Kakao Arena Melon Playlist Continuation
추천 알고리즘 비교 분석 보고서

박기범
인하대학교 컴퓨터공학과
12191601@inha.edu

김규리
인하대학교 통계학과
12191881@inha.edu

마민경
인하대학교 통계학과
12191890@inha.edu

이예원
인하대학교 통계학과
12172171@inha.edu

함우석
인하대학교 산업경영공학과
12150378@inha.edu

1. 서론

추천 시스템(Recommender System)은 사용자가 관심을 가질만한 정보를 추천하는 것이다. 이는 기업의 사용자 유치에 영향을 미치는 요소로 작용하기 때문에 많은 기업들이 추천 시스템 연구·개발을 진행하고 있다. 추천 시스템이 가장 활발하게 이용되고 있는 분야는 상품 판매, 음악 스트리밍, OTT서비스(Over-the-top media service)이다.

음악 스트리밍 서비스인 Melon(이하 멜론)은 2021년 5월 기준 국내 음악/오디오 주요 앱 점유율에서 31.41%로 [1] 가장 많은 사람들이 사용하고 있는 음악 스트리밍 앱이다. 사용자가 가장 많다는 것은 추천 시스템 훈련과 검증에 필요한 데이터가 충분하다고 볼 수 있다. 카카오에서 운영중인 '카카오 아레나'에서는 지난 2020년 4월, 멜론 플레이리스트 데이터와 멜론에서 제공중인 음악 데이터 일부를 활용한 추천 시스템을 개발하는 대회인 'Melon Playlist Continuation'을 진행했다. 이는 2007년에서 2009년까지 진행한 'Netflix Prize'처럼 실제 서비스에 활용할 추천 알고리즘을 대회를 통해 수집하는 것이 목적이다.

이 보고서에서는 'Melon Playlist Continuation'에서 제공하는 데이터를 활용하여 전통적인 방식의 추천 알고리즘인 Content-based Filtering, Collaborative-Filtering을 통해 추천 모델을 개발하고 각 모델들의 성능을 비교할 것이다.

2. 대회 소개 및 데이터 구조

이 단락에서는 카카오 아레나에서 제공하는 'Melon Playlist Continuation'의 목표와 평가지표를 설명하고 모델 개발에 사용되는 데이터의 구조를 알아볼 것이다.

2.1 대회 소개

'Melon Playlist Continuation'은 카카오 아레나에서 2020년 4월부터 9월까지 진행한 추천 알고리즘 개발 대회이다. 실제 Melon에 있는 플레이리스트 메타 데이터와 일부 노래 메타 데이터를 통해 추천 플레이리스트 목록과 플레이리스트에 어울리는 태그를 구성하는 것이 목적이다. 대회 개최 배경은 플레이리스트의 절반을 보여주고 나머지 숨겨진 절반을 예측하는 모델이 있다면, 전체 플레이리스트가 제공될 때 해당 플레이리스트와 어울리는 새로운 플레이리스트를 구성할 수 있을 것이라는 것이다. [2]

이 대회의 자세한 목표는 테스트 셋에 제공되는 플레이리스트의 정보를 참조하여 해당 플레이리스트에 있었을 것으로 생각되는 곡 리스트 100개와 태그 리스트 10개를 예측하는 것이다. 곡과 태그는 모두 평가지표로 nDCG를 사용했다.

평가지표는 다음과 같이 정의된다:

$$\mathrm{CG_n} = \sum_{i=0}^{n} rel_i$$

$$\mathrm{DCG}_n = \sum_{i=0}^{n} \frac{rel_i}{log_2(i+1)}$$

$$\mathrm{nDCG}_n = \frac{\mathrm{DCG}_n}{\mathrm{IDCG}_n}$$

IDCG_n ≔ ideal DCG about answer data

CG(Cumulative Gain)은 관련성(relevance, rel) 점수를 합한 값이다. 관련성 점수란 사용자가 추천한 아이템을 얼마나 선호하는 지를 나타내는 점수이다. 일반적으로 관련성 점수는 아이템에 대한 사용자의 평점이

활용된다. 이번 대회에서는 플레이리스트의 음악 선호 강도는 모두 동일하다고 가정하기 때문에 관련성 점수는 모두 1로 통일하였다. CG는 단순히 사용자의 아이템 선호 강도만을 나타내므로 아이템의 랭킹을 적용하기 어렵다는 문제가 있다. 이를 해결하고자 DCG(Discounted Cumulative Gain)이 도입되었다. DCG는 CG에 각 아이템의 랭킹을 분모에 가중치로 적용하여 discounted 된 관련도를 계산하는 지표이다. 랭킹값이 커질수록 분모 값이 커지므로 하위 랭크일수록 관련성 점수에 페널티가 부여된다. 즉, 상위 랭크 아이템을 잘 추천하는 것이 하위 랭크 아이템을 잘 추천하는 것보다 높은 점수를 얻게 된다. 하지만 누적합인 DCG는 n이 커질수록 DCG값이 커지게 되므로 n 값에 영향을 받게 된다. 따라서 서로 다른 개수의 추천 리스트인 경우 정확한 비교를 하기 어렵다. 따라서 정규화과정을 통해 n 값과 무관하게 스케일을 일정하게 조정한다. 정규화 과정에 사용되는 IDCG는 ground truth에서 relevance를 기준으로 오름차순 정렬한 후 상위 n개로 DCG를 계산하는 것이다. 이 대회에서는 각 노래의 relevance는 모두 1이므로 정답 플레이리스트 수록 곡 순서대로 100개의 DCG를 IDCG로 설정한다.

2.2 데이터 구조

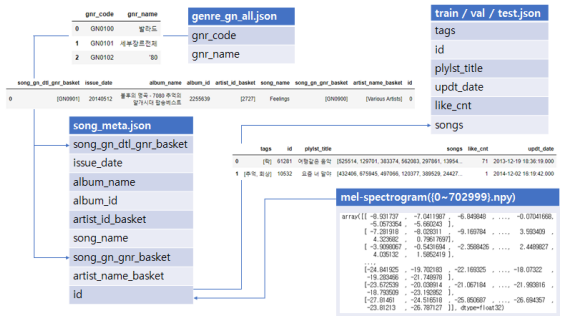

그림 1 전체 데이터 스키마 [3]

멜론에서 서비스하고 있는 일부 노래의 메타데이터를 저장하고 있는 song_meta.json이다. 노래제목, 수록 앨범 제목, 앨범 id, 노래 id, 노래의 대장르와 세부장르, 아티스트명, 발매일을 저장하고 있다. 총 707,989개의 곡 메타데이터를 저장하고 있으며 train, validation, test 데이터에 수록된 모든 곡의 메타데이터가 포함되어 있다. song_meta.json에 있는 노래 중 일부 데이터는 해당 곡의 멜로디 스펙트럼을 numpy array형태로 저장하고 있는 mel-spectrogram 데이터가 있다. 해당 데이터는 Content-based Filtering에서 사용할 수 있는 아이템의 특성이다. 하지만 전체 데이터의 크기가 약 240GB이기 때문에 주최측에서도 컴퓨팅 자원이 충분한 경우에만 사용하는 것을 권장하고 있다.

모델 학습, 검증, 평가에 사용되는 train, validation, test.json에는 플레이리스트 메타데이터를 저장하고 있다. train.json은 115,071개, val.json은 23,015개, test.json은 10,740개의 플레이리스트 정보가 저장되어 있다. 해당 플레이리스트의 제목, 곡 목록, 태그 목록, 좋아요 수, 플레이리스트 등록일, 고유 id를 저장하고 있다. 학습에 사용되는 train에는 태그와 곡 목록이 모두 제시되어 있으며 val.json과 test.json에는 태그 리스트의 일부와 플레이리스트 곡 목록 리스트의 일부가 제시된다.

3. 모델 개발

추천 알고리즘에는 전통적인 행렬 방식의 Content-based Filtering(CBF), Collaborative Filtering(CF)이 있고 딥러닝 방식을 활용하는 Autoencoder와 Neural Collaborative Filtering [4] 등이 있다. 이 보고서에서는 전통적인 방식의 Content-based Filtering과 User-based CF, Memory-based CF인 Matrix Factorization을 사용한 모델로 성능 비교를 진행할 것이다.

3.1 Content-based Filtering

Content-based Filtering(CBF)이란 사용자가 높은 평점을 주거나 큰 관심을 갖은 아이템 x가 있다면 해당 아이템과 유사한 아이템 y를 추천하는 방식이다. 다른 유저의 정보가 필요하지 않고 아이템 전체에서 추천 목록을 찾는 방식이기 때문에 유명하지 않거나 새로운 아이템에 대해서도 추천이 가능하다는 장점이 있다. 하지만 신규 사용자의 경우 아이템을 평가한 정보가 없기 때문에 추천이 어렵다는 점이 있고 아이템에 적절한 feature를 찾기가 어렵다는 단점이 존재한다.

노래를 추천하는 이번 대회의 특성상 노래의 특성으로는 아티스트, 장르, 멜로디가 대표적으로 있다. 멜론에서는 음악을 추천할 때, CBF의 경우 음악의 멜로디 스펙트럼을 사용하는 것으로 보인다. [5] 하지만 컴퓨팅 자원의 한계로 멜로디 스펙트럼 사용에는 제한이 있다. 따라서 노래의 특징을 가장 잘 나타내는 것으로 보이는 장르를 아이템 특성으로 활용할 것이다. 전체 데이터에서 자주 나타나는 장르의 가중치를 낮추고 해당 플레이리스트의 특징을 나타내는 장르의 가중치를 높이고자 IDF를 가중치로 활용하였다.

IDF는 다음과 같이 정의된다:

$$df(t) = \frac{|d \in D: t \in d|}{|D|}$$

$$idf(t, D) = log\frac{|D|}{1 + |\{d \in D: t \in d\}|}$$

$|D|$ = 전체 문서의 수

$|d \in D: t \in d|$ = 단어 t가 포함된 문서의 수

DF(Document Frequency)는 전체 데이터에서 특정 데이터가 등장한 수에만 관심을 갖는다. 일반적으로

이 가중치는 자연어 처리, 검색엔진에서 자주 사용되는 가중치이다. 문서에서 is, a, the, 국어의 조사와 같은 단어들은 문서 전체에서 높은 빈도를 갖는다. 하지만 그 자체로는 어떤 의미를 갖거나 높은 의미를 지니지 않는 경우가 많다. DF에 역수를 취한 IDF(Inverse Document Frequency)는 빈도수가 높은 정보의 가중치를 낮춘다.

	g_1	g_2	g_3	g_4	g_5
s_1	idf_{g_1}	0	0	idf_{g_4}	0
s_2	0	idf_{g_2}	0	0	0
s_3	0	idf_{g_2}	idf_{g_3}	0	0
s_4	0	0	0	0	idf_{g_5}
s_5	0	idf_{g_2}	0	idf_{g_4}	0

그림 2 노래의 장르 IDF를 저장하는 sparse matrix, idf_{g_n}은 장르 g_n의 idf 값

song_meta.json에 있는 707,989곡 전체의 장르 IDF를 저장하는 matrix를 생성했다. 이 matrix에 예측을 위해 입력으로 들어온 플레이리스트가 보유하고 있는 곡의 장르를 원핫인코딩으로 표현하는 $genre\ size \times 1$ matrix와 내적을 한 결과인 $song\ size \times 1$ matrix를 가중치로 정렬한 후 상위 100개의 곡을 추천하는 방식을 활용했다. 태그의 경우 모든 플레이리스트를 가장 많이 등장한 태그 10개로 추천하였다.

```
song_gnr_matrix := song genre idf sparse matrix
test := test datasets
top_songs := song list sorted by number of appearances
procedure Recom(pids)
    for pid in pids
        song_already := get_songlist(test_pid)
        if size(song_already) = 0
            cand_song = top_songs_100
        else
            cand_song = argsort_asc(song_gnr_matrix ·playlist_genre size × 1)
            cand_song = list(cand_song not in song_already)_100
```

그림 3 IDF 기반 CBF 추천 알고리즘

3.2 Collaborative Filtering

Collaborative Filtering에는 memory-based 방식과 model-based 방식이 존재한다. 유사한 유저를 기반으로 추천 목록을 만드는 user-based CF와 유사한 아이템을 기반으로 추천 목록을 만드는 item-based CF는 memory-based 방식이다. Sparse matrix를 Singular Value Decomposition(SVD)를 활용해 공통 요소로 분해하여 모델 학습을 진행하는 Matrix Factorization은 model-based 방식이다. 3.2에서는 memory-based 방식 중 user-based CF를 활용한 방식의 알고리즘을 설명하겠다.

알고리즘에 사용한 추천 rule은 다음과 같이 총 2가지로 설정하였다:

1. input playlist의 수록 곡과 train과 val에 존재하는 모든 각 playlist에 대해 겹치는 수록 곡의 수를 가중치로 계산
2. 1의 과정에서 각 플레이리스트의 수록 곡 표시를 원핫인코딩이 아닌 해당 플레이리스트에서 각 노래의 순위를 가중치로 추가한 값으로 설정

3.2.1 Rule 1 : Song intersection

	s_1	s_2	s_3	s_4	s_5
p_1	1	0	0	1	0
p_2	0	1	0	0	0
p_3	0	1	1	0	0
p_4	0	0	0	0	1
p_5	0	1	0	1	0

그림 4 playlist - song sparse matrix one-hot encoding

첫번째 rule을 적용할 때 사용되는 sparse matrix는 그림 4와 같이 구성되어 있다. 특정 플레이리스트 p_n의 수록 곡 s_n이 있는 부분을 1로 표시하는 원핫인코딩 방식으로 되어있다. CF는 기존에 존재하는 모든 정보를 사용해야 하므로 train의 플레이리스트와 val의 플레이리스트에 등장하는 모든 곡을 feature로 사용한다. 이 sparse matrix는 대략 13만 × 68만의 matrix이다. 모든 내용을 실제로 저장하는 matrix는 많은 메모리를 차지하므로 sparse matrix 구현에는 scipy 라이브러리의 csr_matrix함수를 활용하여 데이터가 존재하는 부분만 저장하였다.

```
song_sparse := playlist song sparse matrix
test := test datasets
top_songs := song list sorted by number of appearances
procedure Recom(pids)
    for pid in pids
        song_already = get_songlist(test_pid)
        Onehot_song size × 1 := vector that checked exist songs in test_pid
        inter_songs = song_sparse · Onehot_song size × 1
        cand_song = Song_sparse^T · inter_songs
        cand_song = argsort_asc(cand_song)
        cand_song = list(cand_song not in song_already)_100
```

그림 5 Collaborative Filtering 추천 알고리즘

playlist-song sparse matrix와 입력으로 들어온 플레이리스트가 보유하고 있는 곡을 원핫인코딩으로 표현

하는 $\text{Onehot}_{songsize\times 1}$ 벡터와 내적을 한다. 내적 결과로 나오는 inter_songs는 playlist size × 1 크기의 벡터이다. 해당 벡터는 각 플레이리스트와 입력 플레이리스트와의 겹치는 노래 수를 갖고 있다. 이는 곡에 부여되는 가중치로 활용되며 playlist-song sparse matrix를 transpose한 행렬과 내적 연산을 수행하면 각 노래에 대한 추천 score가 나온다. 이를 추천 score로 정렬하여 겹치지 않는 상위 100곡을 추천하면 된다. 태그도 동일한 방식으로 수행한다.

3.2.2 Rule2 : Rule1 + Song rank

	s_1	s_2	s_3	s_4	s_5
p_1	1/1	0	0	1/2	0
p_2	0	1/1	0	0	0
p_3	0	1/2	1/1	0	0
p_4	0	0	0	0	1/1
p_5	0	1/2	1/3	1/1	0

그림 6 playlist-song inverse rank sparse matrix

두번째 rule을 적용할 때 사용되는 sparse matrix는 그림 6과 같이 구성된다. 기본적인 추천 방식은 Rule1과 동일하다. 하지만 음악의 순위가 중요하기 때문에 각 플레이리스트에 있는 노래의 순위를 가중치로 적용한다. 순위가 높을수록 해당 플레이리스트에서 높은 가중치를 갖게 만들었다. 이후 추천 방식은 그림 5의 알고리즘과 동일하다.

3.3 Matrix Factorization

Collaborative Filtering의 model-based CF 방식을 활용하는 Matrix Factorization(MF)는 특잇값 분해(SVD)를 활용하여 큰 크기의 sparse matrix를 user matrix, item matrix로 분해한다. 두 행렬의 값을 활용하여 오차를 계산하고 오차를 기반으로 모델을 학습하는 과정을 통해 추천 모델을 개발한다. 오차를 최소화하는 과정에서 SGD(Stochastic Gradient Descent)나 ALS(Alternative Least Square)를 사용한다. 이 보고서는 두가지 방식을 활용하는 모델은 Bayesian Personalized Ranking 모델과 ALS모델을 사용할 것이다.

3.3.1 Bayesian Personalized Ranking model

MF의 첫번째 모델은 Bayesian Personalized Ranking (BPR) 모델을 활용했다. [6] 이 모델은 기존의 Collaborative Filtering의 단점을 개선한 베이지안 추론(maximum a posteriori)에 기반한 optimization 기법인 BPR-Opt방법을 제시한다. 또한 이 방식은 ROC curve의 아래 면적인 AUC(Area Unver Curve)를 최대화 하는 문제와 같다.

BPR은 기존의 방식은 implicit-data에 대해서 사용자가 직접 부정적 피드백을 한 아이템(real negative)과 사용자가 관측하지 않은 데이터(missing value)를 모두 0으로 처리해서 모두 negative feedback으로 간주하여 문제를 해결하였다. 이 방식은 사용자가 이후에 positive feedback을 줄 수도 있는 아이템을 무시하여 아이템 예측에 문제가 발생한다고 문제점을 제시하고 있다. 이를 해결하고자 사용자의 아이템 관측 여부에 따라 상대적인 선호도를 표시하는 방식을 제시한다.

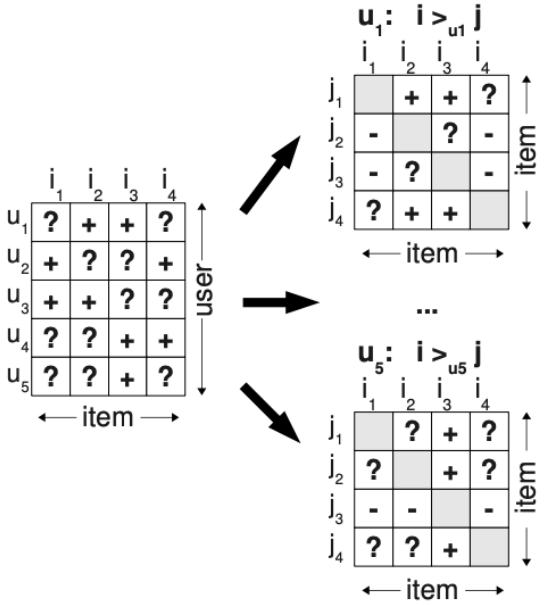

그림 7 BPR에서 제시한 새로운 matrix 표현방법

$i >_u j$ 는 user u는 i를 j보다 선호한다는 표현이다. 논문에서 제시하는 새로운 선호도 표현방법은 다음과 같다:

1. user는 관측된 item은 모든 비관측 item보다 선호한다.
2. 관측된 item들 사이의 선호강도 추론은 불가능하다.
3. 비관측 item들 사이의 선호강도 추론은 불가능하다.

결과적으로 각 user에 대해 아이템의 선호강도를 선호, 비선호, 알 수 없음으로 표현하여 비관측 데이터를 이후 negative feedback으로 판단하는 것을 방지한다.

모델 학습과정을 간단하게 설명하면 user의 아이템 i와 j의 선호관계를 저장하는 데이터셋은 D_s라고 할 때,

Bayesian optimization을 활용하여 사후 확률을 최대화하는 파라미터 Θ를 찾는다. 이를 사후 확률 추정(Maximum A Posteriori Estimation)이라 한다. 이때 파라미터 Θ는 user와 item의 latent vector이다. 최대 사후 확률 추정시 고려하는 정보는 user의 선호 정보인 $>_u$이다. $>_u$는 (u, i, j)에 대해 $i >_u j$와 $j >_u I$ 두 가지 경우밖에 없으므로 베르누이 분포를 따르고 이때 모든 user는 독립임을 가정한다. 이에 베르누이 분포의 likelihood function과 선호 확률인 $p(i >_u j)$는 다음과 같이 정의된다:

$$p(>_u|\Theta)$$
$$= p(i >_u j)^{\delta((u,i,j)\in D_s)}\big(1 - p(i >_u j)\big)^{\delta((u,i,j)\notin D_s)}$$
$$p(i >_u j) \coloneqq \sigma\big(\widehat{x_{uij}}\big), \sigma(x) \coloneqq \frac{1}{1+e^{-x}}$$

위의 식에서 $\widehat{x_{uij}}$ 는 user u의 item i에 대한 점수와 item j에 대한 점수의 차로 정의했다. 파라미터 Θ에 대한 사전 확률은 uniform이나 normal을 사용한다. BPR 논문에서는 normal을 사용했다.

이렇게 구한 파라미터를 모델의 오차를 줄이는 방향으로 학습한다. 학습 과정에서 오차는 SGD방식을 통해 갱신한다.

3.3.2 Alternative Least Square

Alternative Least Square(ALS)는 사용자와 아이템의 Latent Factor를 번갈아가며 학습한다. 기존 방식처럼 두 행렬을 동시에 최적화하는 것은 어렵기 때문에 하나의 행렬을 최적화할 때 다른 행렬을 고정한다. [7] ALS를 사용한 모델에서는 데이터가 존재한다면 1, 존재하지 않는다면 0을 사용하여 sparse matrix를 표현한다. user matrix와 item matrix의 latent factor matrix를 랜덤하게 초기화하고 두 행렬 중 하나를 고정하여 Loss Function을 Convex Function으로 만들어 최적화하는 방식으로 모델을 학습한다.

4. 결과

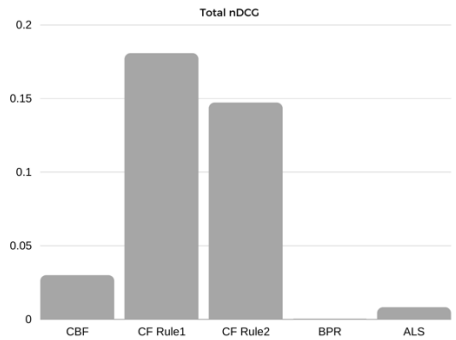

그림 8 노래 + 태그 nDCG

지금까지 총 4가지 알고리즘으로 구현된 추천 모델을 소개하였다. 각 모델들의 validation set score을 비교하면 그림 8과 같은 결과가 나타난다. 가장 좋은 성능을 보이는 것은 겹치는 곡을 가중치로 활용한 Collaborative Filtering인 것을 알 수 있다. 일반적으로 Matrix Factorization의 성능이 좋다고 알려져 있지만 BPR과 ALS의 성능이 CBF보다 떨어지는 것을 볼 수 있다.

카카오 아레나에서 조사한 결과에서 가장 좋은 성능을 보이는 모델은 Autoencoder와 KNN을 활용한 방식이다. [8] 또한 결과에서 보이는 것과 마찬가지로 MF를 활용한 방식은 좋지 못한 성능을 보인다고 밝혔다. 이 보고서에서 활용한 방식은 전통적인 방식의 행렬 기법을 활용하였지만 더 좋은 성능의 모델을 개발하기 위해서는 오토인코더나 Neural Collaborative Filtering같은 신경망을 활용한 딥러닝을 혼합하는 방식을 사용하는 것이 좋아 보인다.

참고문헌

[1] 신현보, “ “음원시장도 유튜브가 먹나” ·s·· 부동의 1위 ‘멜론’이 떨고 있다 [신현보의 딥데이터]”, 2021년 7월 3일 수정, 2021년 12월 10일 접속, https://www.hankyung.com/economy/article/2021070232157.

[2] 카카오 아레나, “Melon Playlist Continuation”, 2021년 12월 10일 접속, https://arena.kakao.com/c/8.

[3] kimhyunwoo, “[EDA] 전체적인 데이터의 형태에 대한 탐색”, 2021 12월 10일 접속, https://arena.kakao.com/forum/topics/228.

[4] Xiangnan Heet al, Neural Collaborative Filtering, IW3C2, 2017.

[5] 카카오 정책산업연구, “멜론에서 음악 추천을 어떻게 할까?”, 2020년 4월 22일 수정, 2021년 12월 10일 접속, https://brunch.co.kr/@kakao-it/342.

[6] Steffen Rendle et al, BPR: Bayesian Personalized Ranking from Implicit Feedback, UAI, 2009.

[7] Yifan Hu et al, Collaborative Filtering for Implicit Feedback Datasets.

[8] 카카오 아레나, “리더보드 스코어 0.07점 달성 이벤트 당첨자 설문조사 결과”, 2020년 7월 30일 수정, 2021년 12월 11일 접속, https://arena.kakao.com/forum/topics/295.