2018605059 정수연

추천 시스템

목 차

추천 시스템

- 추천 시스템 이란?

- 사용자(user)에게 상품(item)을 제안하는 소프트웨어 기술

Ex) 어떤 상품을 구매할 것인지, 어떤 음악을 들을 것인지 등 의사결정에 연관

- 추천 시스템 목표

- 사용자에게 상품을 어떻게 추천할 것인지

추천 타입

- 수작업
  - 즐겨찾기 목록
- 단순집계
  - Top 10, 최신곡
- 개별 사용자에 맞게

- 왓차, 넷플릭스, 유튜브 …

https://www.youtube.com/watch?v=4hsV1Dwlu_s

추천 시스템 만들 때 3가지 문제!

① Matrix를 어떻게 만들 것인가?

② 빈칸을 어떻게 채울 것인가?

③ 맞게 채웠는지 어떻게 평가 할 것인가?

	아바타	설국열차	어바웃타임	스파이더맨
사람1	1		0.2	
사람2		0.5		0.3
사람3	0.2		1	
사람4				0.4

① Matrix를 어떻게 만들 것인가?

- 직설적 방법

- 평가 부탁하기 ex) 리뷰

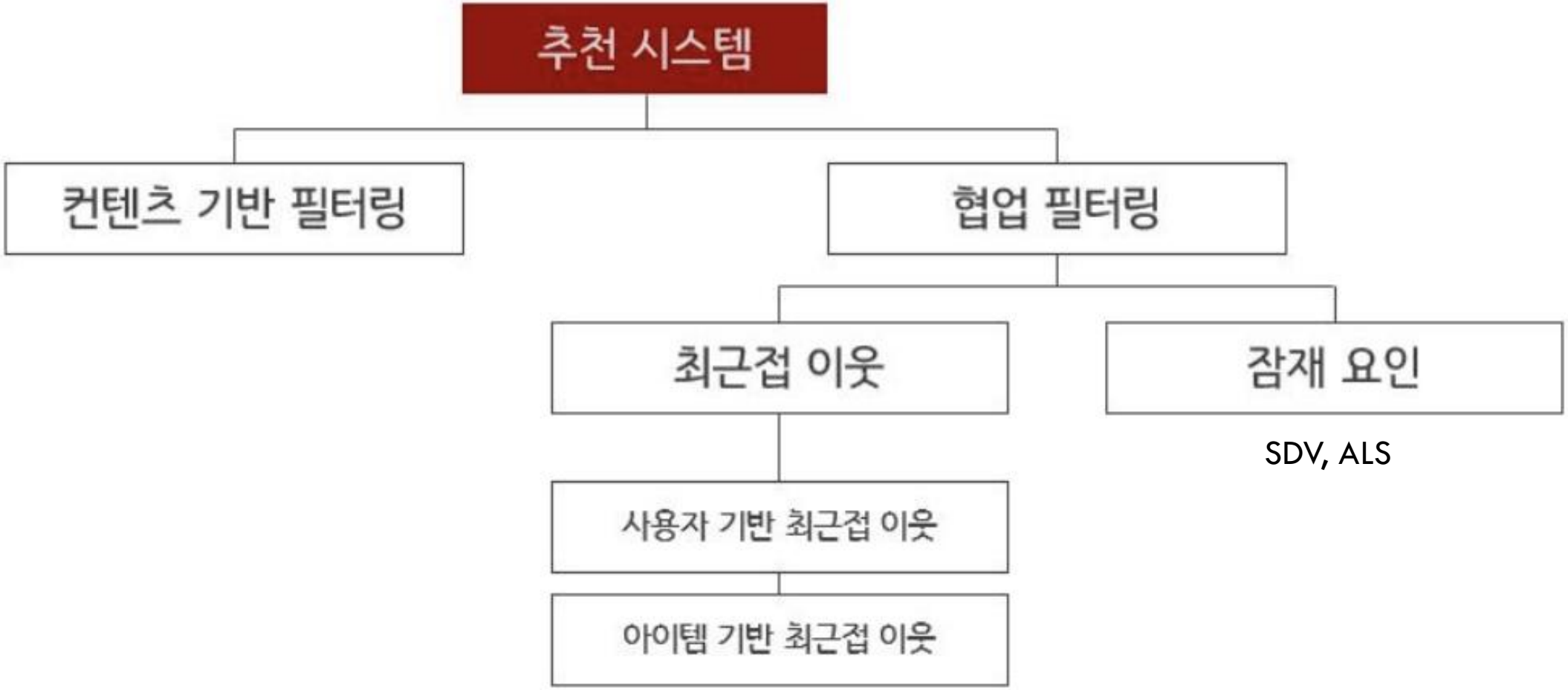
  단점 : 귀찮아서 잘 안함. → 포인트 지급

- 간접적 방법

- 자주 보는 작품 좋게 평가하기

  단점 : 랜덤으로 재생된 작품을 바로 넘겼는데 자주 봤다고 판단 가능.

② 빈칸을 어떻게 채울 것인가?



SDV, ALS

내용 기반 필터링

- Main idea : 고객X가 과거에 높은 점수 주었던 아이템과 비슷한 아이템 추천

  Ex) 비슷한 장르, 같은 감독

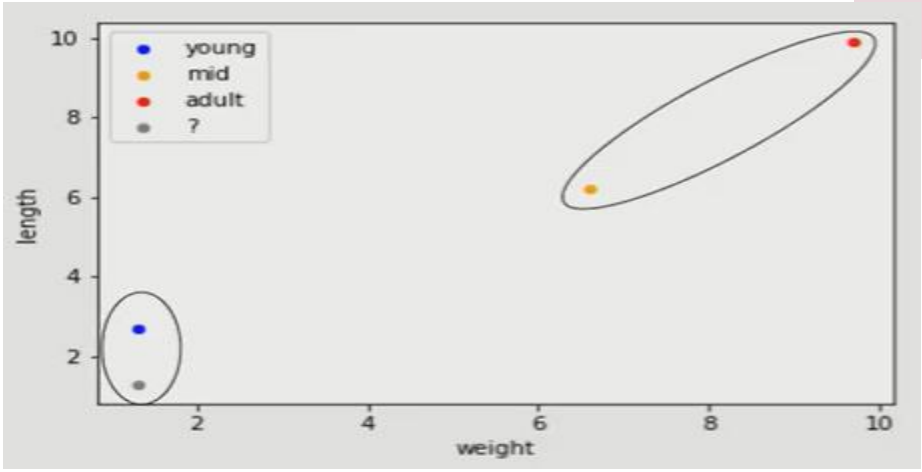
유사도 계산 방법

- 유클리디안 유사도

$$= \frac{1}{\sqrt{\sum_{i=1}^{p}(a_i - b_i)^2} + e^{-5}}$$

유클리디안 거리

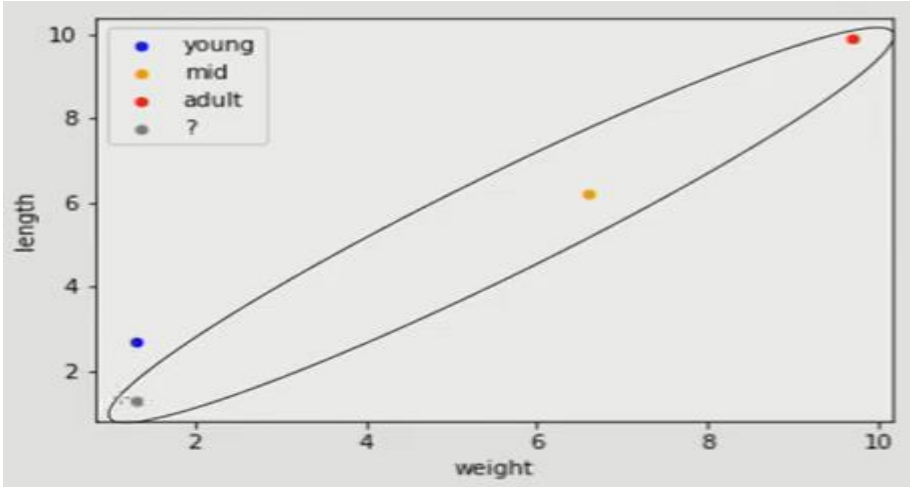
- 거리 기반 유사도 측정 방법
- 스케일이 비슷할 때 사용



- 코사인 유사도

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- 벡터A와 벡터B의 각도로 계산(-1~1사이 값을 가짐)
- 스케일이 차이날 때 사용



유사도 계산 방법

피어슨 유사도

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

- 벡터 x,y 사이의 상관관계를 계산
- 각 벡터의 표본평균으로 벡터를 정규화하고 코사인 유사도를 구함

자카드 유사도

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- 집합 개념을 이용해 계산
- 얼마나 많은 아이템이 겹치는지로 판단

Ex)

A 유저의 주식 종목

삼성전자, 테슬라, LG전자, 카카오, 펄어비스

B 유저의 주식 종목

삼성전자, 카카오, 넷마블, 현대자동차, 셀트리온

$$= \frac{2}{5 + 5 - 2}$$

$$= \frac{2}{8} = \frac{1}{4}$$

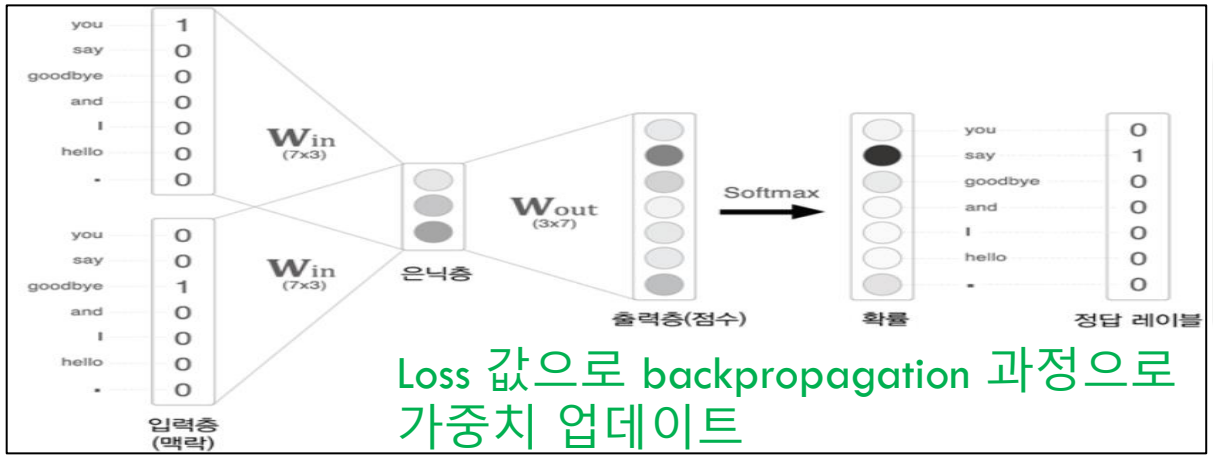
백터형태로 변형 방법

- TF-IDF (통계 기반 기법) → 특정문서에서만 자주 등장하는 단어 파악

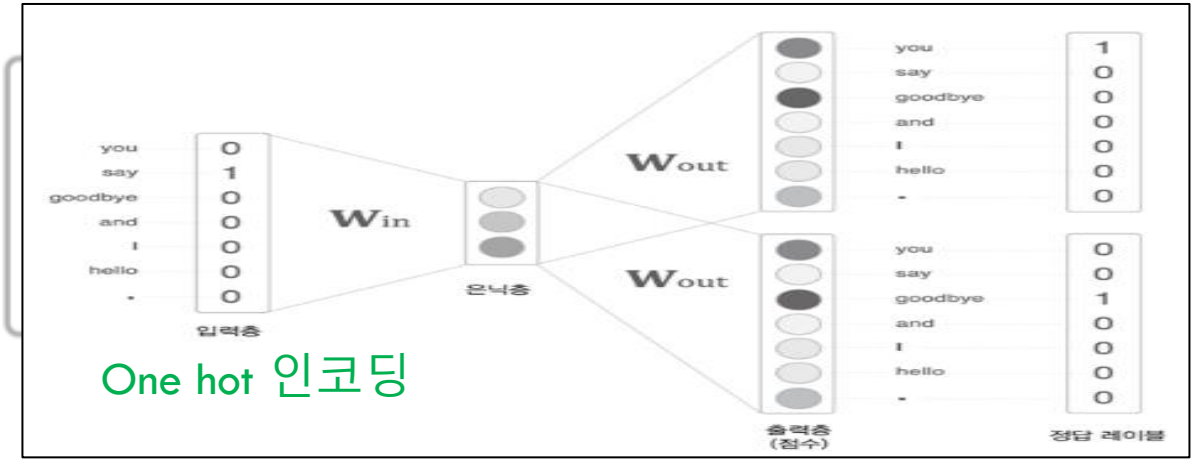
  TF-IDF = TF(d,t) * IDF(d,t)

  TF(d,t) : 특정 문서 d에서 특정 단어 t의 등장 횟수
  IDF(d,t) : 특정 단어 t가 등장한 문서의 수에 반비례하는 수

- Word2Vec (추론 기반 기법) → 비슷한 위치에 등장하는 단어들은 비슷한 의미로 가정



Loss 값으로 backpropagation 과정으로 가중치 업데이트



One hot 인코딩

내용 기반 필터링

- 예시(TF-IDF)

```
data = pd.read_csv(path + 'movies_metadata.csv', low_memory=False)
data.head(2)
```

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	...	release_c
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	...	1995-10-30
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	...	1995-12-15

2 rows × 24 columns

* Overview의 항목추출

```
data.columns
```

```
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count'],
      dtype='object')
```

내용 기반 필터링

▶ 예시(TF-IDF)

* Overview에 대해 tf-idf 수행

```
tfidf_matrix = tfidf.fit_transform(data['overview'])
print(tfidf_matrix.shape)
```

```
(20001, 47665)
```

→ 20001개 문서에 대해 47665토큰 가짐

* movie title와 id 매핑 dictionary 생성

```
movie2id = {}
for i, c in enumerate(data['title']): movie2id[i] = c
```

```
id2movie = {}
for i, c in movie2id.items(): id2movie[c] = i
```

* 코사인 유사도 구함

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_matrix = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

```
cosine_matrix.shape
```

```
(20001, 20001)
```

내용 기반 필터링

▶ 예시(TF-IDF)

* Toy story와 유사한 상위10개 인덱스

```
# Toy Story의 id 추출
idx = id2movie['Toy Story'] # Toy Story : 0번 인덱스
sim_scores = [(i, c) for i, c in enumerate(cosine_matrix[idx]) if i !
= idx] # 자기 자신을 제외한 영화들의 유사도 및 인덱스를 추출
sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse=True) #
유사도가 높은 순서대로 정렬
sim_scores[0:10] # 상위 10개의 인덱스와 유사도를 추출
```

```
[(15282, 0.5262275451171008),
 (2979, 0.463276799830381),
 (10271, 0.2797390476075632),
 (8303, 0.20078538664316947),
 (1058, 0.18287334034120212),
 (11367, 0.15712074193481165),
 (1916, 0.15288512626542436),
 (3039, 0.1433450408051554),
 (483, 0.13765225108436677),
 (11573, 0.1337032693869044)]
```

* Toy story와 유사한 상위10개 제목

```
sim_scores = [(movie2id[i], score) for i, score in sim_scores[0:10]]
sim_scores
```

```
[('Toy Story 3', 0.5262275451171008),
 ('Toy Story 2', 0.463276799830381),
 ('The 40 Year Old Virgin', 0.2797390476075632),
 ('The Champ', 0.20078538664316947),
 ('Rebel Without a Cause', 0.18287334034120212),
 ('For Your Consideration', 0.15712074193481165),
 ('Condorman', 0.15288512626542436),
 ('Man on the Moon', 0.1433450408051554),
 ('Malice', 0.13765225108436677),
 ('Factory Girl', 0.1337032693869044)]
```

내용 기반 필터링

- 예시(Word2Vec) → 영화 제목을 한 단어로 보고 Word2vec을 사용

* Word2vec 알고리즘 호출

```
import gensim
```

* Movie 데이터 시간 순으로 정렬

```
movie = movie.sort_values(by='timestamp', ascending=True).reset_index(drop=True)
movie.head()
```

	userId	movieId	rating	timestamp
0	383	21	3.0	789652009
1	383	47	5.0	789652009
2	383	1079	3.0	789652009
3	409	21	5.0	828212412
4	409	25	4.0	828212412

* 영화 metadata 불러와 movieId로 컬럼명 맞춤

```
meta = meta.rename(columns={'id':'movieId'})
```

```
meta = pd.read_csv(path + 'movies_metadata.csv', low_memory=False)
```

```
movie['movieId'] = movie['movieId'].astype(str)
meta['movieId'] = meta['movieId'].astype(str)
```

* movieId로 join (original_title 활용)

```
movie = pd.merge(movie, meta[['movieId', 'original_title']], how='left', on='movieId')
```

```
movie = movie[movie['original_title'].notnull()].reset_index(drop=True)
```

내용 기반 필터링

예시(Word2Vec)

* userId로 groupby해서 original_tilte의 unique 항목 추출

```
agg = movie.groupby(['userId'])['original_title'].agg({'unique'})
agg.head()
```

userId	unique
1	[Jay and Silent Bob Strike Back, Vivement dima...
2	[Terminator 3: Rise of the Machines, The Conve...
3	[300, The Killing, Shortbus, Finding Neverland...
4	[David, The Wedding Planner, Casablanca, Sleep...
5	[Gleaming the Cube, Cool Hand Luke, Hidalgo, U...

* int형식을 String으로 변경

```
sentence = []
for user_sentence in agg['unique'].values:
    sentence.append(list(map(str, user_sentence)))
```

* Word2vec 학습 진행 (skip gram)

```
from gensim.models import Word2Vec
```

```
embedding_model = Word2Vec(sentence, size=20, window = 5,
                           min_count=1, workers=4, iter=200, sg=1)
```

내용 기반 필터링

- 예시(Word2Vec)

* Spider-Man2 와 유사한 영화목록 추출

```
embedding_model.wv.most_similar(positive=['Spider-Man 2'], topn=10)
```

```
[('Snow Cake', 0.8647751212120056),
 ('Sunrise: A Song of Two Humans', 0.7735385894775391),
 ('Face/Off', 0.764630913734436),
 ('Harry Potter and the Prisoner of Azkaban', 0.7302199602127075),
 ('Licence to Kill', 0.7103623747825623),
 ('The Godfather', 0.7101566791534424),
 ('薔薇の葬列', 0.7095184326171875),
 ('Domicile Conjugal', 0.7022347450256348),
 ('Rumor Has It...', 0.6998509168624878),
 ('Forrest Gump', 0.6897859573364258)]
```

시리즈 영화일 경우, 시청 순서를 반영해 잘 추천 해 줄 수 있겠다 생각했지만 제대로 추천해 주지 못함.

내용 기반 필터링

장점

- 다른 사람 데이터 필요X
- 독특한 취향 가진 사람들에게 좋음
- 새로운 작품 & 안유명한 아이템 추천가능
- 추천 이유 설명 가능

단점

- 이미지나, 음악 같은 경우 특징 자동화 어려움
- 새로운 사람인 경우 추천하기 어려움
- 취향에 지나치게 최적화됨

협업 기반 필터링(최근접 이웃)

- Main idea : 사용자 기반 협업 필터링(유사한 User 찾기)
- 예시

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6
사용자1	7	6	7	4	5	4
사용자2	6	7	?	4	3	4
사용자3	? 6.5	3	3	1	1	? 4
사용자4	1	2	2	3	3	4
사용자5	1	?	1	2	3	3

평균	Cosine(i, 3)	Pearson(i, 3)
5.5	0.956	0.894
4.8	0.981	0.939
2	1.0	1.0
2.5	0.789	-1.0
2	0.645	-0.817

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

$$\text{Cosine}(1,3) = \frac{6*3+7*3+4*1+5*1}{\sqrt{6^2+7^2+4^2+5^2}\cdot\sqrt{3^2+3^2+1^2+1^2}} = 0.956$$

$$sim(x,y) = \frac{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})(r_{ys}-\overline{r_y})}{\sqrt{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})^2}\sqrt{\sum_{s\in S_{xy}}(r_{ys}-\overline{r_y})^2}}$$

$$\text{Pearson}(1,3) = \frac{(6-5.5)*(3-2)+(7-5.5)*(3-2)+(4-5.5)*(1-2)+(5-5.5)*(1-2)}{\sqrt{1.5^2+1.5^2+(-1.5)^2+(-0.5)^2}\cdot\sqrt{1^2+1^2+(-1)^2+(-1)^2}} = 0.894$$

협업 기반 필터링(최근접 이웃)

- Main idea : 아이템 기반 협업 필터링(유사한 Item 찾기)
- 예시

	아이템1	아이템2	아이템3	아이템4	아이템5	아이템6	평균
사용자1	7	6	7	4	5	4	5.5
사용자2	6	7	?	4	3	4	4.8
사용자3	? 3	3	3	1	1	? 1	2
사용자4	1	2	2	3	3	4	2.5
사용자5	1	?	1	2	3	3	2
평균	3.75	4.5	3.25	2.8	3	3.75	
Cosine(1, j)	1	0.735	0.912	-0.848	-0.813	-0.990	
Cosine(6, j)	-0.990	-0.622	-0.912	0.829	0.730	1	

협업 기반 필터링(최근접 이웃)

- 예시(item 기반)

```
rating_data = pd.read_csv('./ratings.csv')
movie_data = pd.read_csv('./movies.csv')
```

```
rating_data.head()
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

→ User가 movie 평가

```
movie_data.head()
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

→ movie 데이터

* 합친 데이터

```
usr_movie_rating = pd.merge(rating_data,movie_data,on="movieId")
usr_movie_rating.head()
```

	userId	movieId	rating	title	genres
0	1	31	2.5	Dangerous Minds (1995)	Drama
1	7	31	3.0	Dangerous Minds (1995)	Drama
2	31	31	4.0	Dangerous Minds (1995)	Drama
3	32	31	4.0	Dangerous Minds (1995)	Drama
4	36	31	3.0	Dangerous Minds (1995)	Drama

협업 기반 필터링(최근접 이웃)

- 예시(item 기반)

* 영화 – 사용자 테이블(index : 영화, column : 사용자)

```
movie_usr_rating = usr_movie_rating.pivot_table('rating',index='title',columns='userId')
```

```
movie_usr_rating.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...	662	663	664	665	666	667	668	669	670	671
title																					
"Great Performances" Cats (1998)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
$9.99 (2008)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
'Hellboy': The Seeds of Creation (2004)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
'Neath the Arizona Skies (1934)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
'Round Midnight (1986)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

* Nan값 0으로 처리

```
movie_usr_rating.fillna(0,inplace=True)
movie_usr_rating.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...	662	663	664	665	666	667	668	669	670	671
title																					
"Great Performances" Cats (1998)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$9.99 (2008)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Hellboy': The Seeds of Creation (2004)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Neath the Arizona Skies (1934)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Round Midnight (1986)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

협업 기반 필터링(최근접 이웃)

- 예시(item 기반)

* 코사인 유사도 값 추출

```
from sklearn.metrics.pairwise import cosine_similarity

similarity_rate = cosine_similarity(movie_usr_rating, movie_usr_rating)
print(similarity_rate)
```

```
[[1.         0.         0.         ... 0.         0.         0.        ]
 [0.         1.         0.         ... 0.05821787 0.         0.        ]
 [0.         0.         1.         ... 0.         0.         0.        ]
 ...
 [0.         0.05821787 0.         ... 1.         0.         0.        ]
 [0.         0.         0.         ... 0.         1.         0.        ]
 [0.         0.         0.         ... 0.         0.         1.        ]]
```

* 유사도 값을 가진 데이터프레임 생성

```
similarity_rate_df = pd.DataFrame(
    data=similarity_rate,
    index=movie_usr_rating.index,
    columns=movie_usr_rating.index)
```

```
similarity_rate_df.head()
```

title	"Great Performances" Cats (1998)	$9.99 (2008)	'Hellboy': The Seeds of Creation (2004)	'Neath the Arizona Skies (1934)	'Round Midnight (1986)	'Salem's Lot (2004)	'Til There Was You (1997)	'burbs, The (1989)	'night Mother (1986)	(500) Days of Summer (2009)	...
title											
"Great Performances" Cats (1998)	1.000000	0.0	0.0	0.164399	0.020391	0.0	0.014046	0.000000	0.0	0.003166	...
$9.99 (2008)	0.000000	1.0	0.0	0.000000	0.000000	0.0	0.000000	0.079474	0.0	0.156330	...
'Hellboy': The Seeds of Creation (2004)	0.000000	0.0	1.0	0.000000	0.000000	1.0	0.000000	0.217357	0.0	0.000000	...
'Neath the Arizona Skies (1934)	0.164399	0.0	0.0	1.000000	0.124035	0.0	0.085436	0.000000	0.0	0.019259	...
'Round Midnight (1986)	0.020391	0.0	0.0	0.124035	1.000000	0.0	0.010597	0.143786	0.0	0.136163	...

협업 기반 필터링(최근접 이웃)

- 예시(item 기반)

  * 영화추천

```
# 가장 유사도가 높은 top 5
def recommand_moive(title):
    return similarity_rate_df[title].sort_values(ascending=False)[:6]
```

```
recommand_moive("Toy Story (1995)")
```

```
title
Toy Story (1995)                                1.000000
Toy Story 2 (1999)                              0.594710
Star Wars: Episode IV - A New Hope (1977)       0.576188
Forrest Gump (1994)                             0.564534
Independence Day (a.k.a. ID4) (1996)            0.562946
Groundhog Day (1993)                            0.548023
Name: Toy Story (1995), dtype: float64
```

협업 기반 필터링(최근접 이웃)

▶ 장점

- 평점만 있으면 추천가능

▶ 단점

- Cold start가 있다. (평가를 안하거나, 신작일 경우)
- 다수가 좋아하는 것 위주로 추천해준다.

★ 코멘트 수정부분

따라서, Hybrid 추천 시스템 사용! (cold start 해결책)

- 새로운 제품&신규유저 경우, 내용기반으로 추천
- 기존 제품 &기존 유저인 경우, 협업기반으로 추천

Cold start 문제

● Cold start란?

- 새로운 유저들에 대한 충분한 정보가 없어 제품을 추천하지 못하는 문제.

	아바타	설국열차	어바웃타임	NEW item
사람1	1	0.7	0.2	
사람2	0.5	0.5	0.8	
사람3	0.2	0.5	1	
New user				

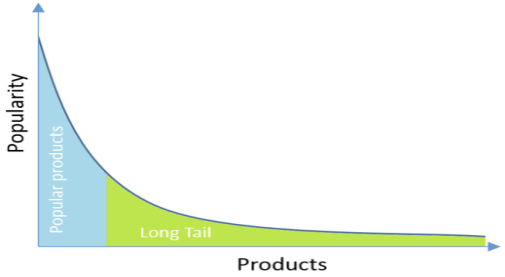
▶ Cold start가 발생하는 원인

① 새로운 커뮤니티

- 서비스를 처음 운영하기 시작해 사용유저가 한 명도 없는 경우

② 새로운 아이템

→ 좋은 제품이여도 인기없는 제품으로 취급되어 추천x



③ 새로운 유저

→ 초기 프로필 정보 요구

- 제품에 대한 유저의 인터렉션 정보가 없기 때문에

Cold start 해결 방법

- Profile Completion
  - 신규 user → FFM 질문지 사용
  - 신규 item → 비슷한 기존 제품 평점 활용
  - 인터페이스 에이전트(사용자들의 함축적 행동 패턴 관찰)
- Feature Mapping
  - 협업 기반 잠재 요인 기반 필터링(인터렉션 정보 결합하여 머신러닝 활용)
- Hybrid Feature Weighting
  - 하이브리드 내용 기반 필터링
    (사용자가 중요하게 생각하는 기준에 따라 가중치 부여)
- Differentiating Regularization weights
  - 제품, 사용자와 연관된 잠재 요인에 제약을 두어(정규화) 일반화

협업 기반 필터링(잠재 요인)

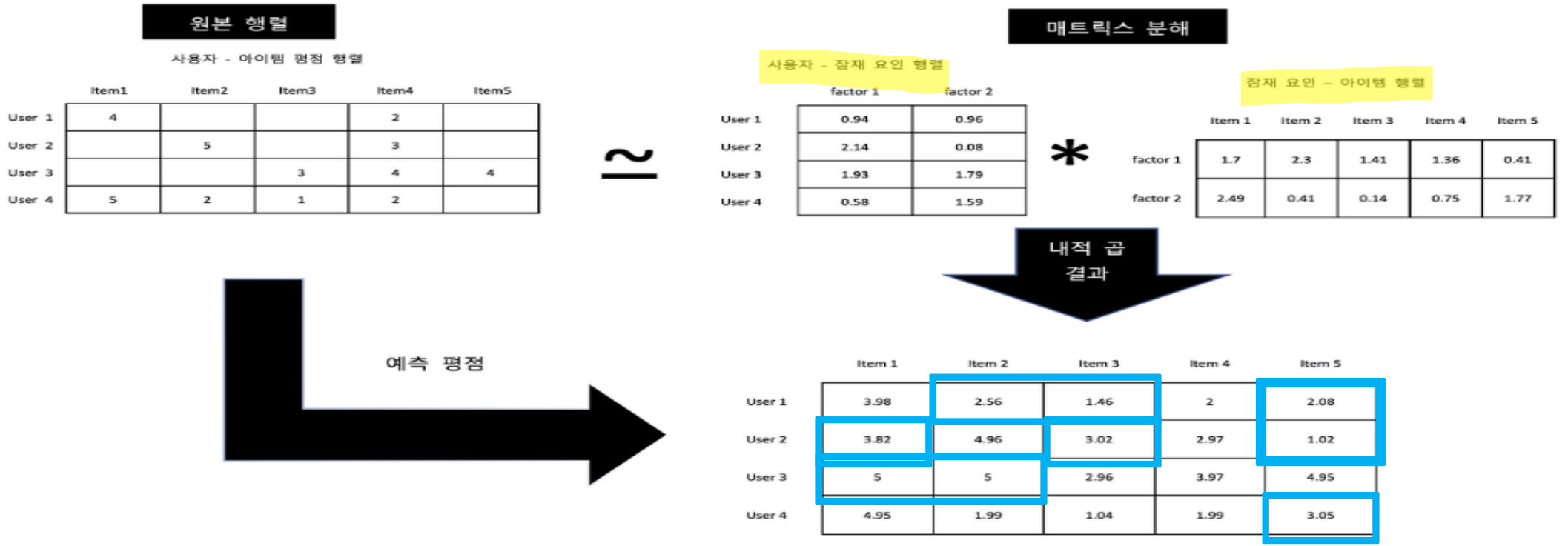
- Main idea : Matrix에 빈 칸을 채우기 위해 사용자와 아이템을 잘 표현하는 잠재 요인을 추출해 예측하는 방법

각 유사도를 계산해서 추천

행렬곱을 이용해 추천

협업 기반 필터링(잠재 요인)

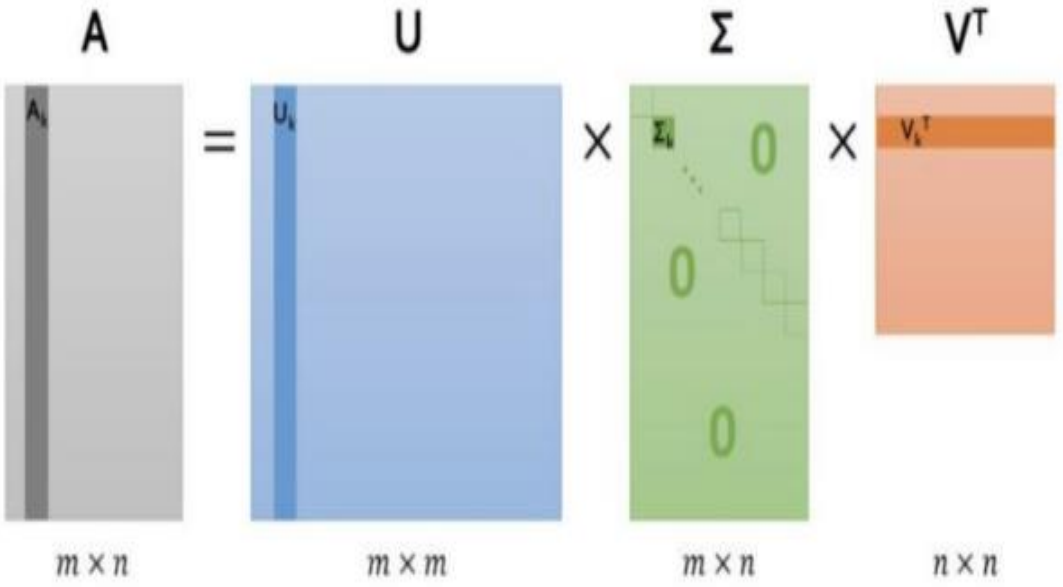
- Main idea : Matrix에 빈 칸을 채우기 위해 사용자와 아이템을 잘 표현하는 잠재 요인을 추출해 예측하는 방법



협업 기반 필터링(잠재 요인)

● SVD(특이값 분해)

$$A = U\Sigma V^T$$



▶ 예시

```
movie_user_pivot = usr_movie_pivot.values.T
movie_user_pivot.shape
```

```
(9064, 671)
```

* 특이값을 12로 설정

```
from sklearn.decomposition import TruncatedSVD

SVD = TruncatedSVD(n_components=12)
matrix=SVD.fit_transform(movie_user_pivot)
matrix.shape
```

```
(9064, 12)
```

```
matrix[0]
```

```
array([ 0.01227491,  0.00250758,  0.01554873, -0.0339781 , -0.01444946,
        0.00355278, -0.00214457,  0.04470513, -0.01596995, -0.02188731,
        0.00978786, -0.00940475])
```

협업 기반 필터링(잠재 요인)

● SVD(특이값 분해)

* 상관계수값 >= 0.9 인 영화 추천

```
movie_title = usr_movie_pivot.columns #user-item 원래행렬
movie_title_list = list(movie_title)
```

```
def recommand_movie(title):
    target=movie_title_list.index(title)
    corr_target = corr[target] #상관계수값
    result = list(movie_title[(corr_target>=0.9)])[:50] #상관계수값이 0.9이상인 영화만 추천

    return result
```

비슷한 영화 알려줌. 개인 맞춤형 x

```
recommand_movie("Iron Man 2 (2010)")
```

```
['10,000 BC (2008)',
 '2012 (2009)',
 '21 (2008)',
 '300: Rise of an Empire (2014)',
 'A-Team, The (2010)',
 'Abduction (2011)',
 'Adjustment Bureau, The (2011)',
 'Adventures of Tintin, The (2011)',
 'Alice in Wonderland (2010)',
 'Amazing Spider-Man, The (2012)',
 'Angels & Demons (2009)',
 'Ant-Man (2015)',
 'Argo (2012)',
 'Avatar (2009)',
 'Avengers, The (2012)',
 'Avengers: Age of Ultron (2015)',
 'Bad Teacher (2011)',
 'Batman: The Dark Knight Returns, Part 1 (2012)',
 'Batman: The Dark Knight Returns, Part 2 (2013)',
 'Battle: Los Angeles (2011)',
 'Bee Movie (2007)',
 'Beowulf (2007)',
 'Big Hero 6 (2014)',
 'Bolt (2008)',
 'Book of Eli, The (2010)',
 'Brave (2012)',
 'Captain America: Civil War (2016)',
 'Captain America: The First Avenger (2011)',
 'Captain America: The Winter Soldier (2014)',
 'Chronicle (2012)',
 'Click (2006)',
 'Cloverfield (2008)',
 'Cop Out (2010)',
 'Crank: High Voltage (2009)',
 'Dark Knight Rises, The (2012)',
 'Deadpool (2016)',
```

협업 기반 필터링(잠재 요인)

- ALS

- 두 행렬 중 하나를 고정시키고 다른 하나의 행렬을 순차적으로 반복하면서 최적화 하는 방법.

- Convex형태로 바뀌기 때문에 수렴된 행렬을 찾을 수 있다는 장점.

```
schema_ratings = StructType([
    StructField("user_id", IntegerType(), False),
    StructField("item_id", IntegerType(), False),
    StructField("rating", IntegerType(), False),
    StructField("timestamp", IntegerType(), False)])

schema_items = StructType([
    StructField("item_id", IntegerType(), False),
    StructField("movie", StringType(), False)])

training = spark.read.option("sep", "₩t").csv("MovieLens.training", header=False, schema=schema_ratings)
test = spark.read.option("sep", "₩t").csv("MovieLens.test", header=False, schema=schema_ratings)
items = spark.read.option("sep", "|").csv("MovieLens.item", header=False, schema=schema_items)
```

$$\forall u_i : J(u_i) = ||R_{i\cdot} - u_i \times V^T||_2 + \lambda \cdot ||u_i||_2$$

$$\forall v_j : J(v_j) = ||R_{i\cdot} - U \times v_j^T||_2 + \lambda \cdot ||v_j||_2$$

$$u_i = (V^T \times V + \lambda I)^{-1} \times V^T \times R_{i\cdot}$$

$$v_j = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{\cdot j}$$

협업 기반 필터링(잠재 요인)

- 예시(ALS)

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

als = ALS(maxIter=5, regParam=0.1, userCol="user_id", itemCol="item_id", ratingCol="rating",
          coldStartStrategy="drop", seed=20220509)
model = als.fit(train)

user_recs = model.recommendForAllUsers(5)
Urecs=user_recs.select("user_id","recommendations.item_id").collect()
sorted(Urecs,reverse=False)
```

```
/usr/local/lib/python3.7/dist-packages/pyspark/sql/context.py:127: FutureWarning: Deprecated
  FutureWarning
[Row(user_id=1, item_id=[1449, 867, 1643, 408, 1642]),
 Row(user_id=2, item_id=[1643, 1642, 1449, 318, 863]),
 Row(user_id=3, item_id=[1368, 854, 1174, 346, 1597]),
 Row(user_id=4, item_id=[1664, 1368, 1062, 793, 1260]),
 Row(user_id=5, item_id=[1240, 390, 1268, 919, 169]),
 Row(user_id=6, item_id=[1643, 1449, 1512, 408, 1368]),
 Row(user_id=7, item_id=[1643, 1449, 867, 1467, 1500]),
 Row(user_id=8, item_id=[954, 1347, 169, 867, 1344]),
 Row(user_id=9, item_id=[1643, 867, 313, 64, 318]),
 Row(user_id=10, item_id=[1643, 1449, 1642, 1368, 318]),
 Row(user_id=11, item_id=[1642, 1131, 867, 318, 1368]),
 Row(user_id=12, item_id=[1233, 1344, 64, 318, 1449]),
 Row(user_id=13, item_id=[867, 851, 1167, 408, 1628]),
 Row(user_id=14, item_id=[320, 1643, 1449, 185, 853]),
 Row(user_id=15, item_id=[1278, 909, 1503, 1167, 793]),
 Row(user_id=16, item_id=[1643, 613, 1269, 64, 1467]),
 Row(user_id=17, item_id=[1449, 1467, 1142, 1388, 408]),
 Row(user_id=18, item_id=[1642, 1449, 1643, 1233, 1599]),
```

```
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

```
Root-mean-square error = 0.9443591036410887
```

③ 맞게 채웠는지 어떻게 평가 할 것인가?

- MAE(Mean absolute Error)

$$\frac{1}{n}\sum_{i=1}^{n}\left|y_i-\hat{y}_i\right|$$

예측1) MAE

$$\sqrt{\frac{1+1+1+1+1}{5}}=1$$

=

예측2) MAE

$$\sqrt{\frac{0+0+5+0+0}{5}}=1$$

- RMSE(예측값과 실제값의 차의 루트 값)

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}$$

→ 값이 작을수록 좋음, 에러 증폭시킴

실제	0	0	5	0	0
예측1	1	1	4	1	1
예측2	0	0	0	0	0

예측1) RMSE

$$\sqrt{\frac{1+1+1+1+1}{5}}=1$$

<

예측2) RMSE

$$\sqrt{\frac{0+0+25+0+0}{5}}=\sqrt{5}$$

③ 맞게 채웠는지 어떻게 평가 할 것인가?

- MAP(Mean Average Precision) : AP를 평균낸 값

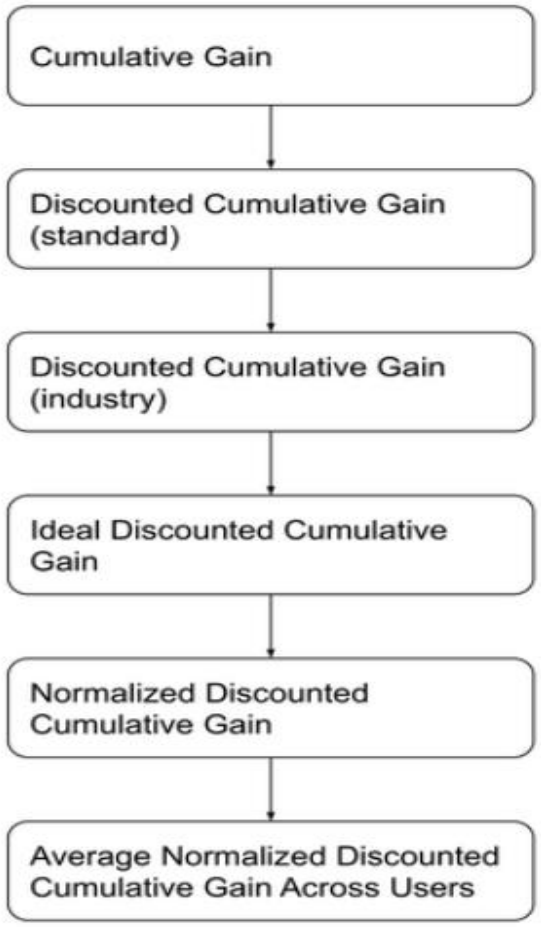
Recommendations	Precision @k's	AP@3
[0, 0, 1]	[0, 0, 1/3]	(1/3)*(1/3) = 0.11
[0, 1, 1]	[0, 1/2 , 2/3]	(1/3) [1/2 + 2/3] = 0.38
[1, 1, 0]	[1/1, 2/2, 2/3]	(1/3) [1/1 + 2/2 + 2/3] = 0.89
[1, 1, 1]	[1/1, 2/2, 3/3]	(1/3) [1 + 2/2 + 3/3] = 1

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ FalsePositive}$$

→ 순서에 따라 값이 다름.
→ 값이 클수록 좋음

③ 맞게 채웠는지 어떻게 평가 할 것인가?

- NDCG(Normalized Discounted Cumulative Gain)



$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

예시)

$$Recommendations\ Order = [2, 3, 3, 1, 2]$$
$$Ideal\ Order = [3, 3, 2, 2, 1]$$

$$CG_A = 2 + 3 + 3 + 1 + 2 = 11$$
$$CG_B = 3 + 3 + 2 + 2 + 1 = 11$$

순서 반영 x

$$DCG = \frac{2}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{3}{\log_2(3+1)} + \frac{1}{\log_2(4+1)} + \frac{2}{\log_2(5+1)} \approx 6.64$$

순서 반영 o

$$iDCG = \frac{3}{\log_2(1+1)} + \frac{3}{\log_2(2+1)} + \frac{2}{\log_2(3+1)} + \frac{2}{\log_2(4+1)} + \frac{1}{\log_2(5+1)} \approx 7.14$$

$$NDCG = \frac{DCG}{iDCG} = \frac{6.64}{7.14} \approx 0.93$$

정규화

→ 값이 클수록 좋음

넷플릭스 알고리즘

https://www.youtube.com/watch?v=me--WQKQQAo&t=4s

- 나이브 베이즈 이론

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

조건부 확률 P(A|B) 는 **사건 B 가 발생한 경우 A 의 확률**

감사합니다!