

# 2020-1 Data Analysis with Applications

## Lecture 17. Recommender Systems 2

---

Department of Industrial and Information Systems Engineering,  
Soongsil University

# Collaborative Filtering

---

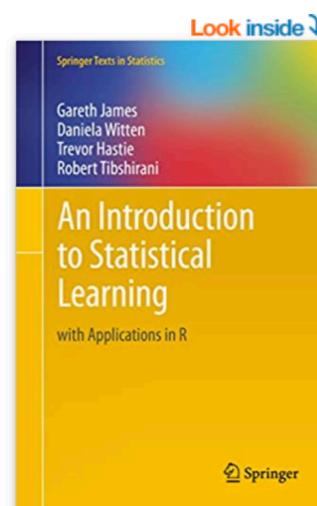
# Collaborative Filtering

- Collaborative filtering은 사용자 단위의 개별 추천을 위해 활용되는 기법으로, Netflix의 영화 추천, Amazon의 상품 추천, Google의 웹페이지 추천 등을 예로 들 수 있다.
- 특정 사용자와 유사한 행태 (구매, 평점매기기, 좋아요 클릭 등)를 보이는 다른 사용자들이 선호하는 item (상품, 영화, 음악, 도서, 웹페이지 등)을 추천하거나 (User-based collaborative filtering), 특정 사용자가 선호하는 item과 유사하게 선호되는 item을 추천한다 (Item-based collaborative filtering).

*At root, the retail giant's recommendation system is based on a number of simple elements: what a user has bought in the past, which items they have in their virtual shopping cart, items they've rated and liked, and what other customers have viewed and purchased. Amazon calls this homegrown math "item-to-item collaborative filtering", and it's used this algorithm to heavily customize the browsing experience for returning customers.*

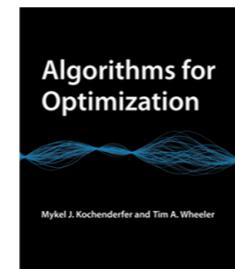
*"Amazon's Recommendation Secret (June 30, 2012)"*

# Collaborative Filtering



구매한 책

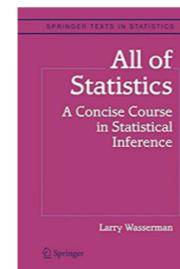
## Books you may like



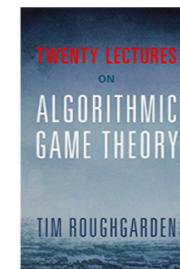
Algorithms for Optimization (The MIT Press)  
Mykel J. Kochenderfer  
★★★★★ 21  
Kindle Edition  
\$74.79



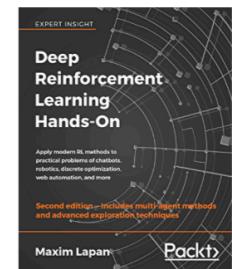
Foundations of Machine Learning (Adaptive Computation and...  
› Mehryar Mohri  
★★★★★ 8  
Kindle Edition  
\$65.99



All of Statistics: A Concise Course in Statistical Inference (Springer Texts...  
› Larry Wasserman  
★★★★★ 67  
Kindle Edition  
\$76.27



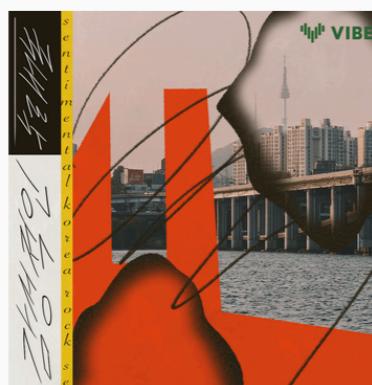
Twenty Lectures on Algorithmic Game Theory  
› Tim Roughgarden  
★★★★★ 7  
Kindle Edition  
\$19.64



Deep Reinforcement Learning Hands-On: Apply modern RL methods to...  
› Maxim Lapan  
★★★★★ 16  
Kindle Edition  
\$18.91

## [Item-based collaborative filtering (Amazon 책 추천 예)]

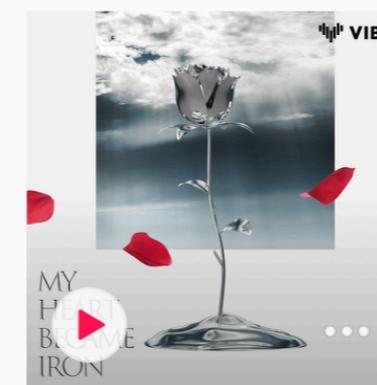
### 좋아할 것 같아서



감성적인 국내 락  
VIBE 국내 락/메탈



빠져든다 이 목소리  
VIBE 국내 알앤비/소울



한국의 락 발라드  
VIBE 국내 락/메탈



BULGOGIDISCO 대표곡  
VIBE



설레는 순간  
VIBE 발라드

## [User-based collaborative filtering (VIBE 음악 추천 예)]

# User-Based Collaborative Filtering

---

# User-Based Collaborative Filtering

→ 나와 흡사한 사용자

- User-based collaborative filtering (UBCF)은 다음 두 단계로 구성된다.
  - 1) 추천 대상인 사용자와 유사한 다른 사용자의 그룹 (neighbor)을 찾는다.
  - 2) 사용자가 아직 구매하지 않은 item들 중에서 사용자의 neighbor들이 선호하는 item을 추천한다.
- 다음 table은 6명의 사용자 ( $u_1 \sim u_6$ )가 8개의 영화 ( $i_1 \sim i_8$ )에 대해 1 ~ 5점 사이의 평점을 매긴 것이다. 이 예제를 대상으로 사용자  $u_a$ 에게 user-based collaborative filtering을 적용하여 영화를 추천해보자.

영화.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
$u_2$	3				5	1		
$u_3$	3			3	2	2		3
$u_4$	4			2	1	1	2	4
$u_5$	1	3						2
$u_6$		1			3	4		2
$u_a$			4	3		1		5

사용자



$r_{ij}$  : 사용자  $i$ 의 영화  $j$ 에 대한 평점

→ 나와 흡사한 사용자  
사용자  $u_a$  찾기

# User-Based Collaborative Filtering

정의: k개의 neighbor를 찾는다  
 거리 =  $knn - Euclidean distance$   
 UBCF: Similarity measure.

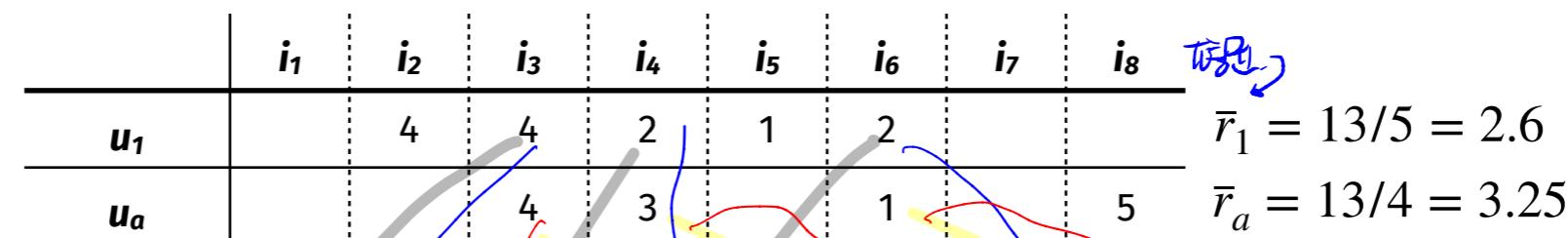
- UBCF에서 특정한 사용자의 neighbor를 찾는 방식은  $k\text{-nn}$ 에서 neighbor를 찾는 방식과 동일하지만, 유사한 정도를 나타내는 similarity measure로 Euclidean distance보다는 다음 두 척도를 사용할 때 일반적으로 성능이 더 좋아진다.

드시용자가  
영향을 미친다

- Pearson correlation:  $Corr(u_1, u_2) = \frac{\sum_j (r_{1,j} - \bar{r}_1)(r_{2,j} - \bar{r}_2)}{\sqrt{\sum_j (r_{1,j} - \bar{r}_1)^2} \sqrt{\sum_j (r_{2,j} - \bar{r}_2)^2}}$

- Cosine similarity:  $CosSim(u_1, u_2) = \frac{\sum_j r_{1,j} \cdot r_{2,j}}{\sqrt{\sum_j (r_{1,j})^2} \sqrt{\sum_j (r_{2,j})^2}}$

→ Pearson correlation이나 Cosine similarity를 알아보자!!



어떤 사용자는 평점을 주거나, 평점을 주거나 대체

$$\rightarrow Corr(u_1, u_a) = \frac{(4 - 2.6)(4 - 3.25) + (2 - 2.6)(3 - 3.25) + (2 - 2.6)(1 - 3.25)}{\sqrt{(4 - 2.6)^2 + (2 - 2.6)^2 + (2 - 2.6)^2} \sqrt{(4 - 3.25)^2 + (3 - 3.25)^2 + (1 - 3.25)^2}} \approx 0.653$$

$u_1 \times u_a$

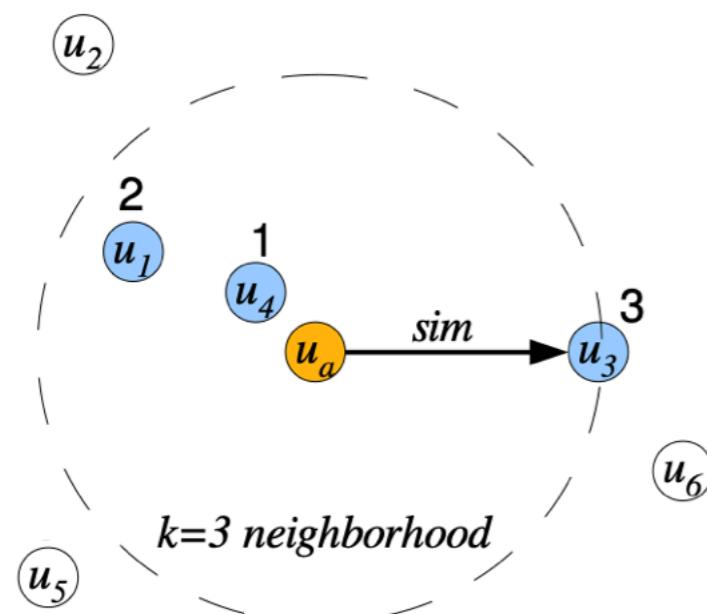
$$CosSim(u_1, u_a) = \frac{(4 \times 4) + (2 \times 3) + (2 \times 1)}{\sqrt{4^2 + 2^2 + 2^2} \sqrt{4^2 + 3^2 + 1^2}} \approx 0.960$$

그냥 단순히 절대적으로 높았는지 같은 예인.

# User-Based Collaborative Filtering

- 다음으로 similarity 값을 기준으로 k개의 neighbor (사용자)를 선택한다. 또는 similarity 값이 특정 threshold 이상인 neighbor를 선택할 수도 있다.

*k=3 = 3개의 높은 값 선택!!*



	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$	COS
$u_1$		4	4	2	1	2			0.960
$u_2$	3		2			3			0.739
$u_3$	3			3	2	2	3	0.936	
$u_4$	4			2	1	1	2	4	0.995
$u_5$	1			4				2	0.843
$u_6$		1			3	4		2	0.613
$u_a$			4	3		1		5	

- 예제에서 Cosine similarity를 기준으로  $k = 3$ 개의 neighbor를 선택하면  $u_1, u_3, u_4$ 가 포함된다.

# User-Based Collaborative Filtering

- 주천대상인 사용자의 특정 item에 대한 평점 예측은  $k$ 개의 neighbor가 해당 item에 매긴 평점의 평균으로 계산할 수 있다.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$	COS
$u_1$		4	4	2	1	2			0.960
$u_3$	3			3	2	2		3	0.936
$u_4$	4			2	1	1	2	4	0.995
$u_a$			4	3		1		5	
예측평점		3.5	4	-	-	1.33	-	2	
$u_a$ 가 보지 않은 평점 계산									

- 예측 평점을 구할 때,  $u_a$ 와의 similarity 값을 사용한 가중평균을 계산할 수도 있다. 예를 들어, 가중평균에 의한  $i_5$ 의 예측 평점은 다음과 같이 계산된다.

$$\left( \frac{1}{0.960 + 0.936 + 0.995} \right) \{ (0.960 \times 1) + (0.936 \times 2) + (0.995 \times 1) \} = 1.323$$

↳ 더 가까운 대상에 가중치를 더 높게 준다

- $u_a$ 에게 예측 평점이 높은  $i_2, i_1, i_7, i_5$ 의 순서대로 새로운 영화를 추천할 수 있다. 하지만 영화  $i_7$ 과  $i_5$ 는 예측 평점이 2 이하로 매우 낮기 때문에 실제로 추천하기는 무리일 것이다.

# User-Based Collaborative Filtering

사용자마다 기준이 다른 거예요!!



- 사용자 별로 평점의 분포가 다르기 때문에 (어떤 사용자는 평점이 매우 관대하기도 하고, 어떤 사용자는 평점을 매우 엄격하게 부여하기도 한다.) UBCF를 적용하기 전에 사용자 별로 평점을 표준화하기도 한다. 데이터의 특성에 따라 표준화로 인해 예측성능이 향상될 수도 있다.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
Z-score 표준화		1.4	1.4	-0.6	-1.6	-0.6		

$\frac{\text{평균} - \text{평균평균}}{\text{표준편차}}$

↑ 사용자 평균과 차이가 커요!

- UBCF의 단점으로는, 첫째 모든 사용자의 데이터베이스를 메모리에 가지고 있어야 하며, 둘째 사용자가 많을 경우 사용자 사이의 similarity 계산에 많은 시간이 걸릴 수 있다.
- 평점 데이터가 아닌, 상품의 구매 여부를 나타내는 데이터는 0 또는 1의 값을 가지며, 동일한 방식으로 UBCF를 적용할 수 있다.

# Item-Based Collaborative Filtering

---

# Item-Based Collaborative Filtering

- Item의 수보다 사용자의 수가 훨씬 많을 때에는 UBCF보다 item-based collaborative filtering (IBCF)를 사용하는 것이 계산적으로 더 효율적이다.
- IBCF는 다음 두 단계로 구성된다.
  - 1) 모든 Item 사이의 similarity를 계산한다.
  - 2) 계산된 similarity를 기반으로 추천대상인 사용자가 선호하는 item과 유사한 item을 추천한다.
- UBCF에서 사용한 영화 평점 예제를 계속해서 사용하자.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
$u_2$	3				5	1		
$u_3$	3			3	2	2		3
$u_4$	4			2	1	1	2	4
$u_5$	1	3						2
$u_6$		1			3	4		2

# Item-Based Collaborative Filtering

- 먼저 item 사이의 similarity를 계산하자. UBCF에서 사용한 Pearson correlation과 Consine similarity를 활용할 수 있다.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$i_1$	-	-	-	0.95	0.78	0.91	-	0.98
$i_2$	-	-	-	-	0.54	0.65	-	0.89
$i_3$	-	-	-	-	-	-	-	-
$i_4$	0.95	-	-	-	0.99	0.97	-	0.94
$i_5$	0.78	0.54	-	0.99	-	0.74	-	0.79
$i_6$	0.91	0.65	-	0.97	0.74	-	-	0.73
$i_7$	-	-	-	-	-	-	-	-
$i_8$	0.98	0.89	-	0.94	0.79	0.73	-	-

$$CosSim(i_4, i_5) = \frac{(2 \times 1) + (3 \times 2) + (2 \times 1)}{\sqrt{2^2 + 3^2 + 2^2} \sqrt{1^2 + 2^2 + 1^2}} \approx 0.99$$

[similarity 계산 예]

- Item의 수가  $n$ 개인 경우,  $n \times n$  similarity matrix를 만들 수 있으며, similarity matrix는 항상 symmetric이다.
- 두 item을 동시에 평가한 사용자가 0명이면 similarity를 계산할 수 없고, 1명이면 similarity 값은 항상 1이 되기 때문에, 이 두 경우에는 similarity를 계산하지 않는다.

# Item-Based Collaborative Filtering

- Item의 수가 매우 많을 경우, 저장공간과 계산시간을 줄이기 위해 각 item에 대해 similarity가 가장 높은  $k$ 개의 item만 사용할 수 있다.

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$i_1$	-	-	-	0.95	0.78	0.91	-	0.98
$i_2$	-	-	-	-	0.54	0.65	-	0.89
$i_3$	-	-	-	-	-	-	-	-
$i_4$	0.95	-	-	-	0.99	0.97	-	0.94
$i_5$	0.78	0.54	-	0.99	-	0.74	-	0.79
$i_6$	0.91	0.65	-	0.97	0.74	-	-	0.73
$i_7$	-	-	-	-	-	-	-	-
$i_8$	0.98	0.89	-	0.94	0.79	0.73	-	-

[ $k=3$ , 각 item별 similarity가 높은 3개의 item선택]

# Item-Based Collaborative Filtering

- 사용자에게 추천하기 위한 item을 선택하기 위해서, 사용자가 평점을 매기지 않은 영화들에 대해서 예측평점을 계산한다. 이때 item 간의 similarity 값을 가중치로 가지는 가중평균을 사용한다.

영화 1의 예측평점:  $\left(\frac{1}{0.95 + 0.91 + 0.98}\right)\{(0.95 \times 3) + (0.91 \times 1) + (0.98 \times 5)\} \approx 3.05$

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$	예측평점
$i_1$	-	-	-	0.95	0.91	-	-	0.98	3.05
$i_2$	-	-	-	-	0.54	0.65	-	0.89	3.31
$i_3$	-	-	-	-	-	-	-	-	-
$i_4$	0.95	-	-	-	0.99	0.97	-	-	-
$i_5$	0.78	-	-	0.99	-	-	-	0.79	3.89 추천
$i_6$	0.91	-	-	0.97	0.74	-	-	-	-
$i_7$	-	-	-	-	-	-	-	-	-
$i_8$	0.98	0.89	-	0.94	-	-	-	-	-
$u_a$			4	3	1			5	

# Item-Based Collaborative Filtering

- IBCF 알고리즘을 사용하면, 사용자  $u_a$ 에게 예측 평점이 높은  $i_5, i_2, i_1$ 의 순서대로 새로운 영화를 추천할 수 있다.
- 만약 사용자  $u_a$ 가 가장 높은 평점인 5점을 부여한 영화  $i_8$ 과 가장 유사한 영화를 추천하고자 한다면,  $i_8$ 과의 similarity가 가장 높은  $i_1$ 을 추천할 수 있을 것이다.
- 아래 table은 UBCF와 IBCF의 사용자  $u_a$ 에 대한 예측 평점을 비교한 것으로, 영화  $i_5$ 에 대한 두 알고리즘의 예측 평점은 차이가 매우 큰 것을 확인할 수 있다.

data가 커짐으로, 차이가 커지거나-

		$i_1$	$i_2$	$i_5$	$i_7$
USER	UBCF	3.5	4	1.33	2
ITEM	IBCF	3.05	3.31	3.89	-

# UBCF vs. IBCF

↗ 사용자가 누군지 상관없이 미리 계산하면 된다.

UBCF: 사용자가 추가되면 계속 계산해야 한다.

- IBCF에서는 similarity matrix를 미리 계산할 수 있다. 따라서 사용자에 대한 추천 시에는 예측평점 계산만 필요하기 때문에 계산적으로 UBCF에 비해 효율적이다.
- IBCF에 의해 추천된 item들은 UBCF에 의해 추천된 item들에 비해서 content가 서로 유사할 가능성이 매우 크다. 즉, 다양성이 부족할 수 있다. 예를 들어 영화의 경우 비슷한 장르의 영화들만 추천될 수 있다.  
↳ item이 비슷한 content일 수 있다.
- 일반적으로 IBCF 추천의 성능이 UBCF보다는 조금 떨어진다고 알려져 있다.
- UBCF와 IBCF를 포함하는 collaborative filtering은, 새로운 사용자나 item에 대해서는 similarity를 계산할 수 없기 때문에 알고리즘을 적용하는 데 한계가 있다.  
→ 처음 들어보면 평가점과 많지 않은데 (영화가 개봉되었을 때 사용자 평점이 차트에 넣힐 때)
- 실제 영화 및 음악 스트리밍, 도서 구매 및 대여 사이트 등에 가입할 때 사용자의 개별 선호 사항을 미리 선택하도록 하는 경우가 많다.

# Evaluation of Recommender Algorithms

- CF를 포함한 추천 알고리즘의 성능은 supervised learning에서의 성능평가와 유사한 방식으로 평가할 수 있다.
- 전체 사용자의 집합  $U$ 를  $U_{train}$ 과  $U_{test}$ 로 분할한 후에,  $U_{train}$  데이터를 활용하여  $U_{test}$ 에 속한 사용자 각각에 대해 예측 평점을 계산한다.

사용자를 train, test set으로

나누어서 성능을 비교!!

$U_{train}$

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
$u_2$	3				5	1		
$u_3$	3			3	2	2		3
$u_4$	4			2	1	1	2	4
$u_5$	1	3						2
$u_6$		1			3	4		2
$u_7$			4	3		1		5
$u_8$	2	4		2	2		3	1
$u_9$	3	4	5			3	4	
$u_{10}$		2	3	4			3	3

$U_{test}$

# Evaluation of Recommender Algorithms

- 이때  $U_{test}$ 에 속한 사용자에 각각에 대해  $x$ 개의 item을 랜덤으로 선택하여 알고리즘에 사용하고,  $x$ 개를 제외한 나머지 item들의 평점을 예측한다.
- 예측 평점과 실제 평점의 오차를 MAE 또는 RMSE 값으로 평가할 수 있다.

각 사용자당 3개의 평점을 선택.

$U_{test}$	2	4		2	2		3	1
$u_8$	2	4		2	2		3	1
$u_9$	3	4	5			3	4	
$u_{10}$		2	3	4			3	3

사용자마다  $x = 3$ 개의 item을 알고리즘에 사용

예측평점

	1.5	4		2.3	2		3	2.9
$u_8$	1.5	4		2.3	2		3	2.9
$u_9$	4.1	4	5			3	2.8	
$u_{10}$		2.7	3.5	4			3	3

- 위 예제에서 알고리즘의 평가 대상이 되는 7개의 평점에 대한 예측 오차는  $MAE = 0.886$ ,  $RMSE = 1.024$ 로 계산된다.  
 $RMSE$ : 각각의 차이에 제곱하고 다 더한 뒤 루트를 쓰우는거  
 $MAE$ : 그냥 각 차이의 평균.

# Personalized Movie Recommendations

---

# MovieLense DataSet

- MovieLense 데이터셋은 <https://movielens.org> 사이트를 통해 1997년 9월부터 1998년 4월까지 수집한 영화 평점 데이터로, 943명의 사용자가 1664개의 영화에 대해 매긴 평점 약 100,000개를 포함한다. 평점은 1 ~ 5 사이의 정수값이다.
- MovieLense 데이터셋을 활용하여 UBCF와 IBCF를 적용해보자.
- recommenderlab 패키지를 활용하여 collaborative filtering을 적용하고, 추천 성능을 평가할 수 있다.

```
# recommenderlab 패키지 사용
library(recommenderlab)

# 데이터 읽기
ratings_raw <- read.csv("movielens.csv")
str(ratings_raw)

## 'data.frame': 99392 obs. of 3 variables:
## $ user  : int 1 1 1 1 1 1 1 1 1 ...
## $ item  : chr "Toy Story (1995)" "GoldenEye (1995)" "Four Rooms (1995)" "Get Shorty (1995) ..."
## $ rating: int 5 3 4 3 3 5 4 1 5 3 ...
```

만개

영화

평점

# recommenderlab

- recommenderlab 패키지를 사용하기 위해서는 데이터프레임 형태의 데이터를 realRatingMatrix라는 객체로 변환해야 한다. 이 matrix에서 행은 사용자, 열은 item, matrix의 entry는 평점을 의미한다.

```
ratings_raw[1:3, ]  
  
##      user  
## 1      1  
## 2      1  
## 3      1  
  
# realRatingMatrix로 변환  
ratings <- as(ratings_raw, "realRatingMatrix")  
ratings  
  
## 943 x 1664 rating matrix of class 'realRatingMatrix' with 99392 ratings.  
  
# matrix 출력  
getRatingMatrix(ratings)[1:5, 1:5]  
  
## 5 x 3 sparse Matrix of class "dgCMatrix"  
##   'Til There Was You (1997) 1-900 (1994) 101 Dalmatians (1996)  
## 1          .          .          2  
## 2          .          .          .  
## 3          .          .          .  
## 4          .          .          .  
## 5          .          .          2
```

2개가 아닙니다.

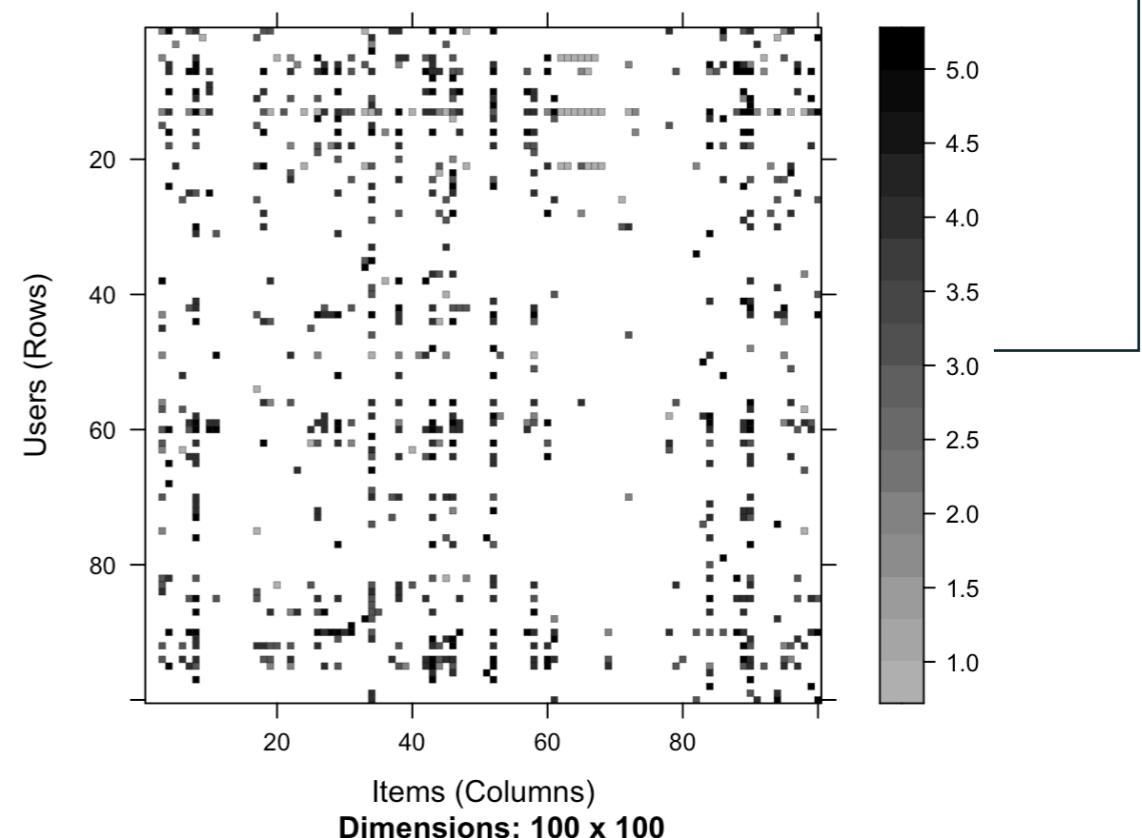
# realRatingMatrix

- ~~as() 함수를~~ 이용하여 realRatingMatrix를 다른 객체로 변환시킬 수 있다.

```
# 첫번째 사용자의 평점을 리스트로 저장  
as(ratings[1,], "list")
```

```
## $`1`  
## 101 Dalmatians (1996)  
## 2  
## 12 Angry Men (1957)  
## 5  
## 20,000 Leagues Under the Sea (1954)  
## 3
```

```
# rating matrix 를 matrix 형태로 저장  
m <- as(ratings, "matrix")  
  
# rating matrix 시각화  
image(ratings[1:100, 1:100])
```



# Visualization

- 각 사용자가 몇 개의 영화에 평점을 매겼는지 분포를 알아보자.

201개로 평점 많다.

```
# 사용자별 평점 수 확인
```

```
rowCounts(ratings[1:5])  
## 1 2 3 4 5  
## 271 61 51 23 175
```

```
# 사용자별 평점 수의 분포
```

```
summary(rowCounts(ratings))  
## Min. 1st Qu. Median Mean 3rd Qu. Max.  
## 19.0 32.0 64.0 105.4 147.5 735.0
```

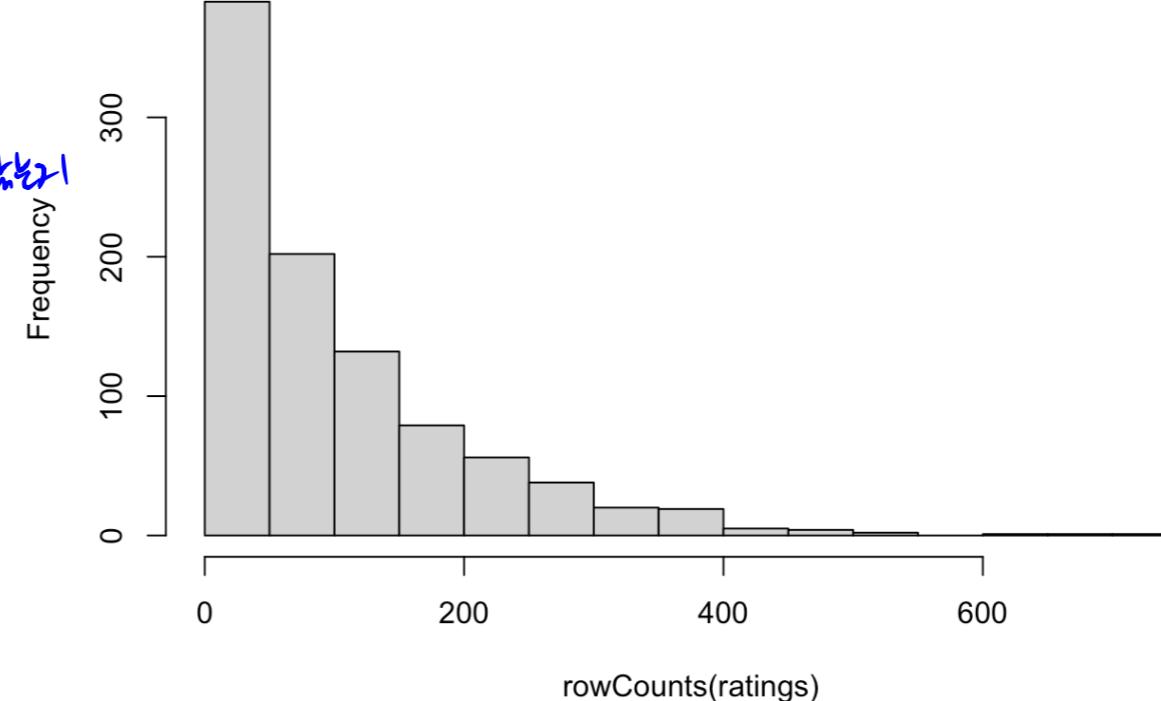
```
# 사용자별 평점 수 히스토그램
```

```
hist(rowCounts(ratings))
```

,break = 15  
!

colMeans --> 평균 평점을 높은 사람

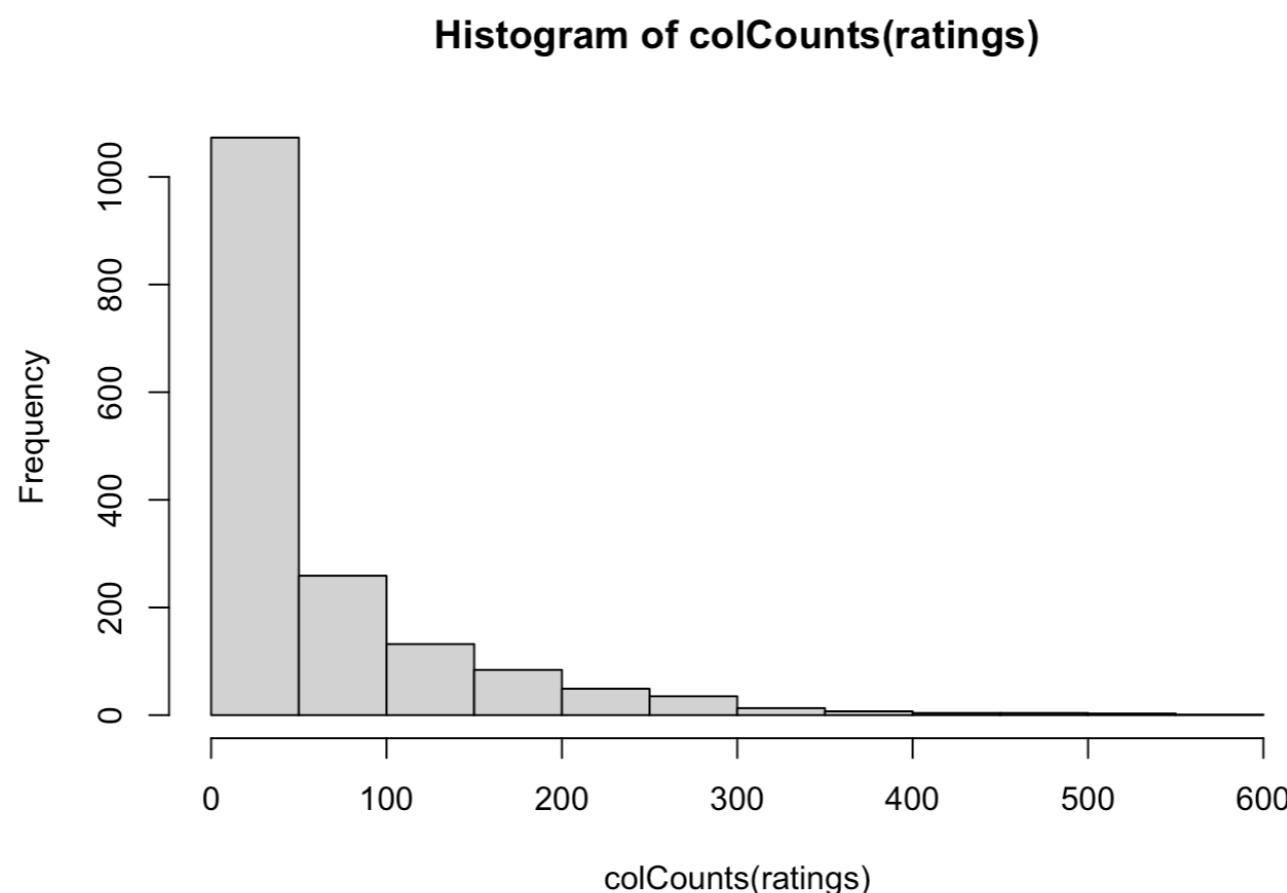
Histogram of rowCounts(ratings)



# Visualization

- 각 영화를 몇 명의 사용자가 평점을 매겼는지 분포를 알아보자.

```
# 영화별 평점 수 확인  
summary(colCounts(ratings))  
##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.  
## 1.00    7.00  27.00   59.73   80.00  583.00  
  
# 영화별 평점 수 히스토그램  
hist(colCounts(ratings))
```

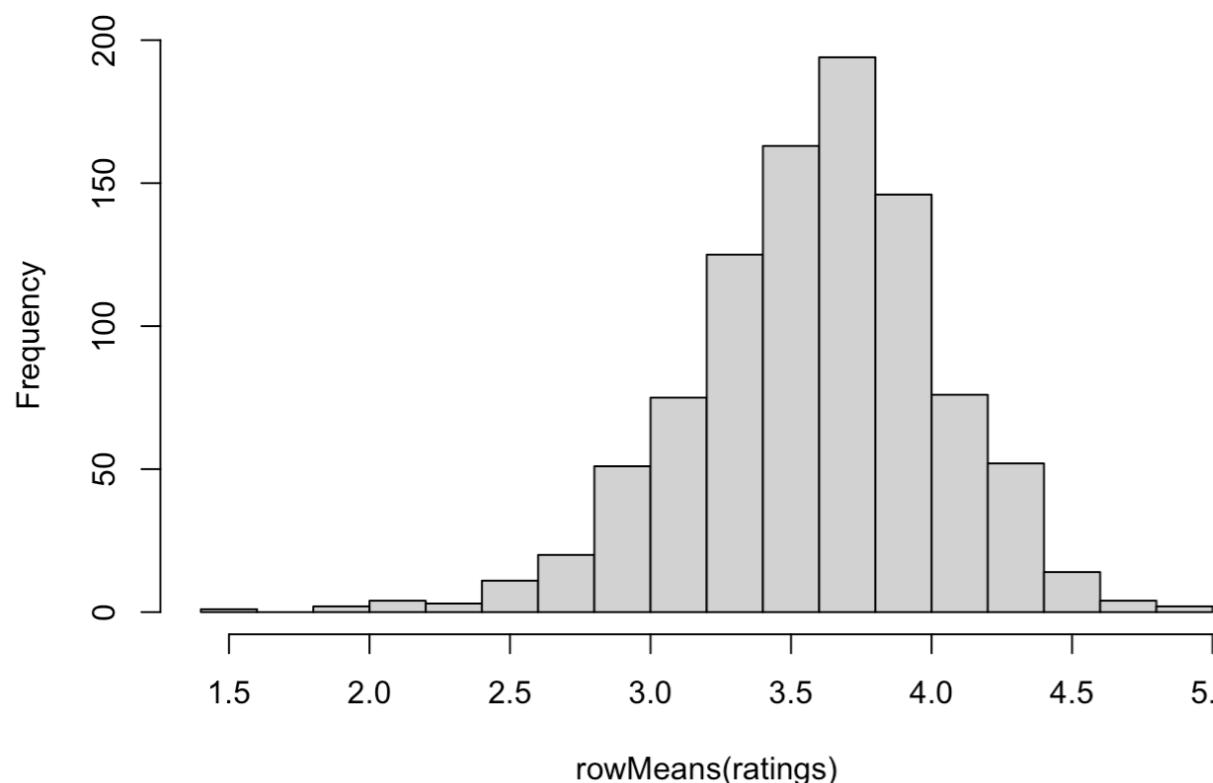


# Visualization

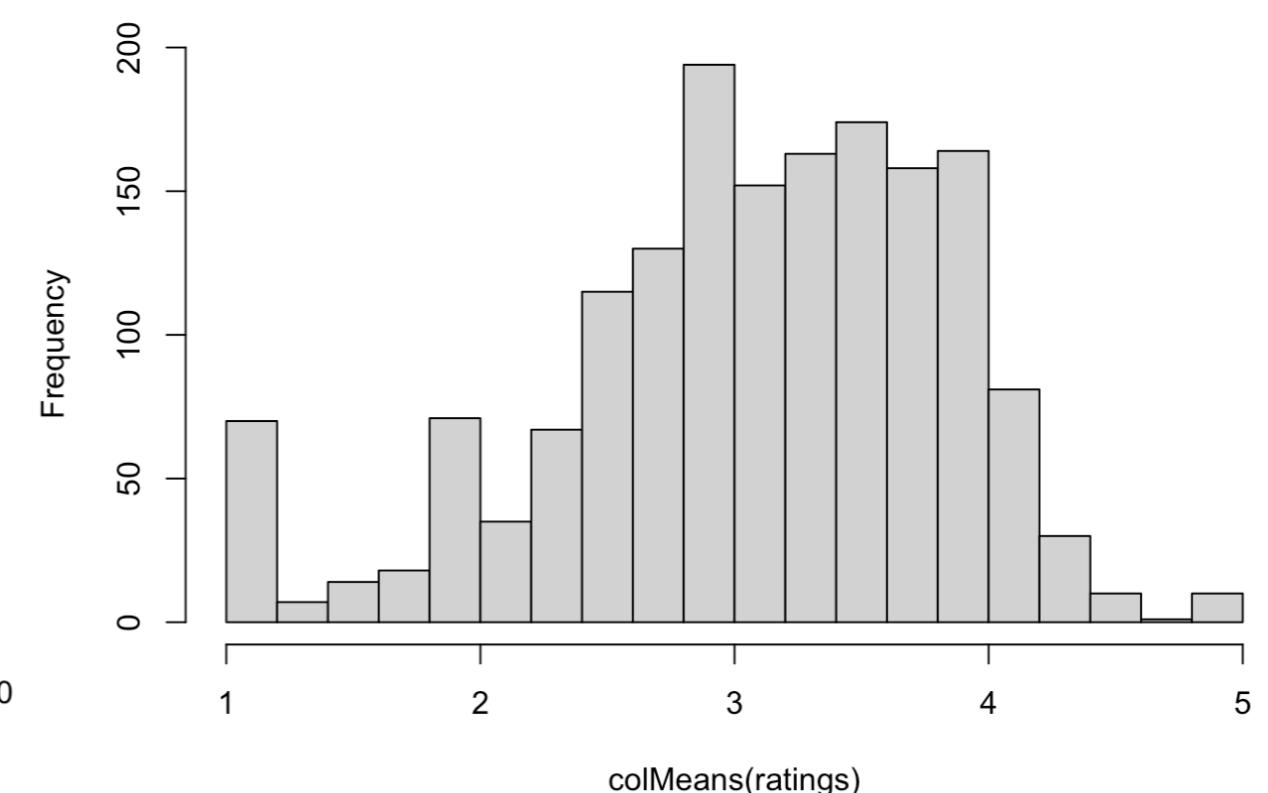
- 영화별, 사용자별 평균 평점의 분포를 시각화해보자.

```
# 사용자별 평균평점 히스토그램  
hist(rowMeans(ratings), breaks = 15)  
  
# 영화별 평균평점 히스토그램  
hist(colMeans(ratings), breaks = 15)
```

Histogram of rowMeans(ratings)



Histogram of colMeans(ratings)



# Recommender()

- Recommender() 함수를 사용하여 추천시스템을 정의할 수 있으며 다음과 같은 옵션을 사용한다.
  - method: 추천 알고리즘 ("UBCF" or "IBCF")
  - parameter: 알고리즘의 컨트롤 파라미터
    - method: similarity 척도 ("cosine" or "pearson" or "euclidean") - default = cosine  
*▶ 유사도 측정하기*      *▶ 표준화 방식과 평균편차로 나누기*
    - normalize: 표준화 방식 ("center" or "z-score") - default = center
    - nn: UBCF에서 각 사용자의 neighbor의 개수 - default = 25
    - k: IBCF에서 각 item별 similairy가 가장 높은 item의 저장 개수 - default = 30
- 1 ~ 940번째 사용자의 평점을 활용하여 UBCF 추천 시스템을 정의해보자.

```
# UBCF 추천 시스템 정의
r <- Recommender(ratings[1:940], method="UBCF",
parameter=list(method="cosine", nn=25, normalize = "center"))
```

설명: 평점 데이터를 활용한 추천 시스템 정의

설명: 평균편차를 활용한 추천 시스템 정의

# Predicting Top N Movies

- predict() 함수를 활용하여 새로운 사용자에 대한 추천을 수행한다. 옵션 `n = 5` 를 설정하면 사용자가 아직 관람하지 않은 영화 중에서 예측 평점이 가장 높은 5개의 영화를 추천한다.

# 941~943번째 사용자에 대해 5개의 영화 추천  
rec\_topN <- predict(r, ratings[941:943], n=5) → 예측하고자 하는 data

## Recommendations as 'topNList' with n = 5 for 3 users.

```
as(rec_topN, "list")
```

## \$`941`  
## [1] "Titanic (1997)" "L.A. Confidential (1997)"  
## [3] "Apt Pupil (1998)" "English Patient, The (1996)"  
## [5] "As Good As It Gets (1997)"  
##  
## \$`942`  
## [1] "Secrets & Lies (1996)" "Truth About Cats & Dogs, The (1996)"  
## [3] "Fargo (1996)" "Pillow Book, The (1995)"  
## [5] "Independence Day (ID4) (1996)"  
##  
## \$`943`  
## [1] "Titanic (1997)" "Contact (1997)"  
## [3] "L.A. Confidential (1997)" "Air Force One (1997)"  
## [5] "Gattaca (1997)"

# Predicting Ratings

- predict() 함수에서 옵션 type = "ratings"를 설정하면 사용자가 아직 관람하지 않은 모든 영화의 예측평점을 계산한다.

```
# 941~943번째 사용자에 대해 영화 평점 예측
rec_rating <- predict(r, ratings[941:943], type="ratings")
rec_rating  
→ 모든 평점을 예측
```

## 3 x 1664 rating matrix of class 'realRatingMatrix' with 4725 ratings.

```
as(rec_rating, "matrix")[,1:5]
```

```
##      'Til There Was You (1997) 1-900 (1994) 101 Dalmatians (1996)
## 941          4.045455    4.045455        4.030382
## 942          4.259740    4.258383        4.235638
## 943          3.378795    3.410714        3.410714
##      12 Angry Men (1957) 187 (1997)
## 941          4.088858    4.045455
## 942          4.259740    4.310317
## 943          3.410714    3.458506
```



```
# 예측평점을 데이터프레임으로 변환
```

dataframe으로 바꾸기.

```
rec_df <- as(rec_rating, "data.frame")
```

```
# 영화 "Toy Story"에 대한 예측평점
```

```
subset(rec_df, item == "Toy Story (1995)")
```

```
##       user           item   rating
## 4334  942 Toy Story (1995) 4.334813
## 4335  943 Toy Story (1995) 3.445581
```

→ 어떤 번호는 이미 봐서 예측이 안된 것

# Performance Evaluation

- `evaluationScheme()` 함수를 사용하여 추천 알고리즘의 성능을 평가하기 위한 scheme을 설정한다.
  - `method`: 성능평가 방법
  - `train`: 전체 사용자 중 training set에 포함되는 사용자의 비율 *test set 사용자 비율 설정* ↗ 15개를 랜덤으로 고른 사용자를
  - `given`: test set에 포함되는 사용자가 매긴 평점 중에서 추천 알고리즘에 제공하는 평점의 개수 (나머지 평점은 알고리즘의 성능 평가에 사용된다.)

```
# 전체 사용자의 90%의 training set으로 사용하며, test set에 속한 각 사용자에 대해 15개의 평점을 랜덤으로 알고리즘에 제공한다. training data train, test 비율.
```

```
e <- evaluationScheme(ratings, method="split", train = 0.9, given = 15)
```

```
e  
## Evaluation scheme with 15 items given  
## Method: 'split' with 1 run(s).  
## Training set proportion: 0.900  
## Good ratings: NA  
## Data set: 943 x 1664 rating matrix of class
```

$U_{train}$

*15개 평점으로 만든 예측평점*

$U_{test}$

*2개를 뺀 나머지 평점에 대해서 예측평점 제공*

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
$u_2$	3				5	1		
$u_3$	3			3	2	2		3
$u_4$	4			2	1	1	2	4
$u_5$	1	3						2
$u_6$		1			3	4		2
$u_7$			4	3		1		5
$u_8$	2	4		2	2		3	1
$u_9$	3	4	5			3	4	
$u_{10}$		2	3	4		3	3	3

given = 2인 경우, 사용자 별 2개의 평점이 알고리즘에 제공된다. 30

# Performance Evaluation

- UBCF와 IBCF 두 추천 알고리즘의 성능을 비교해보자.
- 먼저 Recommender() 함수를 이용하여 두 추천 시스템을 정의하자.
- `getData(e, "train")` 은 앞서 설정한 evaluation scheme에 의해 만들어진 training set을 의미한다.

```
# UBCF 추천 시스템 정의
r_ubcf <- Recommender(getData(e, "train"), "UBCF")
r_ubcf

## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 848 users.

# IBCF 추천 시스템 정의
r_ibcf <- Recommender(getData(e, "train"), "IBCF")
r_ibcf

## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 848 users.
```

# Performance Evaluation

- 다음으로 predict()를 사용하여 test set에 속한 사용자에 대한 예측 평점을 계산한다.  
*getData(e, "known")은 앞서 설정한 evaluation scheme에 의해 만들어진 test set에서 알고리즘에 제공되는 평점 데이터를 의미한다.*

```
# test set에서 알고리즘에 제공되지 않은 평점 데이터에 대한 예측 평점 계산 (UBCF)
p_ubcf <- predict(r_ubcf, getData(e, "known"), type="ratings")
p_ubcf

## 95 x 1664 rating matrix of class 'realRatingMatrix' with 156655 ratings.

# test set에서 알고리즘에 제공되지 않은 평점 데이터에 대한 예측 평점 계산 (IBCF)
p_ibcf <- predict(r_ibcf, getData(e, "known"), type="ratings")
p_ibcf

## 95 x 1664 rating matrix of class 'realRatingMatrix' with 17259 ratings.
```

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$		4	4	2	1	2		
$u_2$	3				5	1		
$u_3$	3			3	2	2		3
$u_4$	4			2	1	1	2	4
$u_5$	1	3						2
$u_6$		1			3	4		2
$u_7$			4	3		1		5
$u_8$	2	4		2	2	3	1	
$u_9$	3	4	5		3	4		
$u_{10}$		2	3	4		3	3	

**$U_{train}$**

**$U_{test}$**

← 알고리즘에 제공되지 않은 평점에 대해 예측평점을 계산함

# Performance Evaluation

- 마지막으로 `calcPredictionAccuracy()` 함수를 사용하여 test set에 속한 사용자의 실제 평점과 예측 평점 간의 오차를 계산한다.
- `getData(e, "unknown")` 은 test set에서 추천 알고리즘에 제공되지 않은 평점 데이터를 의미한다.

```
# test set에서 알고리즘에 제공되지 않은 평점 데이터에 대한 예측 평점 계산 (UBCF)
calcPredictionAccuracy(p_ubcf, getData(e, "unknown"))

##          RMSE         MSE         MAE
## 1.0081935 1.0164542 0.8060362

# test set에서 알고리즘에 제공되지 않은 평점 데이터에 대한 예측 평점 계산 (IBCF)
calcPredictionAccuracy(p_ibcf, getData(e, "unknown"))

##          RMSE         MSE         MAE
## 1.3613713 1.8533317 0.9855816
```

- RMSE와 MAE를 비교했을 때 UBCF의 성능이 IBCF보다 더 우수하다.