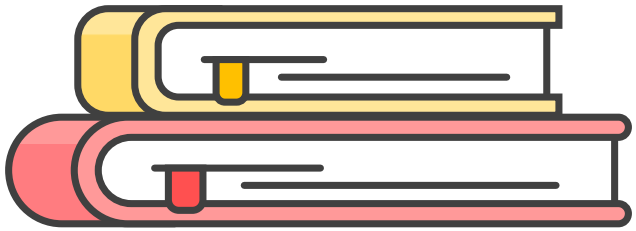
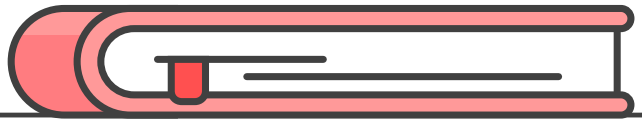


Melon Playlist Continuation

2016010736 최우철

2021210088 허지혜

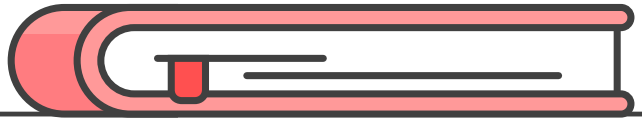
2019010718 강수연



대회 설명

- 플레이리스트에 있는 곡들과 유사한 노래들을 추천해주는 모델을 만드는 것이 목표!

- 플레이리스트에 수록된 곡과 태그의 절반 또는 전부가 숨겨져 있을 때, 주어지지 않은 곡들과 태그를 예측한다



추천 시스템

추천시스템이란?
: 사용자에게 상품을 제안하는 SW 도구이자 기술

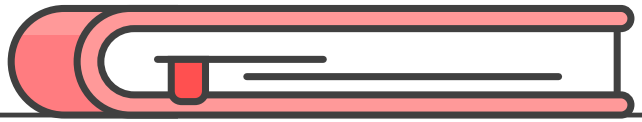
ex) 상품, 음악, 온라인 뉴스 추천 등
다양한 의사결정과 연관

추천 시스템

컨텐츠기반
추천시스템

VS

협업필터링
기반
추천시스템



컨텐츠 기반 추천시스템

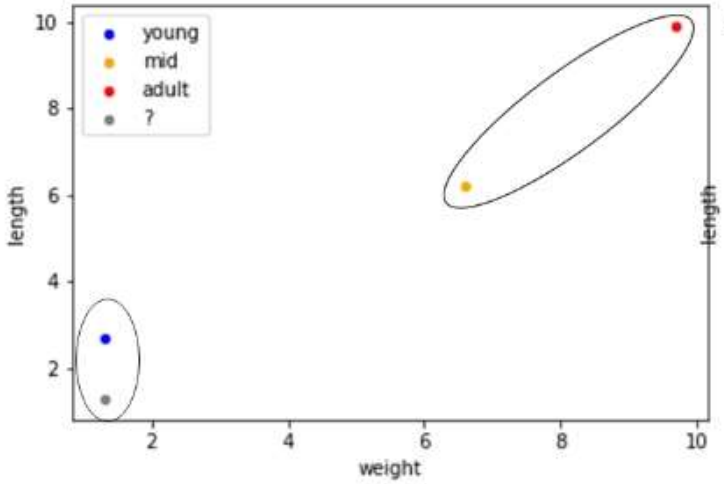
사용자가 이전에 구매한 상품 중에서 좋아하는 상품들과
유사한 상품들을 추천하는 방법

컨텐츠(Items) → 벡터(Vector)

컨텐츠 기반 추천시스템

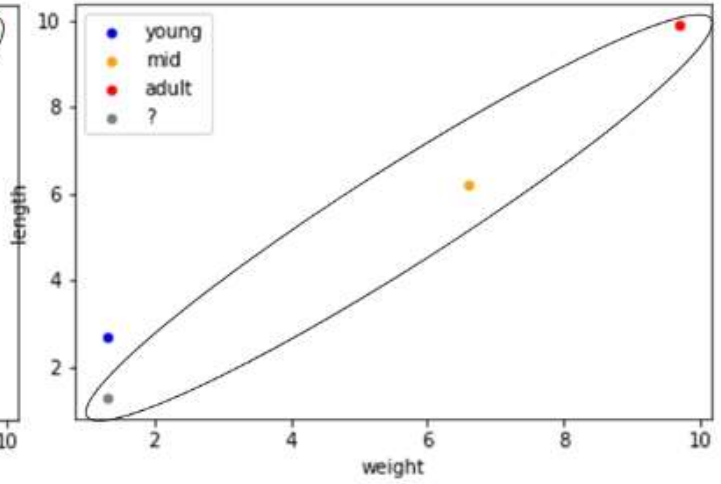
유사도 함수

유클리디안 유사도



거리 중심

코사인 유사도



방향성 중심

평가 함수(NDCG)

Cumulative Gain(CG)

$$Cumulative\ Gain(CG) = \sum_{i=1}^{n} relevance_i$$

$$Set\ A = [2,\ 3,\ 3,\ 1,\ 2]$$

$$Set\ B = [3,\ 3,\ 2,\ 2,\ 1]$$

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

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

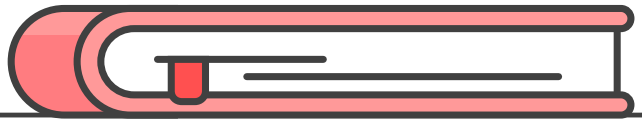
Discounted Cumulative Gain(DCG)

$$DCG = \sum_{i=1}^{n} \frac{relevance_i}{log_2(i+1)} \qquad DCG = \sum_{i=1}^{n} \frac{2^{relevance_i} - 1}{log_2(i+1)}$$

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

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

$$DCG_A < DCG_B$$



평가 함수(NDCG)

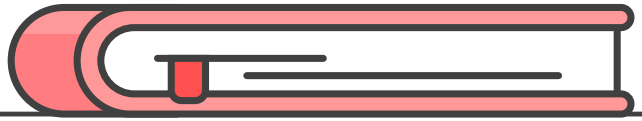
Normalized Discounted Cumulative Gain(NDCG)

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

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

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

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



컨텐츠 기반 추천시스템(TF-IDF)

특정 문서에서만 자주 등장하는 단어를 찾아서 문서 내 단어의 가중치를 계산하는 방법

- TF : 특정 문서에서 특정 단어의 등장 횟수
- DF : 특정 단어가 등장한 문서의 수
- IDF : DF에 반비례하는 수
- TF-IDF : TF와 IDF를 곱해준 값

컨텐츠 기반 추천시스템(TF-IDF)

TF

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

DF

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
총합	1	1	1	2	3	1	2	1	1

컨텐츠 기반 추천시스템(TF-IDF)

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(3+1)) = 0
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

TF-IDF

	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.2876	0	0.6931	0.2876	0	0
문서2	0	0	0	0.2876	0	0	0.2876	0	0
문서3	0	0.6931	0.6931	0	0	0	0	0	0
문서4	0.6931	0	0	0	0	0	0	0.6931	0.6931

코사인 유사도 사용

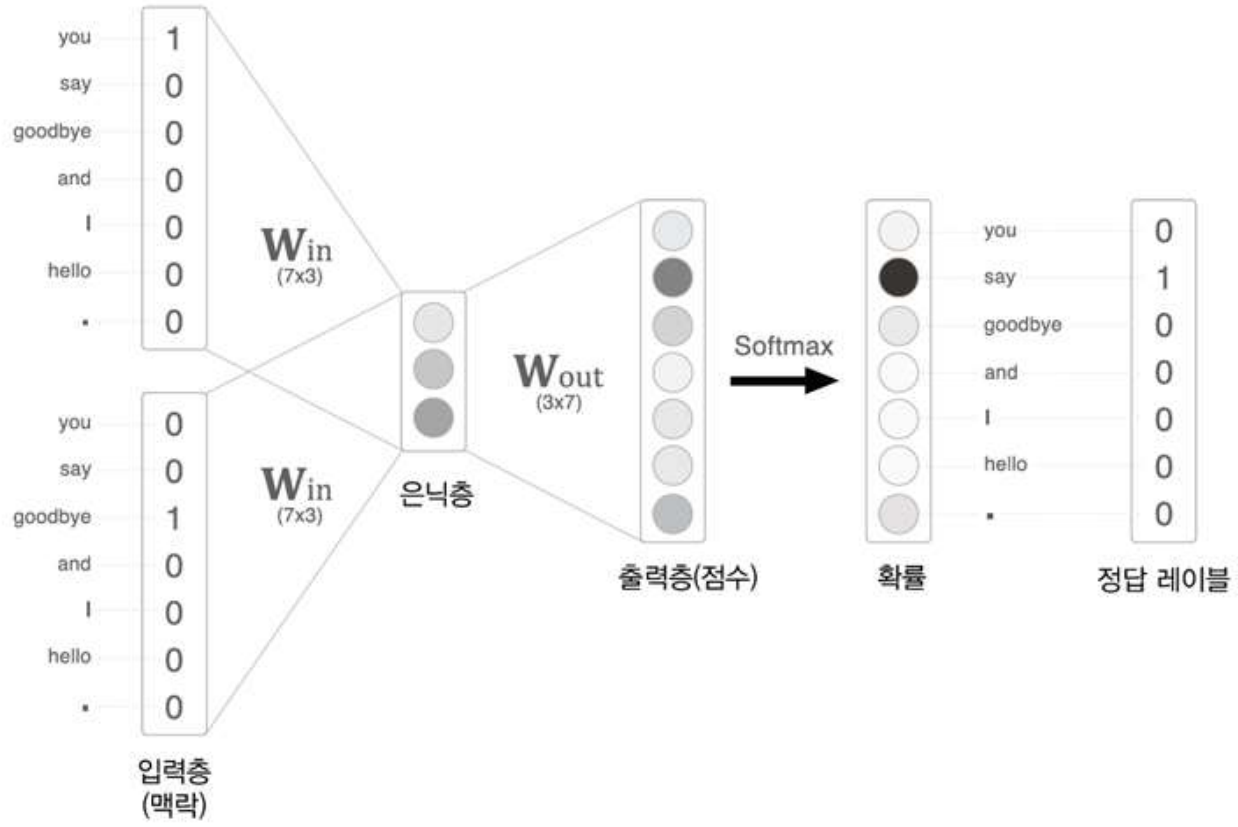
	문서1	문서2	문서3	문서4
문서1	1	0.5061	0	0
문서2	0.5061	1	0	0
문서3	0	0	1	0
문서4	0	0	0	1

컨텐츠 기반 추천시스템(Word2Vec)

CBOW : 주변 단어(맥락)를 통해서 중심 단어를 채우는 방법
Skip-gram : 중심 단어를 통해서 주변 단어를 채우는 방법

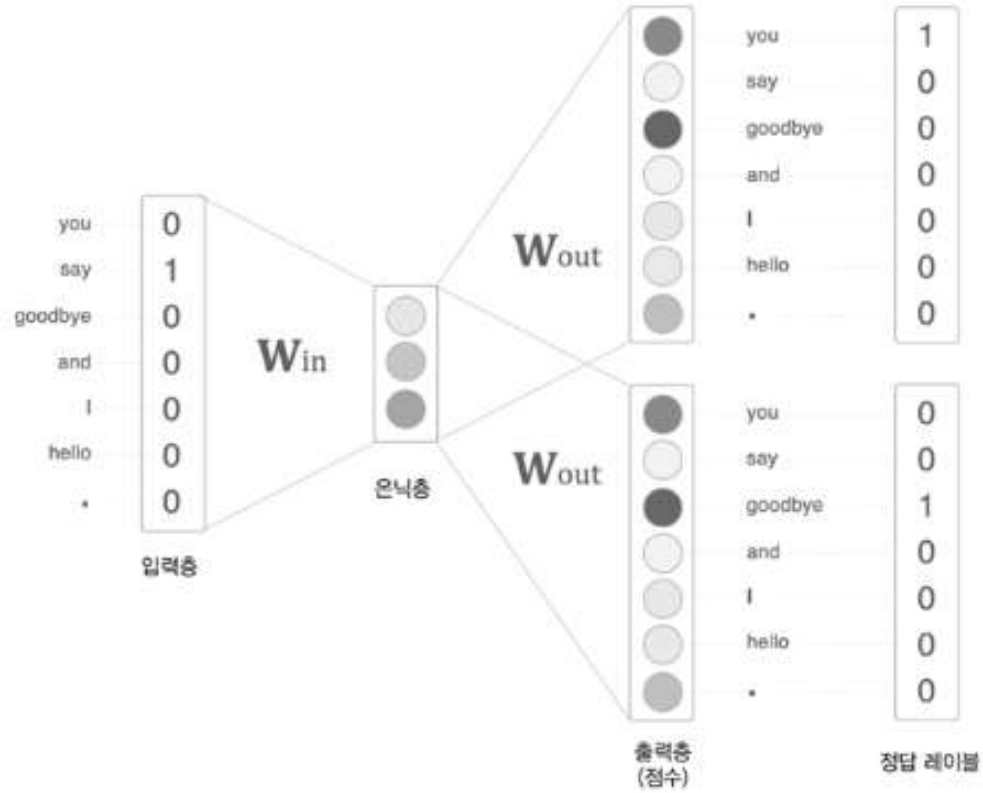
컨텐츠 기반 추천시스템(Word2Vec)

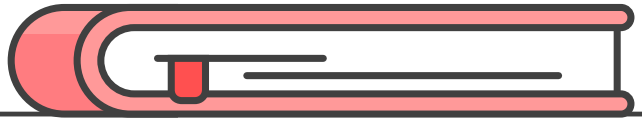
CBOW



컨텐츠 기반 추천시스템(Word2Vec)

Skip-gram





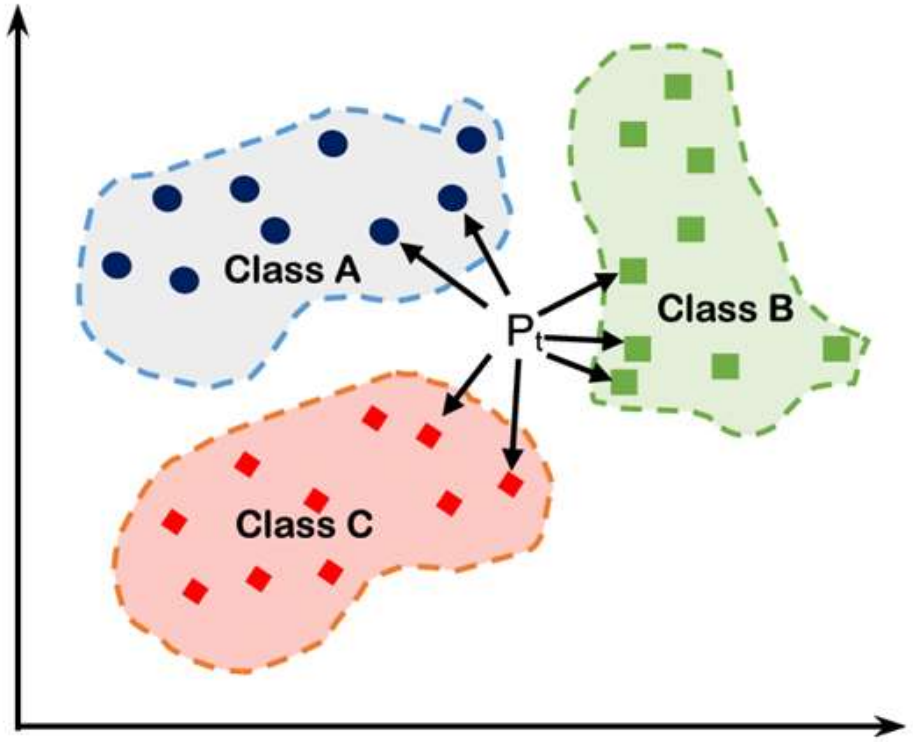
협업필터링 기반 추천시스템

사용자의 구매 패턴이나 평점을 가지고 다른 사람들의 구매 패턴, 평점을 통해서 추천을 하는 방법

- User-based collaborative filtering : 사용자의 구매 패턴(평점)과 유사한 사용자를 찾아서 추천 리스트 생성

- Item-based collaborative filtering : 특정 사용자가 준 점수간의 유사한 상품을 찾아서 추천 리스트 생성

협업필터링 기반 추천시스템(KNN)



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

협업필터링 기반 추천시스템(KNN)

User-based collaborative filtering

	평균	Cosine(i, 3)	Pearson(i, 3)
사용자1	5.5	0.956	0.894
사용자2	4.8	0.981	0.939
사용자3	2	1.0	1.0
사용자4	2.5	0.789	-1.0
사용자5	2	0.645	-0.817

	아이템1	아이템6	평균	Pearson(i, 3)
사용자1	7	4	5.5	0.894
사용자2	6	4	4.8	0.939
사용자3	3.35	0.86	2	1.0
사용자4	1	4	2.5	-1.0
사용자5	1	3	2	-0.817

$$\hat{r}_{31} = \frac{7 * 0.894 + 6 * 0.939}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{4 * 0.894 + 4 * 0.939}{0.894 + 0.939} = 4$$

사용자1의 아이템1의 평점 - 사용자1의 평균 평점

$$\hat{r}_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} \approx 3.35$$

$$\hat{r}_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} \approx 0.86$$

사용자3의 평균 평점

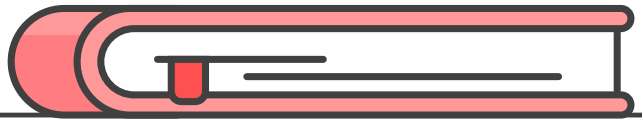
협업필터링 기반 추천시스템(KNN)

Item-based collaborative filtering

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

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$

$$\hat{r}_{36} = \frac{1 * 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$



협업필터링 기반 추천시스템(SGD)

두 개의 행렬을 동시에 최적화하는 방법

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

subject to:

No constraints on U and V

$S = \{(i,j) : r_{ij} \text{ is observed}\}$

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

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) \quad \forall i \in \{1\ldots m\}, q \in \{1\ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) \quad \forall j \in \{1\ldots n\}, q \in \{1\ldots k\}$$

Matrix Factorization

↓

편미분

↓

Regularization

협업필터링 기반 추천시스템(SGD)

1. User Latent와 Item Latent의 임의로 초기화

User Latent (U)

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

np.dot

Item Latent (V)의 Transpose

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

=

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

협업필터링 기반 추천시스템(SGD)

0.5756	1.4534	
-0.199	-1.218	
2.7297	0.48	
-0.039	-2.506	

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

2. Gradient Descent 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.2819	0.6663	1.4981
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

0.446	1.5574	
-0.199	-1.218	
2.7297	0.48	
-0.039	-2.506	

0.3668	-1.0401	1.4593
-0.3392	1.0663	0.4528

-0.3647	1.1967	1.3560
0.3403	-0.8728	-0.8421
0.8384	-2.5933	4.2008
0.8354	-2.2043	-1.1928

협업필터링 기반 추천시스템(SGD)

0.4928	1.5711
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

0.3668	-1.0401	1.4729
-0.3392	1.0663	0.5027

3. 모든 평점에 대해서 반복 (epoch : 1)
- ? 를 제외한 모든 평점에 대해서 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.3522	1.1628	1.5157
0.3403	-1.0924	-0.9057
0.8384	-2.3273	4.2620
0.8354	-2.6308	-1.3184

0.4927	1.5711
-0.015	-1.101
2.3345	0.5363
0.2363	-2.464

0.7858	-0.4137	1.0724
-0.6250	1.0040	-0.3381

3. 모든 평점에 대해서 반복 (epoch : 1)
- ? 를 제외한 모든 평점에 대해서 진행

?	3	2
5	1	2
4	2	1
2	?	4

-0.5947	1.3736	-0.0028
0.6763	-1.0994	0.3561
1.4991	-0.4721	2.3222
1.7260	-2.5722	1.0869

협업필터링 기반 추천시스템(SGD)

1.6462	1.0993	
1.6740	-1.072	
2.0550	0.6342	
0.3135	-2.663	

2.1118	0.8951	0.9768
-0.6646	0.8560	-1.1103

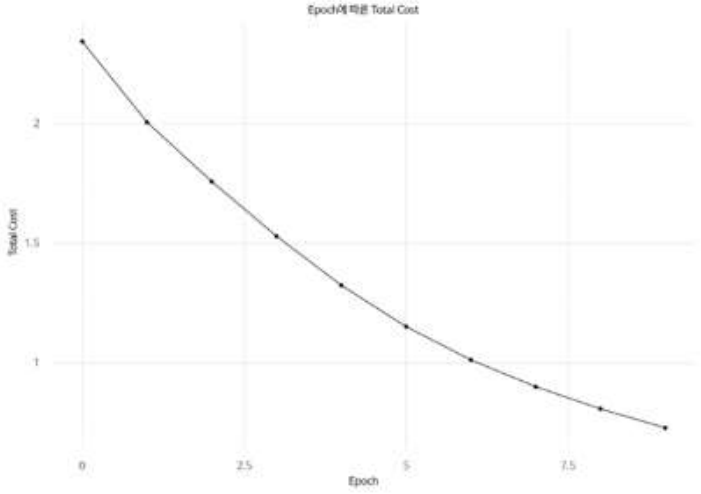
4. 2~3의 과정을 10번 반복 (epoch : 10)

?	3	2
5	1	2
4	2	1
2	?	4

평점이 높은 1번 상품은 1번 유저에게 추천 o

2.7458	2.4147	0.3873
4.2476	0.5806	2.8256
3.9181	2.3825	1.3030
2.4323	-1.9994	3.2637

평점이 낮은 2번 상품은 4번 유저에게 추천 x



협업필터링 기반 추천시스템(ALS)

두 행렬 중 하나를 고정, 나머지 행렬을 반복하면서 최적화

1. 초기 아이템, 사용자 행렬을 초기화

User Latent

0.5756	1.4534
-0.199	-1.218
2.7297	0.48
-0.039	-2.506

Item Latent 의 Transpose

0.3668	-1.1078	1.4593
-0.3392	0.8972	0.4528

2. 아이템 행렬을 고정하고 사용자 행렬을 최적화 $u_i = (V^T \times V + \lambda I)^{-1} \times V^T \times R_{i\cdot}$

0.3151	3.2962
1.1118	0.5423
0.3225	0.9144
2.1187	1.8521

모든 Row에 대해서 진행

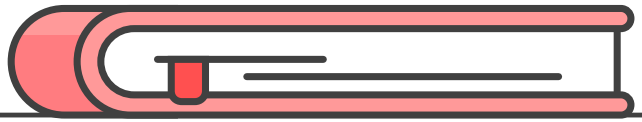
$u_1 = (V^T \times V + \lambda I)^{-1} \times V^T \times R_{1\cdot} = [0.3151, 3.2962]$

3. 사용자 행렬을 고정하고 아이템 행렬을 최적화 $v_j = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{\cdot j}$

1.9557	-0.090
-0.525	0.9939
1.5017	0.4663

모든 Row에 대해서 진행

$v_1 = (U^T \times U + \lambda I)^{-1} \times U^T \times R_{\cdot 1} = [1.9557, -0.0900]$



Matrix Factorization

평가를 내리지 않은 user-item의 빈 공간을 Model-Based-Learning으로 채워 넣는 것

- 유저/아이템간 유사도를 이용하는 Memory-based 와는 달리, 행렬 인수분해라는 수학적 방법으로 접근.

- 행렬이 두 개의 하위행렬로 분해 가능하며, 다시 곱해져서 원래 행렬과 동일한 크기의 단일 행렬이 될 수 있는 성질을 사용했다.

Matrix Factorization

- 일반적으로 작은 수 k(≈ 10)를 고정 후, 각 유저의 u를 k차원 벡터 x_u로 요약 후, 아이템 i를 k차원 벡터 y_i로 요약

- $x_1, \dots, x_n \in R^k$가 유저들에 대한 factor, $y_1, \dots, y_m \in R^k$가 아이템에 대한 factor들이 되게 한다.

- 그 후 Matrix X,Y를 다음과 같이 정의한다.

$$X = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{bmatrix} \quad Y = \begin{bmatrix} | & & | \\ y_1 & \cdots & y_m \\ | & & | \end{bmatrix}$$

- Claim : $R \approx X^T Y$를 추정

→ 목적함수 최소화 및 최적의 X,Y를 찾는 최적화 문제로 변경

$$\min_{X,Y} \sum_{r_{ui} \text{ observed}} (r_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

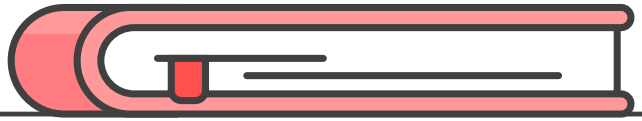
Matrix Factorization

- 목적함수는 볼록하지 않다. ($\because x_u^T y_i$ term) 경사 하강법 사용시 속도도 느리고, iteration cost도 많이 든다.

- Variable X set 고정 후, 상수 취급 시, 목적 함수는 Y의 convex function이 되고, 반대 경우도 가능하다.

- 결론적으로 Y를 고정후 X 최적화, X 고정후 Y 최적화 한다.

- 이를 수렴할 때 까지 반복한다. 이 과정을 ALS라 한다.



Alternating Least squares

- 계산비용 분석

x_u를 업데이트시 $O(n_u k^2 + k^3)$ 의 비용이 듦. n_u : 유저 u가 rating한 item 수

y_i 업데이트시 $O(n_i k^2 + k^3)$ 이 듦. n_i: 아이템 i에 rating한 user 수

- X,Y를 계산 하고나면, x_u, y_i 를 다른 학습 알고리즘 feature로 사용해 이 feature들과 다른 feature들을 합쳐 다른 예측 알고리즘에 활용한다.

Alternating Least squares

M = playlist, u=songs

One of decomposion of M

$M = UV^T$ $U = m \times d$ $V = n \times d$, d는 arbitrary한 dimention

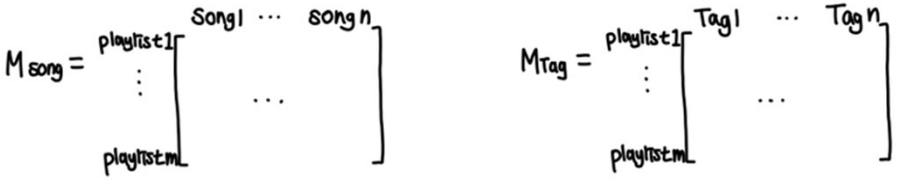
$\min |M - UV^T|^2$: 최소 제곱 오차 $\begin{bmatrix} u_1^T \\ \vdots \\ u_n^T \end{bmatrix} v_j = \begin{bmatrix} m_{1j} \\ \vdots \\ m_{nj} \end{bmatrix}$ v_j : 최소제곱해로 찾는다.

반대로, $u_i^T [v_1 \cdots v_u] = [y_{i1} \cdots y_{in}]$

만약 $u_i^T v_j$ 가 충분히 클 경우, playlist i 안의 song j를 예측 가능.

Collective Matrix factorization

현재 우리는 songs와 tags를 동시 예측을 해야함.
같은 제한사항에서 제시된 두 행렬을 ALS를 사용해 해결.



$= U_{song} V_{song}^T$ $= U_{Tag} V_{Tag}^T$

$\min (\| U_{song} V_{song}^T - M_{song} \|^2 + \| U_{Tag} V_{Tag}^T - M_{Tag} \|^2)$, min 안의 식을 ①이라 두자.

U_{song}, U_{tag}가 유사하다고 다음과 같이 제한하자. $\| U_{song_i} - U_{tag_i} \|^2 \leq \lambda$

정규화 form을 다음과 같이 둔다. $\min (\sum_{x \in \{song, tag\}} \| U_x V_x^T - M_x \|^2 + \| U_{song} - U_{Tag} \|^2)$ or $\lambda = 0$ $U_{song} = U_{Tag} = U$

위 form을 통해 ①식 = $(U \begin{bmatrix} V_{song} \\ V_{tag} \end{bmatrix}^T - [M_{song} M_{tag}])^2$ 을 얻는다.

이를 통해 어떤 코드가 오던 ALS를 통해 문제를 해결 가능하다.

EDA 코드

train.json

1) 데이터 불러오기

```
In [2]: import pandas as pd
        train = pd.read_json('train.json', typ = 'frame')

In [3]: train.head()
```

Out[3]:

	tags	id	plylst_title	songs	like_cnt	updt_date
0	[락]	61281	여행같은 음악	[525514, 129701, 383374, 562083, 297861, 13954...	71	2013-12-19 18:36:19.000
1	[추억, 회상]	10532	요즘 너 말야	[432406, 675945, 497066, 120377, 389529, 24427...	1	2014-12-02 16:19:42.000
2	[까페, 잔잔한]	76951	편하게, 잔잔하게 들을 수 있는 곡.-	[83116, 276692, 166267, 186301, 354465, 256598...	17	2017-08-28 07:09:34.000
3	[연말, 눈오는날, 캐럴, 분위기, 따듯한, 크리스마스캐럴, 겨울노래, 크리스마스,...	147456	크리스마스 분위기에 흠뻑 취하고 싶을때	[394031, 195524, 540149, 287984, 440773, 10033...	33	2019-12-05 15:15:18.000
4	[댄스]	27616	추억의 노래 ㅋ	[159327, 553610, 5130, 645103, 294435, 100657,...	9	2011-10-25 13:54:56.000

```
In [4]: train.shape
Out[4]: (115071, 6)

In [5]: train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115071 entries, 0 to 115070
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   tags          115071 non-null  object
 1   id            115071 non-null  int64 
 2   plylst_title  115071 non-null  object
 3   songs         115071 non-null  object
 4   like_cnt      115071 non-null  int64 
 5   updt_date     115071 non-null  object
dtypes: int64(2), object(4)
memory usage: 5.3+ MB
```

EDA 코드

train.json

2) id+song, id+tag만 추출해 DataFrame 생성

In [7]:
```
# 수록곡들이 list 형태로 되어있으니 각각의 행으로 만들어주는 작업

import numpy as np
# 플레이리스트 아이디(id)와 수록곡(songs) 추출
plylst_song_map = train[['id', 'songs']]

# unnest songs
plylst_song_map_unnest = np.dstack(
    (
        np.repeat(plylst_song_map.id.values, list(map(len, plylst_song_map.songs))),
        np.concatenate(plylst_song_map.songs.values)
    )
)

# unnested 데이터프레임 생성 : plylst_song_map
plylst_song_map = pd.DataFrame(data = plylst_song_map_unnest[0], columns = plylst_song_map.columns)
plylst_song_map['id'] = plylst_song_map['id'].astype(str)
plylst_song_map['songs'] = plylst_song_map['songs'].astype(str)

# unnest 객체 제거
del plylst_song_map_unnest
```

In [9]:
```
# 플레이리스트 아이디(id)와 매핑된 태그(tags) 추출
plylst_tag_map = train[['id', 'tags']]

# unnest tags
plylst_tag_map_unnest = np.dstack(
    (
        np.repeat(plylst_tag_map.id.values, list(map(len, plylst_tag_map.tags))),
        np.concatenate(plylst_tag_map.tags.values)
    )
)

# unnested 데이터프레임 생성 : plylst_tag_map
plylst_tag_map = pd.DataFrame(data = plylst_tag_map_unnest[0], columns = plylst_tag_map.columns)
plylst_tag_map['id'] = plylst_tag_map['id'].astype(str)

# unnest 객체 제거
del plylst_tag_map_unnest
```

EDA 코드

train.json

3) 생성한 DataFrame 확인

```
In [10]: plylst_tag_map
```

Out[10]:

	id	tags
0	61281	락
1	10532	추억
2	10532	회상
3	76951	까페
4	76951	잔잔한
...	...	...
476326	131982	퇴근길
476327	100389	노래추천
476328	100389	팝송추천
476329	100389	팝송
476330	100389	팝송모음

476331 rows × 2 columns

```
In [18]: train_song_cnt = plylst_song_map.songs.nunique()
         train_tag_cnt = plylst_tag_map.tags.nunique()

         print("곡 수 : %s" %train_song_cnt)
         print("태그 수 : %s" %train_tag_cnt)
```

```
곡 수 : 615142
태그 수 : 29160
```

EDA 코드

train.json

4) 워드 크라우드 그리기

In [27]:
```
from pandas.plotting import register_matplotlib_converters
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
font_path = 'C:/WINDOWS/FONTS/BATANG.TTC'
font_name = fm.FontProperties(fname=font_path, size=10).get_name()
plt.rc('font', family=font_name, size=12)
plt.rcParams["figure.figsize"] = (20, 10)
register_matplotlib_converters()

mpl.font_manager._rebuild()
mpl.pyplot.rc('font', family='NanumGothic')
from wordcloud import WordCloud
# 태그 별 매핑 빈도 수 저장
tag_cnt = plylst_tag_map.groupby('tags').tags.count().reset_index(name = 'mapping_cnt')
tag_cnt['tags'] = tag_cnt['tags'].astype(str)
tag_cnt['mapping_cnt'] = tag_cnt['mapping_cnt'].astype(int)

# 빈도 수가 1000회 이상인 태그만 저장
tag_cnt = tag_cnt[tag_cnt['mapping_cnt'] >= 1000]
word_count = list(zip(tag_cnt['tags'], tag_cnt['mapping_cnt']))

# plotting
wc = WordCloud(font_path = font_path, background_color = 'white', max_words = 100, width = 450, height = 450)
wc.generate_from_frequencies(dict(word_count)).to_image()
```

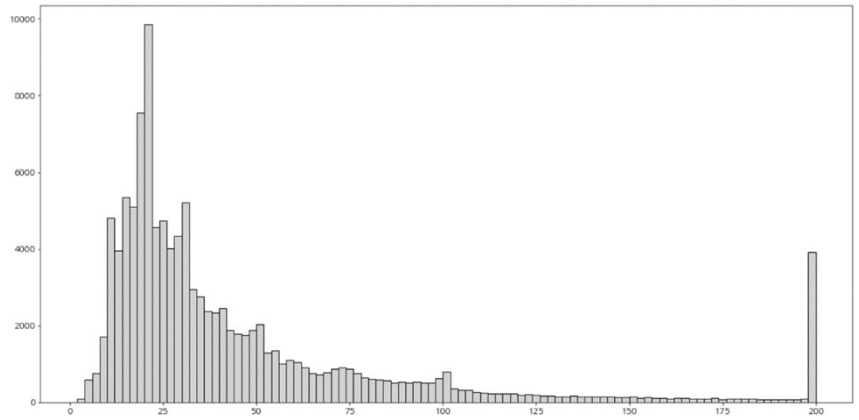
Out[27]:

EDA 코드

train.json

5) 수록곡 분포 시각화

In [28]:
```
# 1. id 별로 리스트 뜯어보기
plylst_song_cnt = pd.DataFrame(plylst_song_map.groupby('id').songs.nunique())

# 2-1. grid setting
grid_list = [i*2 for i in range(1, 101)]

# 2-2. plotting
plt.hist(plylst_song_cnt['songs'], grid_list, color = "lightgrey", edgecolor = "black")
plt.show()
```



In [32]:
```
round(plylst_song_cnt.describe(),2)
```

Out[32]:

	songs
count	115071.00
mean	45.94
std	43.95
min	1.00
25%	19.00
50%	30.00
75%	54.00
max	200.00

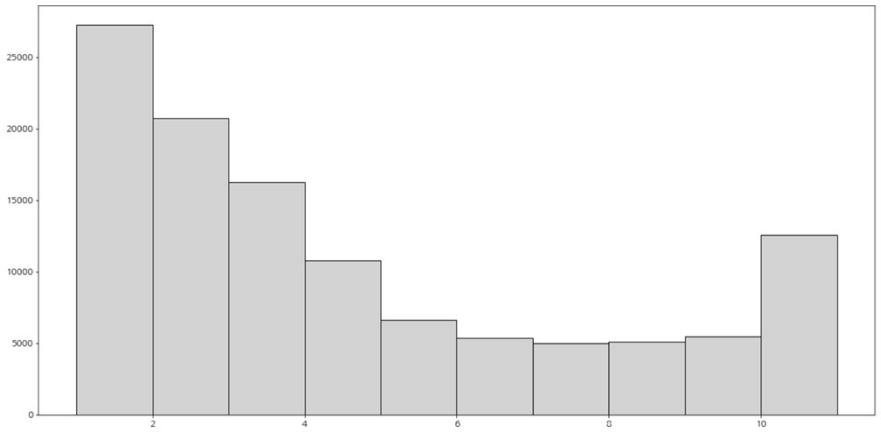
플레이리스트 별 수록된 곡 수의 분포를 아이디 별로 살펴보니 평균 약 46개의 곡이 리스트 안에 수록되어 있으며 최대 수록곡은 200곡이 있다.

EDA 코드

train.json

6) 태그 분포 시각화

In [33]:
```
# 1. 플레이리스트 별 매핑 태그 수 count 테이블 생성 : plylst_tag_cnt
plylst_tag_cnt = pd.DataFrame(plylst_tag_map.groupby('id').tags.nunique())

# 2. plotting
plt.hist(plylst_tag_cnt['tags'], range(1, 12), color = "lightgrey", edgecolor = "black")
plt.show()
```



In [34]:
```
round(plylst_tag_cnt.describe(),2)
```

Out[34]:

	tags
count	115071.00
mean	4.14
std	3.07
min	1.00
25%	2.00
50%	3.00
75%	6.00
max	11.00

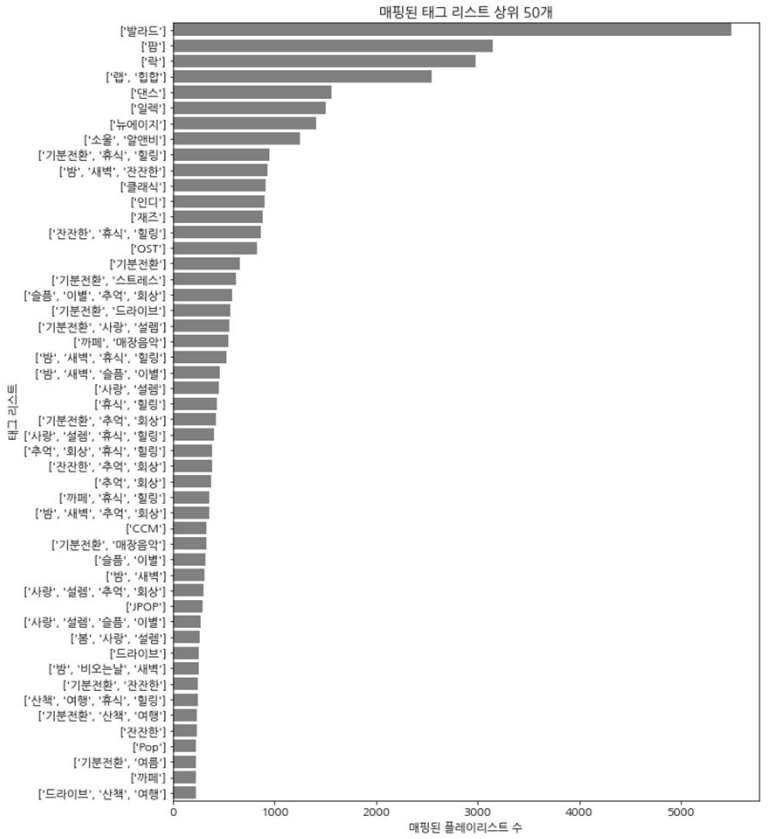
플레이리스트 당 태그는 평균 4개 이며 가장 많은 태그 수는 11개이다.

EDA 코드

train.json

7) 많이 매핑된 상위 태그 50개 시각화

In [35]:
```
# 1. unnest 데이터프레임인 plylst_tag_map 테이블에서 태그 이름 정렬 후 list로 묶기
plylst_tag_list_sort = plylst_tag_map.sort_values(by = ['id', 'tags']).groupby('id').tags.apply(list).reset_index(name = 'tag_list')

# 2. 집계를 위해 1번 테이블에서 list 타입을 문자열 타입으로 변경
plylst_tag_list_sort['tag_list'] = plylst_tag_list_sort['tag_list'].astype(str)

# 3. 태그 리스트 별 매핑되는 플레이리스트 수 집계 테이블 생성 : tag_list_plylst_cnt
tag_list_plylst_cnt = plylst_tag_list_sort.groupby('tag_list').id.nunique().reset_index(name = 'plylst_cnt')

# 4. 매핑 수 기준 상위 50개 필터링
tag_list_plylst_cnt = tag_list_plylst_cnt.nlargest(50, 'plylst_cnt')

# 5. plotting
plt.figure(figsize = (11, 15))
tag_list_plylst_cnt_plot = sns.barplot(y = 'tag_list', x = 'plylst_cnt', data = tag_list_plylst_cnt, color = 'grey')
tag_list_plylst_cnt_plot.set_title('매핑된 태그 리스트 상위 50개')
tag_list_plylst_cnt_plot.set_xlabel('매핑된 플레이리스트 수')
tag_list_plylst_cnt_plot.set_ylabel('태그 리스트')
plt.show()
```



코드

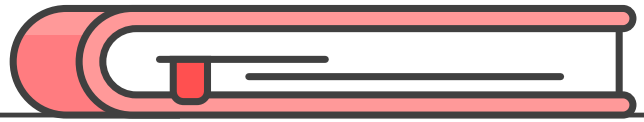
Split_data.py

1) 필요한 패키지 불러오기

```
# -*- coding: utf-8 -*-
import copy
import random
# fire 패키지는 파이썬에서 모든 객체를 command line interface로 만들어준다.
import fire
import numpy as np

# 같은 폴더에 arena_util.py에서 함수 호출
from arena_util import load_json
from arena_util import write_json
```

2) main

```
if __name__ == "__main__":
    fire.Fire(ArenaSplitter)
```



코드

Split_data.py

3) Run method

```python
# run method 지정해주기
def run(self, fname):
    # raondom shuffle 때문에 seed 지정
    random.seed(777)

    print("Reading data...\n")
    # json 파일 불러오기
    playlists = load_json(fname)
    # 불러온 파일 순서 섞기
    random.shuffle(playlists)
    print(f"Total playlists: {len(playlists)}")

    print("Splitting data...")
    # split
    train, val = self._split_data(playlists)
```

```python
    # train, val 나눈 데이터 json 파일 작성
    print("Original train...")
    write_json(train, "orig/train.json")
    print("Original val...")
    write_json(val, "orig/val.json")

    print("Masked val...")
    # masking 작업
    val_q, val_a = self._mask_data(val)
    write_json(val_q, "questions/val.json")
    write_json(val_a, "answers/val.json")
```

코드

Split_data.py

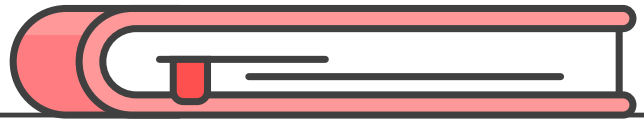
3) Run method

```python
# run method 지정해주기
def run(self, fname):
    # raondom shuffle 때문에 seed 지정
    random.seed(777)

    print("Reading data...\n")
    # json 파일 불러오기
    playlists = load_json(fname)
    # 불러온 파일 순서 섞기
    random.shuffle(playlists)
    print(f"Total playlists: {len(playlists)}")

    print("Splitting data...")
    # split
    train, val = self._split_data(playlists)
```

<arena._util.py>

```python
def load_json(fname):
    with open(fname, encoding="utf-8") as f:
        json_obj = json.load(f)

    return json_obj
```

```python
def _split_data(self, playlists):
    tot = len(playlists)
    # 8:2로 나누기
    train = playlists[:int(tot*0.80)]
    val = playlists[int(tot*0.80):]

    return train, val
```



코드

Split_data.py

<arena._util.py>

```
def write_json(data, fname):
    def _conv(o):
        # isinstance는 첫번째 객체가 뒤에 타입에 속해있는지 확인한다.
        # o라는 객체가 numpy 배열 int32, int64 타입에 속해있는지 확인한다.
        if isinstance(o, (np.int64, np.int32)):
            return int(o)
        # 속해져 있으면 error 발생
        raise TypeError
    # 부모 directory 경로 설정
    parent = os.path.dirname(fname)
    # 새로운 경로 만들기
    distutils.dir_util.mkpath("./arena_data/" + parent)

    with io.open("./arena_data/" + fname, "w", encoding="utf-8") as f:
        json_str = json.dumps(data, ensure_ascii=False, default=_conv)
        f.write(json_str)
```

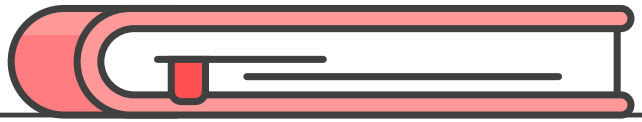
```
# train, val 나눈 데이터 json 파일 작성
print("Original train...")
write_json(train, "orig/train.json")
print("Original val...")
write_json(val, "orig/val.json")


print("Masked val...")
# masking 작업
val_q, val_a = self._mask_data(val)
write_json(val_q, "questions/val.json")
write_json(val_a, "answers/val.json")
```

코드

Split_data.py

```python
def _mask_data(self, playlists):
    playlists = copy.deepcopy(playlists)
    tot = len(playlists)
    song_only = playlists[:int(tot * 0.3)] # 곡만 존재
    song_and_tags = playlists[int(tot * 0.3):int(tot * 0.8)] # 곡, 태그 둘 다 존재
    tags_only = playlists[int(tot * 0.8):int(tot * 0.95)] # 태그만 존재
    title_only = playlists[int(tot * 0.95):] # 제목만 존재

    print(f"Total: {len(playlists)}, "
          f"Song only: {len(song_only)}, "
          f"Song & Tags: {len(song_and_tags)}, "
          f"Tags only: {len(tags_only)}, "
          f"Title only: {len(title_only)}")

    song_q, song_a = self._mask(song_only, ['songs'], ['tags'])
    songtag_q, songtag_a = self._mask(song_and_tags, ['songs', 'tags'], [])
    tag_q, tag_a = self._mask(tags_only, ['tags'], ['songs'])
    title_q, title_a = self._mask(title_only, [], ['songs', 'tags'])

    q = song_q + songtag_q + tag_q + title_q
    a = song_a + songtag_a + tag_a + title_a

    # random하게 섞기
    shuffle_indices = np.arange(len(q))
    np.random.shuffle(shuffle_indices)

    q = list(np.array(q)[shuffle_indices])
    a = list(np.array(a)[shuffle_indices])

    return q, a
```

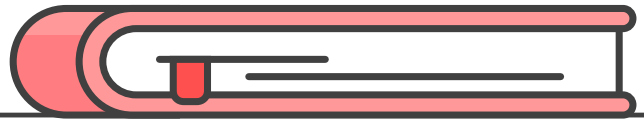
```python
def _mask(self, playlists, mask_cols, del_cols):
    # 깊은 복사 : 내부에 객체들까지 모두 새롭게 copy
    q_pl = copy.deepcopy(playlists)
    a_pl = copy.deepcopy(playlists)

    for i in range(len(playlists)):
        # 삭제할 컬럼
        for del_col in del_cols:
            q_pl[i][del_col] = []
            if del_col == 'songs':
                # 상위 100개 곡 추출
                a_pl[i][del_col] = a_pl[i][del_col][:100]
            elif del_col == 'tags':
                # 상위 10개 태그 추출
                a_pl[i][del_col] = a_pl[i][del_col][:10]
        # masking
        for col in mask_cols:
            mask_len = len(playlists[i][col])
            mask = np.full(mask_len, False)
            # 절반은 true, 절반은 false
            mask[:mask_len//2] = True
            np.random.shuffle(mask)

            q_pl[i][col] = list(np.array(q_pl[i][col])[mask])
            a_pl[i][col] = list(np.array(a_pl[i][col])[np.invert(mask)])

    return q_pl, a_pl
```



코드

Split_data.py

데이터를 불러와서 train, val 나누는데
Mask 작업을 같이 해준다.



코드

Most_popular.py

1) 필요한 패키지 불러오기

```python
# -*- coding: utf-8 -*-
import fire
from tqdm import tqdm

from arena_util import load_json
from arena_util import write_json
from arena_util import remove_seen
from arena_util import most_popular
```

2) main

```python
# MostPopular이 class로 정의되어있는데하나의 클래스를 interpreter 형식으로 인식시켜주게 만들어준다.
# interpreter란 사용자가 입력한 소스 코드를 실행하는 환경을 뜻한다.
if __name__ == "__main__":
    fire.Fire(MostPopular)
```

코드

Most_popular.py

3) Run method

```python
# train_fname=train.json, question_fname=val.json
def run(self, train_fname, question_fname):
    print("Loading train file...")
    train = load_json(train_fname)

    print("Loading question file...")
    questions = load_json(question_fname)

    print("Writing answers...")
    answers = self._generate_answers(train, questions)
    write_json(answers, "results/results.json")
```

```python
def load_json(fname):
    with open(fname, encoding="utf-8") as f:
        json_obj = json.load(f)

    return json_obj
```

```python
def write_json(data, fname):
    def _conv(o):
        # isinstance는 첫번째 객체가 뒤에 타입에 속해있는지 확인한다.
        # o라는 객체가 numpy 배열 int32, int64 타입에 속해있는지 확인한다.
        if isinstance(o, (np.int64, np.int32)):
            return int(o)
        # 속해져 있으면 error 발생
        raise TypeError
    # 부모 directory 경로 설정
    parent = os.path.dirname(fname)
    # 새로운 경로 만들기
    distutils.dir_util.mkpath("./arena_data/" + parent)

    with io.open("./arena_data/" + fname, "w", encoding="utf-8") as f:
        json_str = json.dumps(data, ensure_ascii=False, default=_conv)
        f.write(json_str)
```

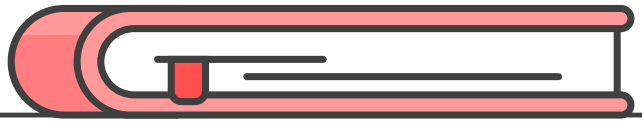
코드

Most_popular.py

```python
class MostPopular:
    def _generate_answers(self, train, questions):
        # 빈도수가 가장 높은 값을 출력한다.
        _, song_mp = most_popular(train, "songs", 200)
        _, tag_mp = most_popular(train, "tags", 100)

        answers = []

        for q in tqdm(questions):
            answers.append({
                "id": q["id"],
                "songs": remove_seen(q["songs"], song_mp)[:100],
                "tags": remove_seen(q["tags"], tag_mp)[:10],
            })

        return answers # question에 적힌 노래를 제외한 상위를 불러오기
```

```python
def most_popular(playlists, col, topk_count):
    # 리스트 개수 세기
    c = Counter()
    # 각 플레이리스트마다 컬럼을 count 시켜서 업데이트 해주기
    for doc in playlists:
        c.update(doc[col])
    # most_common : 빈도수 높은것부터출력
    # topk : 반환형이 (song,개수) 의 tuple 형태
    topk = c.most_common(topk_count)
    return c, [k for k, v in topk]
```

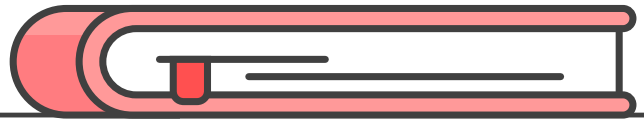
```python
def remove_seen(seen, l):
    # set 자료구조를 통해 중복을 제거한다.
    seen = set(seen)
    #실제 데이터에는 없는 태그와 수록곡을 불러온다.
    return [x for x in l if not (x in seen)]
```



코드

Most_popular.py

Train data를 불러오고 val data를 불러와 답을 작성하는 파일이다.



코드

Genre_most_popular.py

1) 필요한 패키지 불러오기

```
# -*- coding: utf-8 -*-
from collections import Counter

import fire
from tqdm import tqdm

from arena_util import load_json
from arena_util import write_json
from arena_util import remove_seen
from arena_util import most_popular
```

2) main

```
if __name__ == "__main__":
    fire.Fire(GenreMostPopular)
```

코드

Genre_most_popular.py

3) Run method

```
def run(self, song_meta_fname, train_fname, question_fname):
    print("Loading song meta...")
    song_meta_json = load_json(song_meta_fname)

    print("Loading train file...")
    train_data = load_json(train_fname)

    print("Loading question file...")
    questions = load_json(question_fname)

    print("Writing answers...")
    answers = self._generate_answers(song_meta_json, train_data, questions)
    write_json(answers, "results/results.json")
```

코드

```python
def _generate_answers(self, song_meta_json, train, questions):
    # key를 song_id value를 해당 song_id에 대한 정보로 dictionary 생성
    song_meta = {int(song["id"]): song for song in song_meta_json}
    # 상위 200개 곡
    song_mp_counter, song_mp = most_popular(train, "songs", 200)
    # 상위 100개 태그
    tag_mp_counter, tag_mp = most_popular(train, "tags", 100)
    song_mp_per_genre = self._song_mp_per_genre(song_meta, song_mp_counter)

    answers = []
    for q in tqdm(questions):
        genre_counter = Counter()

            for sid in q["songs"]:
                for genre in song_meta[sid]["song_gn_gnr_basket"]:
                    genre_counter.update({genre: 1})

            top_genre = genre_counter.most_common(1)
            # 가장 인기있는 장르가 존재하면
            if len(top_genre) != 0:
                # 해당 장르에서 가장 많이 등장한 song 추천
                cur_songs = song_mp_per_genre[top_genre[0][0]]
            else:
                # 아니면 가장 많이 등장한 노래 추천
                cur_songs = song_mp

            answers.append({
                "id": q["id"],
                "songs": remove_seen(q["songs"], cur_songs)[:100],
                "tags": remove_seen(q["tags"], tag_mp)[:10]
            })

        return answers
```

```python
47  def most_popular(playlists, col, topk_count):
48      # 리스트 개수 세기
49      c = Counter()
50      # 각 플레이리스트마다 컬럼을 count 시켜서 업데이트 해주기
51      for doc in playlists:
52          c.update(doc[col])
53      # most_common : 빈도수 높은것부터출력
54      # topk : 반환형이 (song,개수) 의 tuple 형태
55      topk = c.most_common(topk_count)
56      return c, [k for k, v in topk]
```

```python
class GenreMostPopular:
    def _song_mp_per_genre(self, song_meta, global_mp):
        res = {}
        # key는 song_meta에 있던 genre, value는 해당 genre에 속한 id값
        for sid, song in song_meta.items():
            for genre in song['song_gn_gnr_basket']:
                res.setdefault(genre, []).append(sid)

        for genre, sids in res.items():
            # Before: res {genre : song_id}
            # After: res {genre : Counter(song_id : 개수)}
            res[genre] = Counter({k: global_mp.get(int(k), 0) for k in sids})
            # 가장 많은 상위 200개 song_id만 추출
            res[genre] = [k for k, v in res[genre].most_common(200)]

        return res
```

코드 결과

In [3]:
```
results = pd.read_json("results.json")
```

In [2]:
```
import pandas as pd
```

In [4]:
```
results
```

Out[4]:

	id	songs	tags
0	147640	[348200, 443914, 362966, 518420, 553171, 44016...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
1	149981	[213435, 222305, 383011, 166761, 701801, 49996...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
2	82689	[663256, 140867, 177460, 554751, 686809, 41328...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
3	39967	[443914, 518420, 704707, 588471, 326204, 45566...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
4	7531	[144663, 675115, 396828, 701557, 520093, 65049...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
...	...	...	...
23010	7433	[696494, 202185, 571790, 595181, 52192, 29793,...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
23011	112943	[144663, 116573, 357367, 366786, 654757, 13314...	[기분전환, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤, 카페]
23012	134619	[357367, 174749, 461341, 169984, 348200, 50503...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 밤, 추억]
23013	113348	[144663, 349492, 675115, 463173, 396828, 42155...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]
23014	132400	[705515, 321724, 335757, 205939, 650367, 15464...	[기분전환, 감성, 휴식, 발라드, 잔잔한, 드라이브, 힐링, 사랑, 새벽, 밤]

23015 rows × 3 columns

결과 평가 지표

nDCG로 모델 성능 평가

sample score

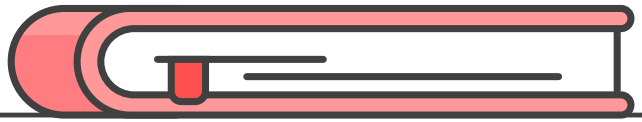
Music nDCG: 0.165272

Tag nDCG: 0.329114

Score: 0.189849

```
def _idcg(self, l):
    return sum((1.0 / np.log(i + 2) for i in range(l)))

def __init__(self):
    self._idcgs = [self._idcg(i) for i in range(101)]

def _ndcg(self, gt, rec):
    dcg = 0.0
    for i, r in enumerate(rec):
        if r in gt:
            dcg += 1.0 / np.log(i + 2)

    return dcg / self._idcgs[len(gt)]
```

```
    music_ndcg = 0.0
    tag_ndcg = 0.0

    for rec in rec_playlists:
        gt = gt_dict[rec["id"]]
        music_ndcg += self._ndcg(gt["songs"], rec["songs"][:100])
        tag_ndcg += self._ndcg(gt["tags"], rec["tags"][:10])

    music_ndcg = music_ndcg / len(rec_playlists)
    tag_ndcg = tag_ndcg / len(rec_playlists)
    score = music_ndcg * 0.85 + tag_ndcg * 0.15

    return music_ndcg, tag_ndcg, score

def evaluate(self, gt_fname, rec_fname):
    try:
        music_ndcg, tag_ndcg, score = self._eval(gt_fname, rec_fname)
        print(f"Music nDCG: {music_ndcg:.6}")
        print(f"Tag nDCG: {tag_ndcg:.6}")
        print(f"Score: {score:.6}")
    except Exception as e:
        print(e)
```



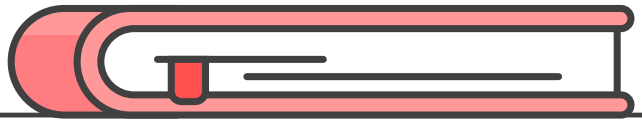
두번째 코드

```
val[(val['tags']+val['songs']).map(len) == 0]
```

	tags	id	plylst_title	songs	like_cnt	updt_date
1	[]	131447	앨리스테이블	[]	1	2014-07-16 15:24:24.000
9	[]	142007	기분 좋은 재즈와 함께 만드는 달달한 하루	[]	0	2015-06-22 09:11:02.000
35	[]	65114	■■■■ 사랑,그리고이별 ■■■■	[]	6	2010-10-27 10:34:34.000
57	[]	87700	마쉬멜로우같은 멜로우한 음악	[]	6	2016-01-14 10:19:30.000
71	[]	35271	공부와 독서를 위한 #Newage	[]	10	2020-01-17 15:46:20.000
...	...	...	...	...	...	...
22903	[]	140513	10년이 지나 들어도 좋은 감성 Ballad	[]	405	2016-01-11 10:58:05.000
22920	[]	124704	가사의 의미와 뜻은모른다!! 오직 멜로디로만 선곡한 팝송!!	[]	27	2016-02-05 12:31:59.000
22981	[]	13045	* 카페 느낌 샹송♭	[]	38	2011-07-12 00:58:39.000
22991	[]	32537	컨트리 황제 조니 캐시가 선 레코드 시절 발표한 초기 대표작	[]	28	2019-06-17 14:22:48.000
22996	[]	86721	해 저무는 밤	[]	5	2016-04-27 15:32:55.000

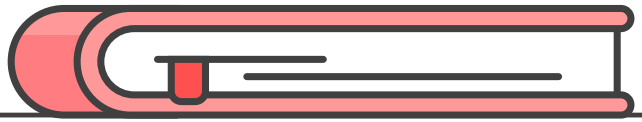
```
train.tail()
```

	tags	id	plylst_title	songs	like_cnt	updt_date
115066	[록메탈, 밴드사운드, 록, 락메탈, 메탈, 락, extreme]	120325	METAL E'SM #2	[429629, 441511, 612106, 516359, 691768, 38714...	3	2020-04-17 04:31:11.000
115067	[일렉]	106976	빠른 리스너를 위한 따끈따끈한 최신 인기 EDM 모음!	[321330, 216057, 534472, 240306, 331098, 23288...	13	2015-12-24 17:23:19.000
115068	[담시, 가족, 눈물, 그리움, 주인공, 나의_이야기, 사랑, 친구]	11343	#1. 눈물이 앞을 가리는 나의_이야기	[50512, 249024, 250608, 371171, 229942, 694943...	4	2019-08-16 20:59:22.000
115069	[잔잔한, 버스, 퇴근버스, Pop, 풍경, 퇴근길]	131982	퇴근 버스에서 편히 들으면서 하루를 마무리하기에 좋은 POP	[533534, 608114, 343608, 417140, 609009, 30217...	4	2019-10-25 23:40:42.000
115070	[노래추천, 팝송추천, 팝송, 팝송모음]	100389	FAVORITE POPSONG!!!	[26008, 456354, 324105, 89871, 135272, 143548,...	17	2020-04-18 20:35:06.000



두번째 코드

1. 데이터를 보면 제목만 주어지고 songs 와 tags에 대한 정보가 전혀 없는 경우가 있다.
2. Tags와 플레이리스트를 보면 제목에 있는 단어를 그대로 tags에 넣는 경우가 많기 때문에 단어를 파악할 필요가 있다.
3. 따라서 데이터의 제목을 활용하여 전처리를 해보자!



두번째 코드

```
import json
import pandas as pd

with open("/content/gdrive/My Drive/project_data/train.json",'r',encoding='utf-8',errors='ignore') as
F:
    data = json.load(F)
    train = pd.DataFrame(data)

with open("/content/gdrive/My Drive/project_data/val.json",'r',encoding='utf-8',errors='ignore') as F:
    data = json.load(F)
    val = pd.DataFrame(data)
```

```
print(train.shape)
print(val.shape)
```

```
(115071, 6)
(23015, 6)
```

```
import json
import re
from collections import Counter
from typing import *

import matplotlib.pyplot as plt
import numpy as np
from khaiii import KhaiiiApi
```

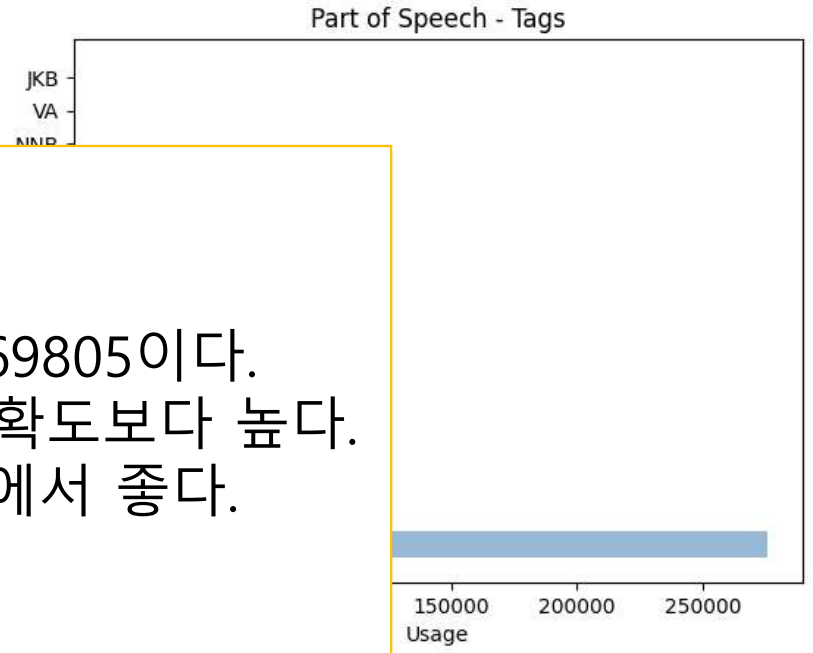
두번째 코드

```python
def re_sub(series: pd.Series) -> pd.Series:
    series = series.str.replace(pat=r'[ㄱ-ㅎ]', repl=r'', regex=True)  # ㅋ 제거용
    series = series.str.replace(pat=r'[^ㅠㅜㄱ-ㅎ가-힣\s]', repl=r'', regex=True)  # 특수문자 제거
    series = series.str.replace(pat=r'[ ]{2,}', repl=r' ', regex=True)  # 공백 제거
    series = series.str.replace(pat=r'[\u3000]+', repl=r'', regex=True)  # u3000 제거
    return series

def flatten(list_of_list : List) -> List:
    flatten = [j for i in list_of_list for j in i]
    return flatten

def get_token(title: str, tokenizer)-> List[Tuple]:

    if len(title)== 0 or title== ' ':  # 제목이 공백인 경우 tokenizer에러 발생
        return []

    result = tokenizer.analyze(title)
    result = [(morph.lex, morph.tag) for split in result for morph in split.morphs]  # (형태소, 품사) 튜플의 리스트
    return result

def get_all_tags(df) -> List:
    tag_list = df['tags'].values.tolist()
    tag_list = flatten(tag_list)
    return tag_list
```

```python
tokenizer = KhaiiiApi()
all_tag = get_all_tags(train)
token_tag = [get_token(x, tokenizer) for x in all_tag]  # 태그를 형태소 분석
```

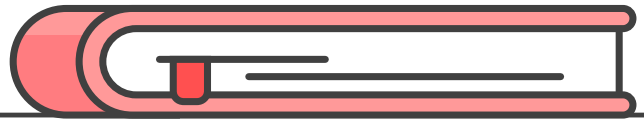
두번째 코드

	tags	id	plylst_title	songs	like_cnt	updt_date	ply_token
0	[락]	61281	여행같은 음악	[525514, 129701, 383374, 562083, 297861, 13954...	71	2013-12-19 18:36:19.000	[(여행, NNG), (음악, NNG)]
1	[추억, 회상]	10532	요즘 너 말야	[432406, 675945,			[(요즘,
2	[까페, 잔잔한]	76951	편하게 잔잔하게 들을 수 있는 곡				
3	[연말, 눈오는날, 캐럴, 분위기, 따듯한, 크리스마스캐럴, 겨울노래, 크리스마스,...	147456	크리스마스 분위기에 흠뻑 취하고 싶을때				
4	[댄스]	27616	추억의 노래				
5	[운동, 드라이브, Pop, 트로피컬하우스, 힐링, 기분전환, 2017, 팝, 트렌...	69252	2017 Pop Trend				
6	[짝사랑, 취향저격, 슬픔, 고백, 사랑, 이별]	45339	짝사랑고백사랑이별슬픔 감성을 자극하는곡들				(곡, NNG)]
7	[잔잔한, 추억, 회상]	36557	멍청이 내맘도 몰라	[496913, 632529, 501426, 515574, 411161, 10341...	5	2008-09-23 22:32:02.000	[(맘, NNG)]
8	[일렉트로니카, 포크, 메탈, 락, 댄스, 인디]	70741	DANCING IN THE MOONLIGHT 01	[634861, 270738, 163936, 692209, 449477, 56342...	0	2019-11-30 21:17:59.000	[(01, SN)]



Tags 예측 정확도는 0.169805이다.
앞에 나왔던 코드의 tags 정확도보다 높다.
이 코드는 성능 개선면에서 좋다.

형태소 분석을 해서 tags에 가장 자주 등장하는 품사들만 남겨두고 나머지를 제거한 채 tags에 대한 예측을 실시



세번째 코드

```
# 필요한거 import
from implicit.evaluation import  *
from implicit.als import AlternatingLeastSquares as ALS
from implicit.bpr import BayesianPersonalizedRanking as BPR
import numpy as np
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
from sklearn.utils import shuffle
from scipy.sparse import *
```

```
# 파일 불러오기
import pandas as pd
os.chdir('c:/temp')
tr = pd.read_json("train.json")
te = pd.read_json("val.json")
```

```
# 태그랑 언급횟수 딕셔너리로 저장
ret = []
for tag in tr.tags.tolist():
    ret += tag
from collections import Counter
r = dict(Counter(ret))
```

```
# 내림차순 정렬
r = sorted(r.items(), key=lambda x: -x[1])
```

```
# 상위권 태그 추출
top_tags = [x[0] for x in r[:1000]]
```

세번째 코드

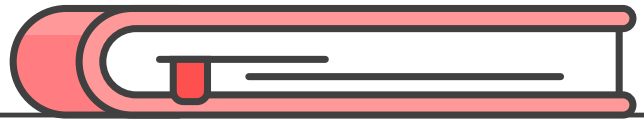
```
# train, val 안 songs,tags,ids 리스트로 변환
tr_songs = tr.songs.tolist()
te_songs = te.songs.tolist()
tr_tags = tr.tags.tolist()
te_tags = te.tags.tolist()
te_ids = te.id.tolist()
```

```
# tr 변수에 view, tag_to_idx 값 추가
from itertools import groupby
tr = []
iid_to_idx = {}
tag_to_idx = {}
idx = 0

for i, l in enumerate(tr_songs):
    view = l
    for item_id in view:
        if item_id not in iid_to_idx:
            iid_to_idx[item_id] = idx
            idx += 1
    view = [iid_to_idx[x] for x in view]
    tr.append(view)

idx = 0
n_items = len(iid_to_idx)
for i, tags in enumerate(tr_tags):
    for tag in tags:
        if tag not in tag_to_idx:
            tag_to_idx[tag] = n_items + idx
            idx += 1
    tr[i].extend([tag_to_idx[x] for x in tags])
n_tags = len(tag_to_idx)
# n_items = len(iid_to_idx)
# n_tags = len(tag_to_idx)
```

```
# te변수에 ret 추가
from itertools import groupby
te = []

idx = 0
for i, l in enumerate(te_songs):
    view = l
    ret = []
    for item_id in view:
        if item_id not in iid_to_idx:
            continue
        ret.append(iid_to_idx[item_id])
    te.append(ret)
idx = 0
for i, tags in enumerate(te_tags):
    ret = []
    for tag in tags:
        if tag not in tag_to_idx:
            continue
        ret.append(tag)
    te[i].extend([tag_to_idx[x] for x in ret])
```

```
# tr변수 내 데이터 섞기
tr = shuffle(tr)
```

```
# 데이터 딕셔너리화
idx_to_iid = {x:y for(y,x) in iid_to_idx.items()}
idx_to_tag = {(x - n_items):y for(y,x) in tag_to_idx.items()}
```



세번째 코드

```
# 희소행렬 생성
from scipy.sparse import csr_matrix

tr_csr = csr_matrix(tr.astype(float), (n_tags,n_items))
te_csr = csr_matrix(te.astype(float), (n_tags,n_items))

# tr_csr = rec_util.lil_to_csr(tr, (len(tr), n_tags + n_items))
# te_csr = rec_util.lil_to_csr(te, (len(te), n_tags + n_items))
```

```
# 희소행렬 행방향으로 붙이기
import scipy.sparse
r = scipy.sparse.vstack([te_csr, tr_csr])
r = csr_matrix(r)
```

```
# als 모델링
als_model = ALS(factors=128, regularization=0.08)
als_model.fit(r.T * 15.0)
```

```
# 적용
item_model = ALS(use_gpu=False)
tag_model = ALS(use_gpu=False)
item_model.user_factors = als_model.user_factors
tag_model.user_factors = als_model.user_factors
```

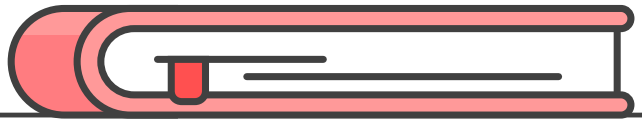
```
item_model.item_factors = als_model.item_factors[:n_items]
tag_model.item_factors = als_model.item_factors[n_items:]
```

세번째 코드

```
# 학습된 추천 아이템, 태그 저장
item_ret = []
tag_ret = []
from tqdm.auto import tqdm
for u in tqdm(range(te_csr.shape[0])):
    item_rec = item_model.recommend(u, item_rec_csr, N=100)
    item_rec = [idx_to_iid[x[0]] for x in item_rec]
    tag_rec = tag_model.recommend(u, tag_rec_csr, N=100)
    tag_rec = [idx_to_tag[x[0]] for x in tag_rec if x[0] in idx_to_tag]
    item_ret.append(item_rec)
    tag_ret.append(tag_rec)
```

```
tag_model.item_factors
```

```
array([[ 3.55746090e-01, -5.69548428e-01, -5.59754260e-02, ...,
         4.83457536e-01,  1.95368275e-01,  1.14752978e-01],
       [ 1.13077201e-01,  4.99642581e-01,  1.01401411e-01, ...,
         8.70525062e-01,  5.08861303e-01,  4.67959046e-01],
       [ 1.34080434e-02,  5.13347864e-01,  1.56607270e-01, ...,
         6.01837337e-01,  4.08071131e-01,  3.32719356e-01],
       ...,
       [-5.10524551e-04,  3.28546972e-04,  8.81741638e-04, ...,
         8.72330857e-04,  1.68862971e-04,  1.18788111e-03],
       [ 9.88858286e-04,  1.02399255e-03,  3.29685328e-03, ...,
         1.42479164e-03, -1.69632386e-03, -1.65981124e-03],
       [ 9.73270435e-05,  1.62653669e-04, -2.30579590e-05, ...,
         1.98975016e-04,  2.20649337e-04, -1.06205756e-04]], dtype=float32)
```



결론

활동을 통해 얻은 경험

1. 추천 알고리즘에 대해 알아보고 싶었는데 팀 프로젝트를 계기로 공부하게 되어서 좋았다.

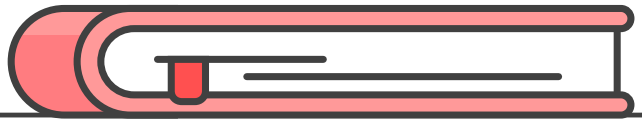
2. json 파일 다루는 방법 말고도 처음 보는 모델과 평가 기준 등등을 알게 되어서 신기하다.

3. 추천 알고리즘 관련 이론 및 수학적 지식을 코드로 작성해 보면서 실력 향상에 도움이 됨.

아쉬운 점

1. 여러가지 feature들을 담은 데이터들이 많았는데, 주로 train, test 데이터만 쓴 것 같아서 아쉬웠다.

2. 처음 공부하는 추천 알고리즘이었는데 처음부터 난이도가 높은 주제를 다뤄서 스스로 해결하지 못한 점이 아쉬웠다.



Reference

[1] Train.json EDA, https://arena.kakao.com/forum/topics/191
[2] collaborative filtering, https://arena.kakao.com/forum/topics/227
[3] Matrix factorization, https://arena.kakao.com/forum/topics/200
[4] baseline code, https://github.com/kakao-arena/melon-playlist-continuation
[5] study baseline code, https://bladejun.tistory.com/21