

# 1. 추천 시스템 Matrix Factorization 의 구현 순서



데이터 탐색하기와 전처리		MF 모델		추천 시스템
Indexing	암묵적 평가	MF 모델	CSR	비슷한 아티스트 찾기
Unique() Map() Dropna()	-	Rating Matrix = Feature P x Feature Q	Sparse 데이터 유효한 데이터와 동일한 행렬 표현을 위한 CSR	Implicit()

## 추천이란?



## 아직 친구가 없나요?

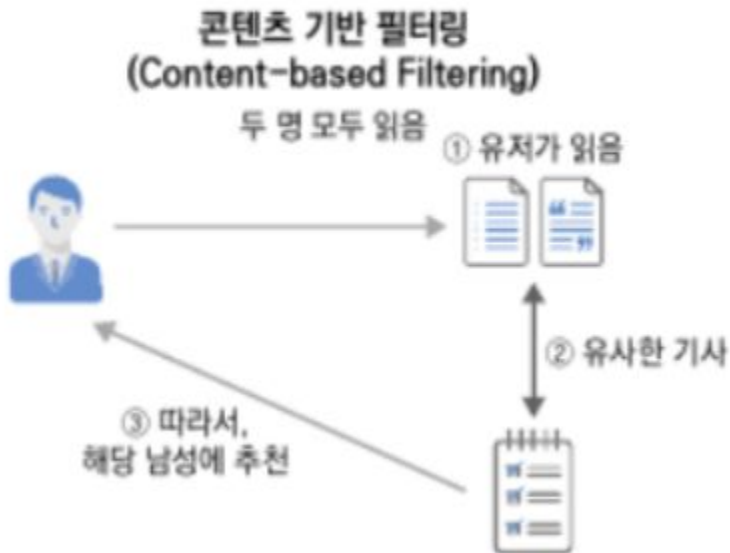
카카오뮤직 친구를 추가하고,  
유지름의 음악을 무료로 감상해보세요.



(정말 다양한 추천 서비스들)

## contents based filtering

“사용자와 아이템에 대하여 프로필을 작성하고, 이를 기반으로 추천하는 것을 의미합니다.”



user-based recommendation

(나와 같은 속성을 가진 사용자가 선호하는 것을 추천)

item-based recommendation

(컨텐츠에 속성을 기반으로 추천)

단점 : 데이터셋을 구성하기 어렵다.

## Collaborative Filtering?

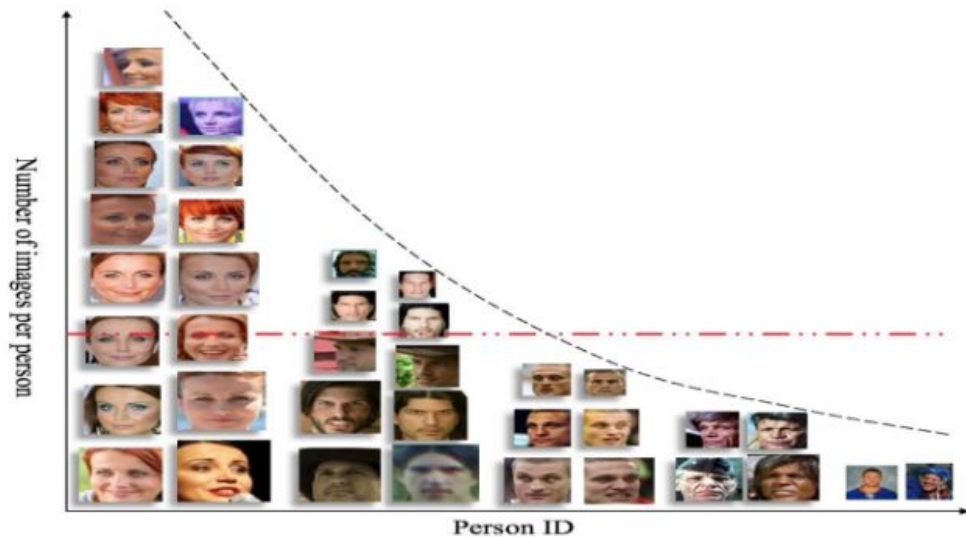
" 내가 남긴 데이터를 가지고 나와 취향이 비슷한 사람이 선호하는 아이템을 추천

장점

1. 도메인에 제약없이 데이터 셋을 쌓기 쉽다.
2. 일반적으로 **contents based filtering** 보다 정확하다.





단점

1. cold star문제
2. longtail문제
3. 계산량이 많음



## Collaborative Filtering?





"내가 남긴 데이터를 가지고 나와 취향이 비슷한 사람이 선호하는 아이템을 추천"

	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

## Collaborative Filtering?

" 빈칸이 있어도 주어진 평점 데이터만으로 평가되지 않는 아이템에 대한 평점을 예측하는 기법"

" 예측 평점이 높은 아이템을 추천해주는 방식"

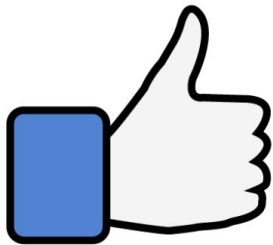
	M1	M2	M3	M4	M5
	?	1	?	3	?
	1	?	4	?	3
	3	1	?	?	1
	4	?	5	4	4

현실에서의 평점 행렬

## Collaborative Filtering?

평점데이터의 성격에 따라

### implicit dataset



### explicit dataset

시청시간, 반복횟수 등  
어떤 아이템을 비 선호하는지 알지는 못함

# Collaborative Filtering?

평점데이터의 성격에 따라

## implicit dataset

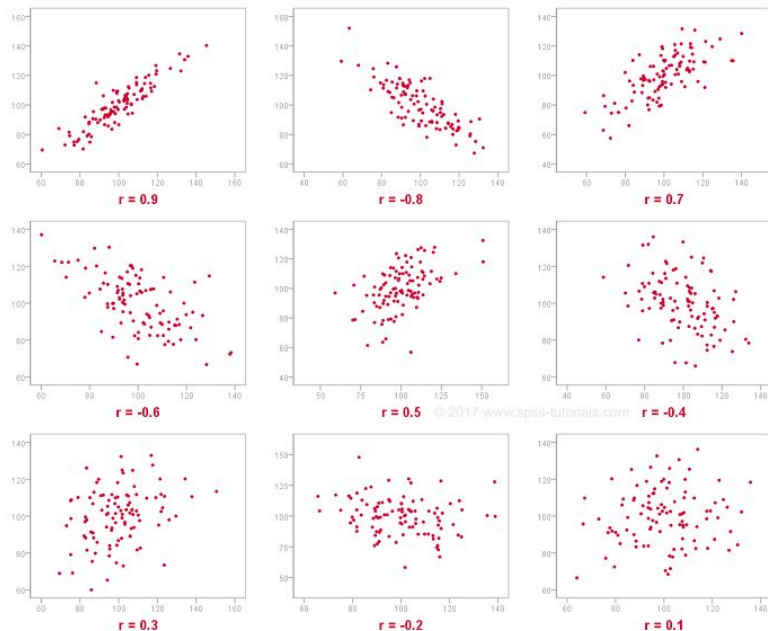
neighborhood model : 주어진 평점을 기반으로 비슷한 유저 혹은 아이টে를 추천  
User-oriented Neighborhood model

$$r_{XY} = \frac{\sum_{i=1}^n (X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^n (X_i - \overline{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \overline{Y})^2}}$$

피어슨 상관계수 [1]

양으로 변할 때 양으로 변하고,  
음으로 변할 때 같이 음으로 변하는 정도를 나타냄

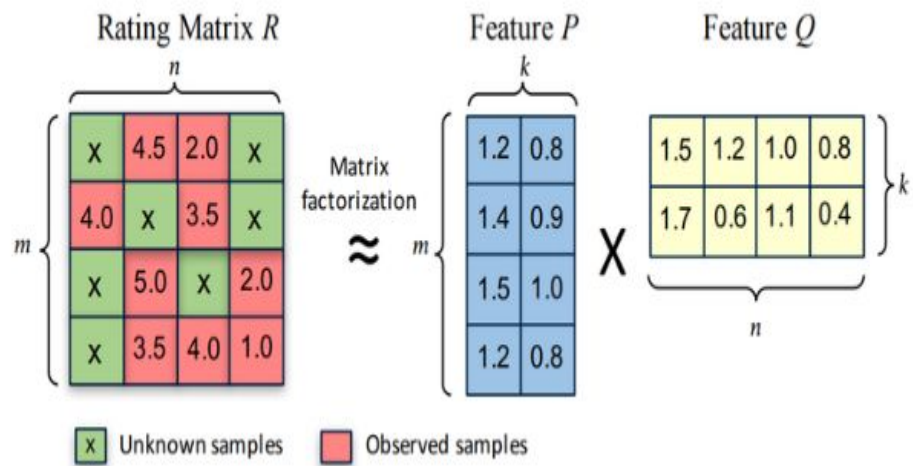
item-oriented Neighborhood model  
k개로 묶어줍니다.





# Matrix Factorization?

latent factor model : 관찰된 데이터와 잠재되어 있는 데이터를 연결시키는 기법



# Matrix Factorization?

## Sparse Matrix처리 > dense matrix

### 1. COO형식

0이 아닌 값, 행의 위치, 열의 위치

3	0	1
0	2	0

```
from scipy import sparse

# 0 이 아닌 데이터 추출
data = np.array([3,1,2])

# 행 위치와 열 위치를 각각 array로 생성
row_pos = np.array([0,0,1])
col_pos = np.array([0,2,1])

# sparse 패키지의 coo_matrix를 이용하여 COO 형식으로 희소 행렬 생성
sparse_coo = sparse.coo_matrix((data, (row_pos,col_pos)))

print(type(sparse_coo))
print(sparse_coo)
dense01=sparse_coo.toarray()
print(type(dense01),"\n", dense01)
```

# Matrix Factorization?

## Sparse Matrix처리 > dense matrix

### 1. CSR형식

0이 아닌 데이터 = [1, 5, 1, 4, 3, 2, 5, 6, 3, 2, 7, 8, 1]

행의 위치 = [0, 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5]

열의 위치 = [2, 5, 0, 1, 3, 4, 5, 1, 3, 0, 3, 5, 0]

0	0	1	0	0	5
1	4	0	3	2	5
0	6	0	3	0	0
2	0	0	0	0	0
0	0	0	7	0	8
1	0	0	0	0	0

```
sparse = np.array([[0,0,1,0,0,5],  
                  [1,4,0,3,2,5],  
                  [0,6,0,3,0,0],  
                  [2,0,0,0,0,0],  
                  [0,0,0,7,0,8],  
                  [1,0,0,0,0,0]])
```

```
coo = sparse.coo_matrix(sparse)  
csr = sparse.csr_matrix(sparse)
```

## Matrix Factorization?

Sparse Matrix처리 > dense matrix

### 1. CSR형식

```
data = np.array([1,3,5,6,2,11,7,9,10,12])
```

```
indices = np.array([1,2,3,0,1,3,1,2,2,3])
```

```
indptr = np.array([0,3,6,8,10])
```

```
data = [1,3,5,6,2,11,7,9,10,12]
```

```
row = [0,0,0,1,1,1,2,2,3,3]
```

```
col = [1,2,3,0,1,3,1,2,2,3]
```

## Matrix Factorization?

ALS: 교대 최소 제곱법

1. P와 Q벡터 중 하나를 고정해 놓고 교대로 계산을 합니다.
2. 분산처리 환경에서 빠르게 연산이 가능해집니다.
3. ALS는 **sparse**한 데이터에서 적절하다고 합니다.
4. 추천알고리즘의 **computation** 환경인 분산처리 플랫폼에서는 GD보다 효과적입니다.

<als추가설명>

<https://yeomko.tistory.com/8?category=805638>

<추천알고리즘 동향>

<http://hoondongkim.blogspot.com/2019/03/recommendation-trend.html>