이기도 하면서 독립변수이기도 하다.

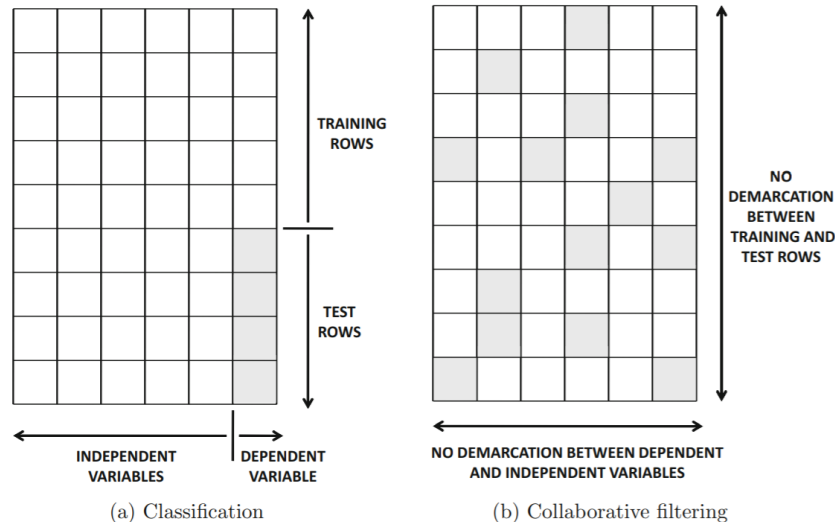


# 01 Model Based Collaborative Filtering

## CF 문제와 전통적 분류/회귀 모델의 차이점 3가지

2. 분류 문제에서는, 학습과 테스트 데이터의 분할이 명확하다. Matrix Completion 문제에서는, 그렇지 않다.  
(no clear demarcation among the rows)

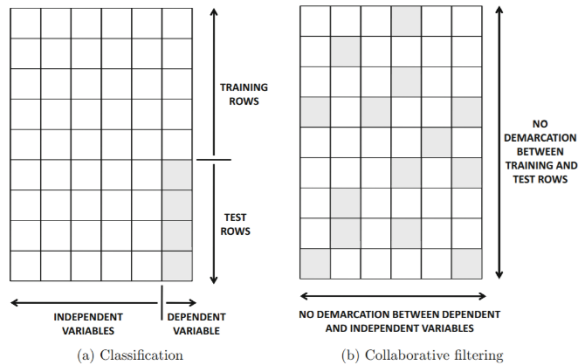
기껏 해야 특정한 관측된 엔트리를 학습 데이터로, 특정되지 않은 데이터를 테스트 데이터로 간주할 수 있을 뿐이다.



# 01 Model Based Collaborative Filtering

## CF 문제와 전통적 분류/회귀 모델의 차이점 3가지

3. 분류 문제에서는, 컬럼은 피처(feature, 특성)를 나타내고 열은 인스턴스를 나타낸다.  
그러나 Matrix Completion 문제에서는,  
전치 행렬이나 원행렬 모두에 같은 접근법을 사용하는 것이 가능하다.



$R^T$ 를 기준으로 MF를 하는 방법과  $R$ 을 기준으로 하는 방법 모두 가능

# 01 Model Based Collaborative Filtering

## 모델 기반 추천시스템의 장점

1. **Space-efficiency:** 보통은 학습된 모델은 원본 평점 행렬보다 훨씬 작다.  
따라서, 요구되는 공간이 상대적으로 적다.  
반면에, 유저 기준 이웃 기반 방식은  $O(m^2)$ 의 공간 복잡도를 가지고,  
아이템 기반 방식은  $O(n^2)$ 의 공간 복잡도를 가진다.
2. **Training speed and prediction speed:**  
이웃 기반 방식의 단점은 전처리 단계가 유저 혹은 아이템 수의 제곱이라는 점이다.  
반면에, 모델 기반 시스템은 학습된 모델을 만들 때 전처리 단계가 훨씬 빠르다.  
대부분의 경우에, 압축되고 요약된 모델이 예측을 효율적으로 수행하는 데 사용된다.
3. **Avoiding overfitting:** 분류나 회귀 문제에서 심각한 문제인 과적합 문제를 회피하는 데 효율적이다.  
또한 이러한 모델을 더욱 강건하게 만들기 위한 정규화(regularization) 방법이 적용될 수 있다.



Chapter



# Latent Factor Model

1. Latent Factor model이란 (feat. Matrix Factorization)
2. Geometric intuition

## 02 Latent Factor Model

### 2.1. Latent Factor model이란 (feat. Matrix Factorization)

잠재적인 요소를 추출하여 그 특성에 대한 상품과 유저의 연관도를 표현, 이를 기반으로 추천

1. 추상적 정리:  
rating matrix 내의 행간, 열간의 상관관계를 사용해,  
fully specified and reduced representation을 얻는 것.  
결과적으로, 원 행렬을 근사한 추정 평점 행렬을 한방에 구할 수 있음.

-> 무슨말인지 잘 안와닿음



## 02 Latent Factor Model

### 2.1. Latent Factor model이란 (feat. Matrix Factorization)

2. 이름 살펴보기

latent(잠재) factor(요소)

= latent(잠재) feature(특성)

= 어떤 대상을 설명하는 숨겨진 상태의 변수

= 뭔가 겉으로 안드러나는 특성이 있나보다!









## 02 Latent Factor Model


### 2.1. Latent Factor model이란 (feat. Matrix Factorization)

#### 3. 예시 들어보기

Matrix  
Factorization

	M1	M2	M3	M4	M5
 Comedy	3	1	1	3	1
 Action	1	2	4	1	3

	 Comedy	 Action
 A	✓	✗
 B	✗	✓
 C	✓	✗
 D	✓	✓

	M1	M2	M3	M4	M5
 A	3	1	1	3	1
 B	1	2	4	1	3
 C	3	1	1	3	1
 D	4	3	5	4	4










- 명수, 하하, 준하, 재석 4명의 유저
- 벚꽃엔딩, 가을편지, 눈의꽃, 봄사랑벚꽃말고, 첫눈 총 5곡의 노래
- 앞의 두 곡은 '봄노래' 뒤의 두 곡은 '겨울노래'라는 새로운 특성으로 표현 가능
- 그러나 이런 특성(factor)은 '감추어져(hidden, latent)' 있음
- 하지만 유저 4명이 보이는 반응을 토대로, 이러한 감추어진 특성들을 찾아내보자! 라는 접근


## 02 Latent Factor Model





### 2.1. Latent Factor model이란 (feat. Matrix Factorization)

#### 3. 예시 들어보기

Matrix  
Factorization

		
A		
B		
C		
D		

	M1	M2	M3	M4	M5
 Comedy	3	1	1	3	1
 Action	1	2	4	1	3

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

- $m \times n$  평점 행렬을  $m \times k, n \times k$ 로 쪼갬  
이는 각각 유저 특성 행렬, 아이템 특성 행렬이라 부름
- 이 때  $k$ 는 잠재차원의 크기  
즉, 아까의 예시에서는  $k=2$
- 유저 특성 행렬과 아이템 특성 행렬을 곱해 원래 행렬이 잘 복원된다면,  
잠재특성에 대한 유저의 유사도와 아이템의 유사도가 잘 함축되었다고 판단할 수 있음











## 02 Latent Factor Model



### 2.1. Latent Factor model이란 (feat. Matrix Factorization)





#### 3. 예시 들어보기

명수, 하하, 준하, 재석 4명의 유저  
벚꽃엔딩, 가을편지, 눈의꽃, 봄사랑벚꽃말고, 첫눈  
총 5곡의 노래

Matrix  
Factorization

		
A		
B		
C		
D		

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3

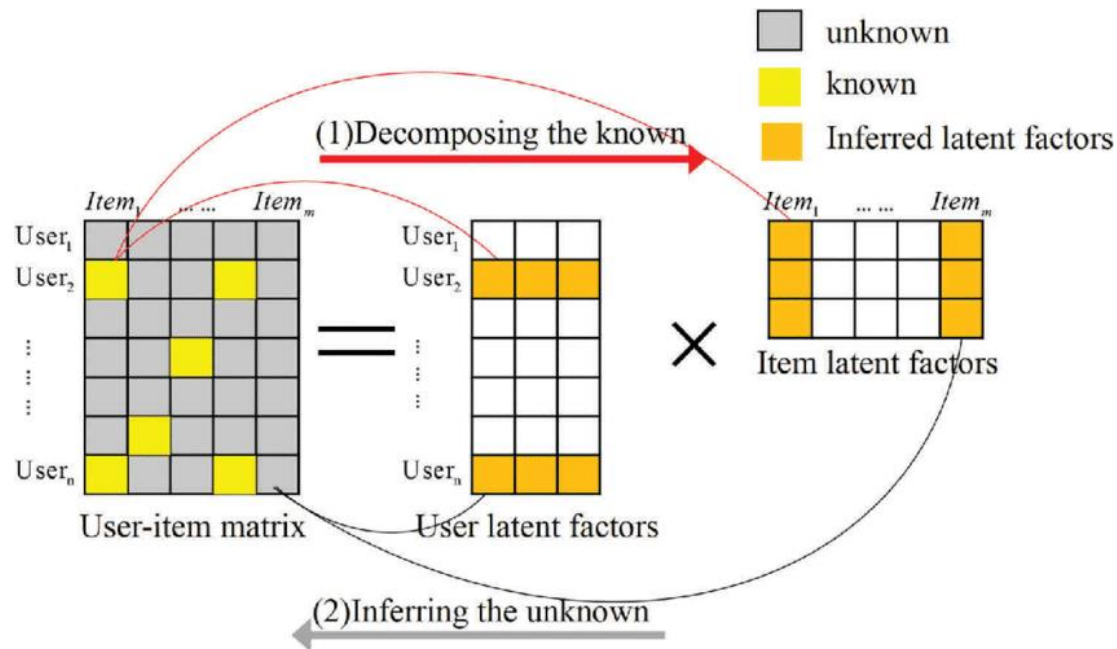
	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

- 가령 명수는 봄노래를 좋아하지만 겨울노래를 싫어하고  
하하는 봄노래를 싫어하지만 겨울노래를 좋아하고  
준하는 봄노래를 좋아하고 겨울노래를 싫어하고,  
재석은 봄노래와 겨울노래 둘 다 좋아한다고 해보자.
- 벚꽃엔딩은 봄점수 3, 겨울점수 1  
가을편지는 봄점수 1, 겨울점수 2  
눈의꽃은 봄점수 1, 겨울점수 4  
봄사랑벚꽃말고는 봄점수 3, 겨울점수 1  
첫눈은 봄점수 1, 겨울점수 3
- 이 두 매트릭스를 곱하면,  
특정 멤버의 (평가를 내리지 않은) 특정 노래에 대한  
예상 선호점수를 얻을 수 있음.

## 02 Latent Factor Model

### 2.1. Latent Factor model이란 (feat. Matrix Factorization)

이처럼 잠재 특성을 표현할 수 있도록,  
유저와 아이템의 행렬 곱으로 표현하는 방식을 Matrix Factorization이라고 함



## 02 Latent Factor Model

### 2.2. 잠재 요소 모델에 대한 기하학적 직관

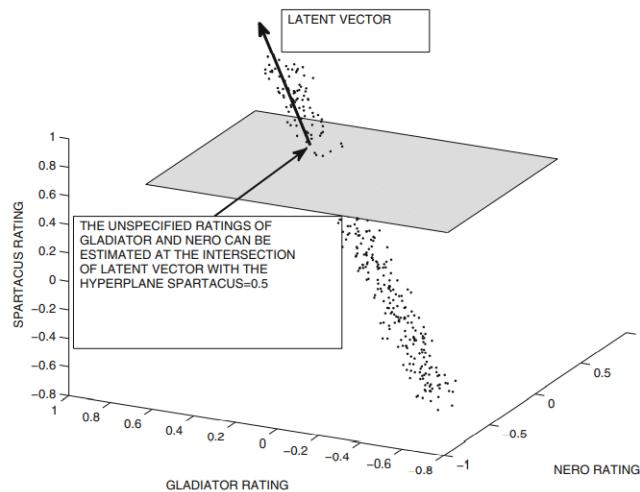


Figure 3.6: Leveraging correlation-based redundancies in missing data estimation for a user whose only specified rating is a value of 0.5 for the movie *Spartacus*

removing noisy variations), the data can be approximately represented on a  $p$ -dimensional hyperplane. In such cases, the missing ratings of a user can often be robustly estimated

- 노이즈 제거 후 1-rank로 approx되는 원래 3차원 데이터의 예를 보자.

- 이 경우 1차원 하이퍼플레인이 얻어지는데, 이 하이퍼플레인으로도 나머지 생략된 엔트리를 (강건하게) 추정해낼 수 있다는 것이다.

- 만일 스파르타쿠스의 평점이 0.5라는 값에 고정되면, 방금의 1차원 latent vector(i.e. 1차원 하이퍼플레인)이 축-평행 하이퍼플레인(spartacus=.5)과 교차하는 지점의 주변에 존재하는 데이터 포인트가 우리의 추정 값이라는 것

(적어도 그 주변에 있으리라 기대하는 것)



Chapter



# Optimization

1. What is Optimization
2. SGD (Stochastic Gradient Descent)
3. ALS (Alternative Least Squares)

## 03 Optimization

### 3.1. What is Optimization

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

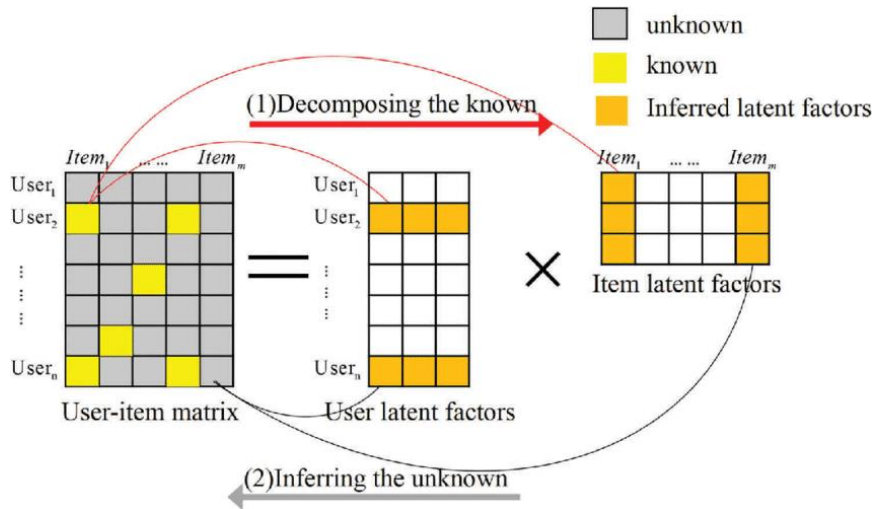
subject to:

No constraints on  $U$  and  $V$

- 일반적인 machine learning 문맥에서와 동일하게, 모델의 일반화된 예측 성능을 확보하기 위해 목적 함수(혹은 손실 함수)의 global minimum을 찾아가는 과정
- 위의 식에서  $R$ 은 원 평점행렬,  $UV^T$ 는 Matrix Factorization 모델의 구성 요소인  $U$ 행렬과  $V$ 행렬의 행렬곱
- 이 두 행렬의 Frobenius norm을 최소화해나가는 과정에서 학습이 이루어짐
- 그러나 우리의 원행렬은 많은 결측치를 포함하고 있기 때문에, 행렬차원의 목적함수가 정의되지 않음.
- 따라서 관측된 엔트리만을 가지고 목적함수를 정의하고, 이를 최소화해 나가는 방식으로 학습을 수행

# 03 Optimization

## 3.2. SGD (Stochastic Gradient Descent)



- 관측된 엔트리의  $u_{is}$ 와  $v_{js}$ 를 내적인 뒤, 이 값을  $r_{ij}$  (원래 값)에서 빼준 값을 error로 사용, 이를 minimize
- $U_{is}$ 와  $v_{js}$ 는 결정변수로,  $U$  행렬과  $V$  행렬의 행벡터를 나타냄. (잠재변수의 특성과의 연관성 내포)
- 이렇게 설정한 목적함수  $J$ 를 최소화하기 위해서는, 목적함수  $J$ 를  $u_{is}$ 와  $v_{js}$  기준으로 편미분해주어야 함.

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

subject to:

No constraints on  $U$  and  $V$

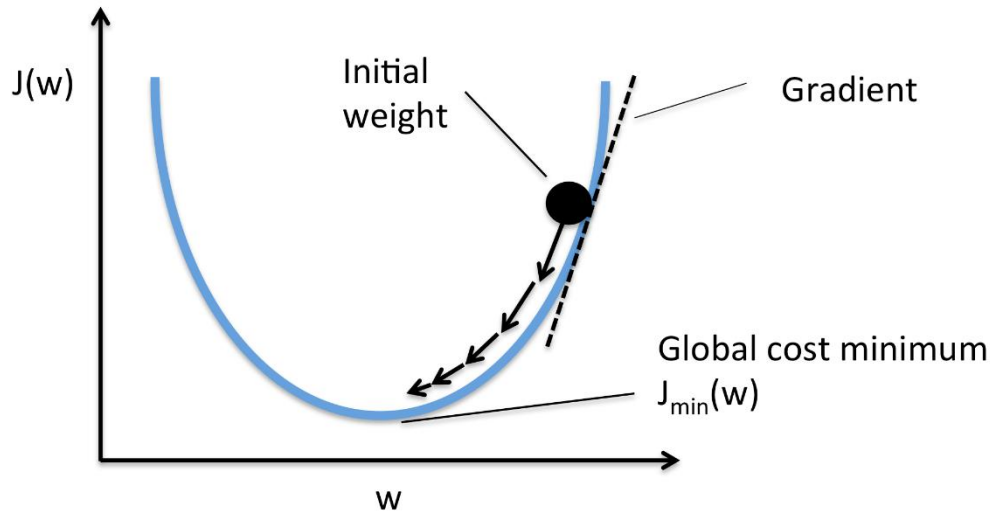
# 03 Optimization

## 3.2. SGD (Stochastic Gradient Descent)

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

subject to:

No constraints on  $U$  and  $V$



- 이 편미분 값은 결정 변수의 벡터 관점에서 기울기를 제공하고, 이 기울기에 일정한 learning rate parameter  $\alpha$ 를 곱해준 만큼을 업데이트함
- 이 과정을 수렴할 때까지 반복해 준다.
- 이 최적화 기법을 **Gradient Descent**이라 부름.

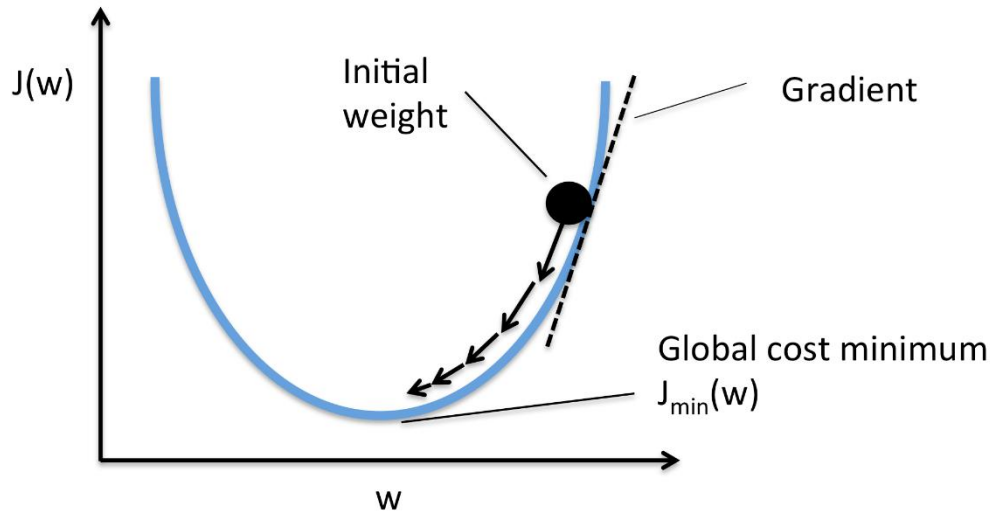
# 03 Optimization

## 3.2. SGD (Stochastic Gradient Descent)

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

subject to:

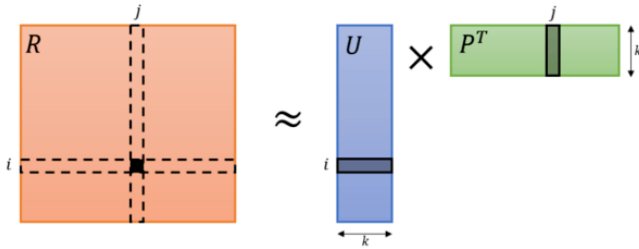
No constraints on  $U$  and  $V$



- **Stochastic Gradient Descent:**  
전체 batch를 기준으로 update를 수행하는 Gradient Descent 방식과는 달리, 각각의 엔트리에 대해 업데이트 하는 방식
- Batch update method (gradient descent)가 더욱 smooth하게 수렴하지만, SGD 방식이 더욱 빠르게 수렴함.

## 03 Optimization

### 3.3. ALS (Alternative Least Squares)



- Lr과 init 값에 크게 좌우받는 SGD 방식에 비해 좀 더 안정적이라고 알려진 방법.
- 병렬화가 가능한 특성때문에 추천시스템의 실구현에 많이 사용됨
- SGD와 동일하게, 2개의 행렬을 반복적으로 업데이트 하는 방식으로 학습이 이루어짐

#### [SGD 대비 ALS의 장점]

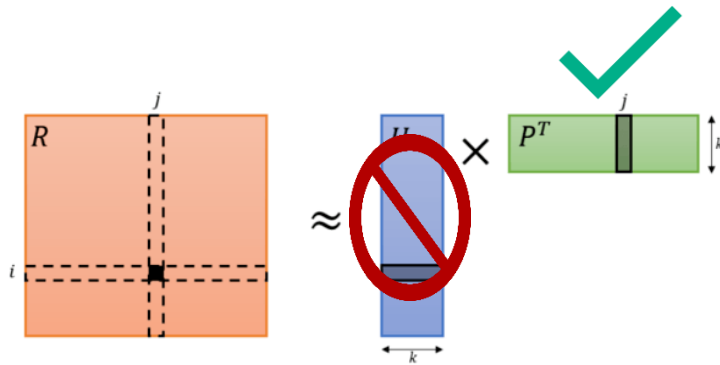
- 행렬의 모든 값이 0으로 fully specified 된 경우, 즉 implicit feedback 상황에서 weighted ALS가 강점을 보임. (non-zero entry를 더욱 가중)
- 대부분의 entry가 0일 경우, SGD방식은 비용이 큼.

#### [SGD 대비 ALS의 단점]

- large scale, explicit rating 상황에서는 SGD만큼 효율적이지는 않음.

# 03 Optimization

## 3.3. ALS (Alternative Least Squares)



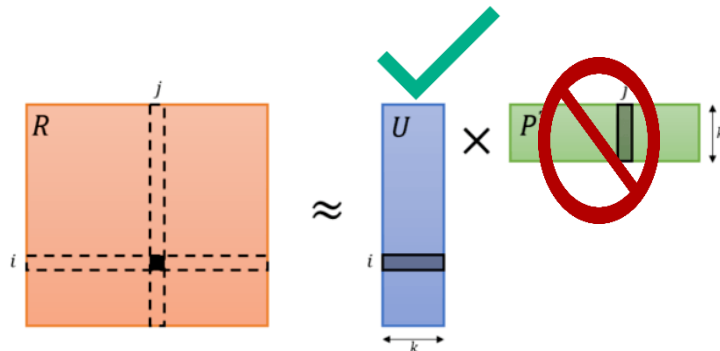
[학습 과정]

1.  $U$ 를 고정시키고,  $V$ 의  $n$ 개 행에 대해 Least-Squares regression problem을 해결함.

$U$ 의 모든 벡터를 상수 취급하고,  $V$ 의 모든 벡터를 최적화 변수로 간주함.

총  $n$ 개의 LS problem이 실행되어야하고, 각 LS problem은  $k$ 개의 변수를 가지고 있음.

각 상품에 대한 LS problem은 독립적이므로, 쉽게 병렬화 가능



2.  $V$ 를 고정시키고,  $U$ 의  $m$ 개 행에 대해 1과 같은 방법을 반복

Chapter



# SVD & NMF

1. SVD (Singular Value Decomposition)
2. NMF (Non-negative Matrix Factorization)



## 04 SVD & NMF

### 4.1. SVD (Singular Value Decomposition)

#### [Truncated SVD (Singular Value Decomposition)]

truncated SVD



[SVD]

임의의  $m \times n$  차원의 행렬  $R$ 을 다음과 같이 분해할 수 있는 행렬 분해 기법 중 하나.

$$R = UDV^T \rightarrow U_k D_k V_k^T$$

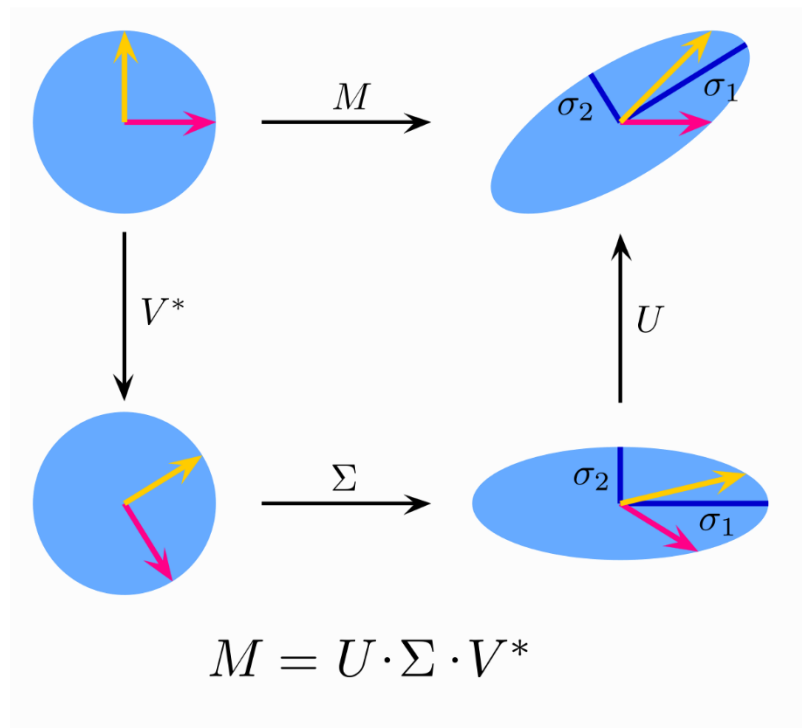
이 때, 가운데의  $\Sigma$ 는 특이값(singular values)들을 포함하는  $m \times n$  대각 행렬로, 대각 성분을 제외한 나머지 원소의 값은 0

이 특이값은 크기 순서로 대각으로 나열되어 있는데, 이는 scaling factor로써 특이 벡터의 중요도를 의미한다고 간주할 수 있음

$\Sigma$ 의 요소를  $k$ 개만 취하고,  $U$ 와  $V$ 의 열벡터도  $k$ 개만 취한 것을 truncated SVD라고 함.

## 04 SVD & NMF

### 4.1. SVD (Singular Value Decomposition)



[SVD의 기하적 의미]

$M$ 이라는 선형변환(matrix)는 rotate-stretch-rotate로 표현해 낼 수 있음.

$\Sigma$ 의 요소값들 (scaling factor)는 각 축의 방향으로의 stretch 강도를 의미함.

## 04 SVD & NMF

### 4.1. SVD (Singular Value Decomposition)

#### [Truncated SVD (Singular Value Decomposition)]

truncated SVD

$$A' = U_t \Sigma_t V_t^T$$

#### [Matrix Factorization에서의 SVD]

- U와  $\Sigma$  곱하거나, V와  $\Sigma$ 를 곱하여 2개의 행렬곱 형태로 만든다. 보통은 U에 곱해준다.
- 기존의 matrix factorization 접근에서, U의 column들이 mutually orthogonal하며, V의 column들도 역시 mutually orthogonal해야한다는 제약조건을 가진 채, 앞에서의 Frobenius norm을 최소화 하는 방식으로 학습함.

## 04 SVD & NMF

### 4.1. NMF (Non-negative Matrix Factorization)

[NMF (Non-negative Matrix Factorization)]

$$\begin{bmatrix} W \\ \times \\ H \end{bmatrix} \approx \begin{bmatrix} R \end{bmatrix}$$

[NMF]

- 기존의 matrix factorization 접근에서, U와 V가 각각  $\geq 0$ 이라는 제약조건을 걸고 최적화시킴.
- 이는 라그랑지언 방법으로 쉽게 최적화할 수 있음
- 모든 값이 양수이기 때문에, 해석상의 이점이 있음