DMA 2021-1

TEAM PROJECT 2 보고서

team 06
김세찬 오재원 이승아 표승희 홍지현

[PART 1] 의사결정나무

0. 의사결정나무(Decision Tree)란?

다른 attribute들의 값이 주어졌을 때, 값이 정해지지 않은 class attribute의 값을 함수로 표현하는 모델을 말한다. 이때, class attribute와 값이 주어진 attribute들로 이루어진 database를 training DB라고 하며, 이를 사용하여 의사결정나무를 만들게 된다.

의사결정나무는 class attribute가 아닌 attribute로부터 질문을 생성하고, 가능한 답들로 표현되며, 각 질문은 node가 된다. node를 만드는 기준은 gini index, entropy와 같은 node의 impurity를 측정하는 기준이며, 각 node들은 impuritiy가 작은 방향으로 만들어진다. 이때, overfitting을 피하기 위해 induction을 조기 종료하는 것이 필요하다. 이는 의사결정나무의 max_depth나 min_samples_leaf등의 조건을 적절히 조절하여 구현할 수 있다.

1. R1-1 & R1-2: 문제 정의 & Training DB 생성

지난 Project 1에서 구현한 사이트 A의 database를 이용하여, 해당 사이트에서 선정한 BEST ITEM의 기준을 찾는 의사결정나무를 만들고자 한다.

이때, training DB는 다음과 같은 column들을 포함하게 만들어야 한다.

분류	세부 정보	null 값 여부
id	아이템의 id	X
ratings	아이템이 사용자들에게 받은 평가 점수	o
num_of_specs	아이템이 가진 스펙의 수	o
num_of_tags	아이템에 붙어 있는 태그의 수	o
num_of_users	아이템을 이용한 이력이 있는 사용자의 수	X
avg_usage_time	아이템을 이용한 사용자들의 인당 평균 이용시간	o
num_of_reviews	아이템에 작성된 리뷰의 수	o
sum_of_recommend	아이템에 작성된 리뷰의 recommend 총합	o
avg_review_len	아이템에 작성된 리뷰의 평균 본문 길이	o
best_item	**아이템의 BEST item 선정 여부**	**X**

<표 1> training DB column

- best_item column은 의사결정나무가 예측해야 하는 class에 해당한다.
- 기존의 item Table에 best_item column을 추가하고, 'best_item_list.txt'의 정보를 사용하여 모든 아이템에 대해 1(txt 파일 포함 item) 또는 0(txt 파일 미포함 item)으로 값을 지정한다.
- null 값 여부는 아이템의 개수가 총 9192개이고, 각각의 data가 포함된 table에서 item_id로 group by한 후 row 개수를 비교하여 확인하였다.

- null 값이 존재하는 경우는 모든 item_id에 대해 해당 column의 값이 존재하지 않는 경우를 말한다.
- best_item_column은 모든 아이템에 대해 값이 정해졌으므로, null값이 없다.
- 해당 column이 존재하는 Table과 item Table을 바로 natural join 시 모든 item_id의 데이터를 보존하지 못할 수 있으므로, left join을 시행한 후 item Table과 natural join을 시행한다.

```
cursor.execute('''
SELECT id, best_item, ratings, num_of_specs, num_of_tags ,num_of_users, avg_usage_time,
num_of_reviews, sum_of_recommend, avg_review_len
FROM item
    NATURAL JOIN (SELECT item.id, COUNT(item_specs.spec_name) AS num_of_specs
        FROM item LEFT JOIN item_specs ON item.id=item_specs.item_id GROUP BY id)a
    NATURAL JOIN (SELECT item.id, COUNT(tag.tag_order) AS num_of_tags
        FROM item LEFT JOIN tag ON item.id=tag.item_id GROUP BY id)b
    NATURAL JOIN (SELECT item.id, COUNT(user_item.user_id) AS num_of_users
        FROM item LEFT JOIN user_item ON item.id=user_item.item_id GROUP BY id)c
    NATURAL JOIN (SELECT item.id, AVG(user_item.usagetime_total AS avg_usage_time
        FROM item LEFT JOIN user_item ON item.id = user_item.item_id GROUP BY id)d
    NATURAL JOIN (SELECT item.id, COUNT(review.id) AS num_of_reviews
        FROM item LEFT JOIN review ON item.id=review.item_id GROUP BY id)e
    NATURAL JOIN (SELECT item.id, SUM(review.recommend) AS sum_of_recommend
        FROM item LEFT JOIN review ON item.id=review.item_id GROUP BY id)f
    NATURAL JOIN (SELECT item.id, AVG(review.body AS avg_review_len
        FROM item LEFT JOIN review ON item.id=review.item_id GROUP BY id)g
''')
```

<코드 1> training DB 생성을 위한 Nested query

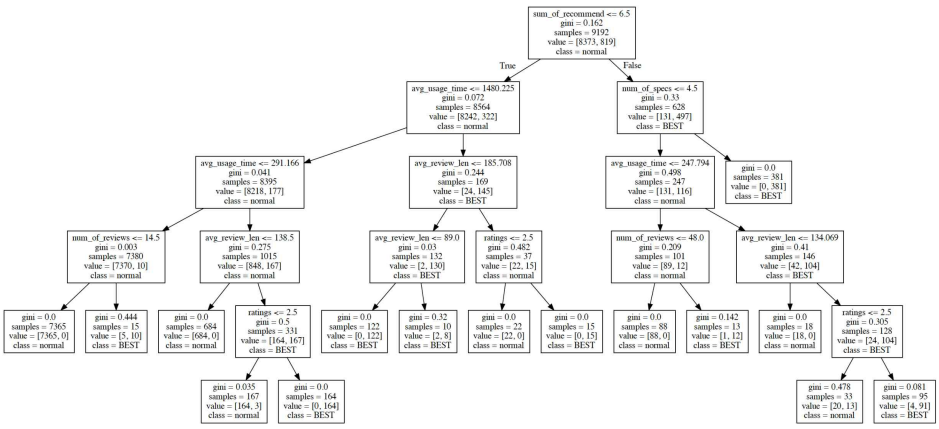
2. R1-3: Decision Tree 생성

조건1. feature names: ratings, num_of_specs, num_of_tags, num_of_users, avg_usage_time, num_of_reviews,sum_of_recommend, avg_review_len

조건2. min_samples_leaf: 10

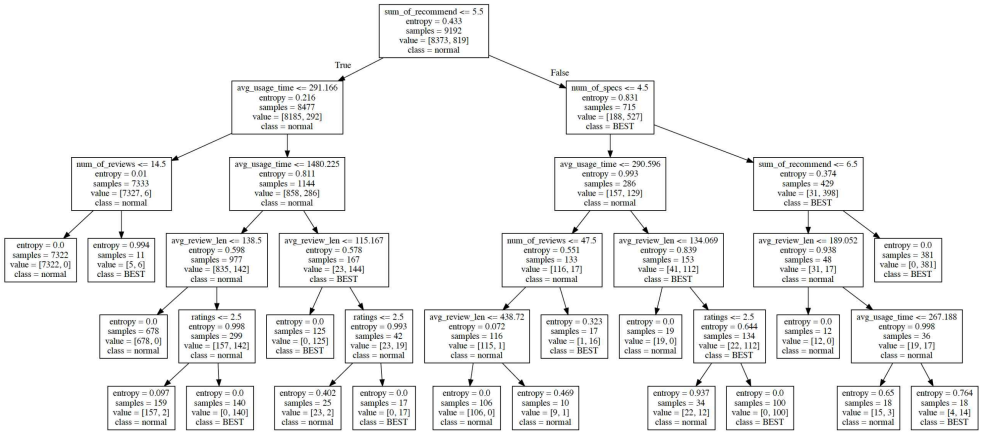
조건3. max_depth: 5

① 위 조건 3개 + 조건 4 (Node impurity criterion: gini) 만족하는 DT



<그림 1> Node impurity를 gini로 측정한 의사결정나무

② 위 조건 3개 + 조건 4 (Node impurity criterion: entropy) 만족하는DT



<그림 2> Node impurity를 entropy로 측정한 의사결정나무

3. R1-4 : input attribute, max_depth, min_samples_leaf에 따른 DT 실험 결과

1) 실험방법

R1-3에서 사용했던 feature_list, max_depth, min_samples_leaf의 값을 변경해가면서 mean_node impurity가 가장 작은 값을 찾아보았다.

Features의 경우, 'ratings', 'num_of_specs', 'num_of_tags', 'num_of_users', 'avg_usage_time', 'num_of_reviews', 'sum_of_recommend', 'avg_review_len'로 총 8가지였고, 각각의 feature를 선택하는 여부에 따른 총 255가지의 경우의 수를 고려하였다(모든 feature가 들어간 경우는 R1-3에서 이미 확인해보았기 때문에 제외하였다).

max_depth는 1, 2, 3, 4, 5, 6, 7 총 7가지의 경우의 수를, min_samples_leaf는 1, 5, 10 총 3가지의 경우의 수를 고려하여 5354 (255*7*3-1 = 5354, 모든 input 값이 들어가지 않는 경우는 제외)번의 실험을 진행하였다.

gini, entropy를 최소화하는 과정에서 leaf node별로 sample수가 다른데, 이들의 gini와 entropy를 각각 가중치 평균을 내주어서 mean_gini와 mean_entropy를 계산하였다.

```
# gini

DT_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=max_depth, min_samples_leaf=min_samples_leaf)
DT_gini.fit(X=features, y=classes)
graph_gini = tree.export_graphviz(DT_gini,
                                  out_file=None,
                                  feature_names=feature_names,
                                  class_names=['normal','BEST'])
sum_gini = 0
total_samples = 0
for line in graph_gini.split('\n'):
    if 'gini' in line:
        if line.split('gini')[0][-3:] == '\\n':
            continue
        else:
            gini = float(line.split('gini')[1].split('\\n')[0][3:])
            samples = int(line.split('gini')[1].split('\\n')[1].split(' = ')[1])
            sum_gini += gini * samples
            total_samples += samples
mean_gini = sum_gini / total_samples
```

<코드 2> mean_gini 구하는 코드

```
# enthropy

DT_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=max_depth, min_samples_leaf=min_samples_leaf)
DT_entropy.fit(X=features, y=classes)
graph_entropy = tree.export_graphviz(DT_entropy,
                                     out_file=None,
                                     feature_names=feature_names,
                                     class_names=['normal','BEST'])

sum_entropy = 0
total_samples = 0
for line in graph_entropy.split('\n'):
    if 'entropy' in line:
        if line.split('entropy')[0][-3:] == '\\n':
            continue
        else:
            entropy = float(line.split('entropy')[1].split('\\n')[0][3:])
            samples = int(line.split('entropy')[1].split('\\n')[1].split(' = ')[1])
            sum_entropy += entropy * samples
            total_samples += samples
mean_entropy = sum_entropy / total_samples
```

<코드 3> mean_entropy 구하는 코드

2) 실험결과 및 고찰 - mean_node impurity가 최소가 되는 DT 생성

(1) 실험결과 1 - gini

5354개의 실험 결과를 df에 저장하여 'mean_gini'를 기준으로 내림차순으로 정렬을 하여, mean_node impurity가 최소가 되는 값을 찾았다.(df.sort_values(by='mean_gini')

Features에는 'ratings', 'num_of_specs', 'avg_usage_time', 'sum_of_recommend', 'avg_review_len'만 포함시키고, max_depth는 7, min_sample_leaf는 1일 때, mean_gini가 0.051856으로 mean_node impurity 가장 작았다.

```
df.sort_values(by='mean_gini')
```

	ratings	num_of_specs	num_of_tags	num_of_users	avg_usage_time	num_of_reviews	sum_of_recommend	avg_review_len	max_depth	min_samples_leaf	mean_gini	mean_entropy
4260	O	O	X	X	O	X	O	O	7	1	0.051856	0.234295
4261	O	O	X	X	O	X	O	O	7	5	0.052045	0.235752
4262	O	O	X	X	O	X	O	O	7	10	0.052099	0.236188
4239	O	O	X	X	O	X	O	X	7	1	0.056847	0.183421
4240	O	O	X	X	O	X	O	X	7	5	0.057031	0.183670
...	...	...	...	...	...	...	...	...	...	...	...	...
1343	X	O	X	X	X	X	X	X	7	10	0.166306	0.429674
1339	X	O	X	X	X	X	X	X	6	5	0.166341	0.428891
1342	X	O	X	X	X	X	X	X	7	5	0.166341	0.429727
1338	X	O	X	X	X	X	X	X	6	1	0.167144	0.428709
1341	X	O	X	X	X	X	X	X	7	1	0.167607	0.429518

5355 rows × 12 columns

<그림 3> mean_gini로 내림차순 정렬한 data frame

(2) 실험결과 2 - entropy

5354개의 실험 결과를 df에 저장하여 'mean_entropy'를 기준으로 내림차순으로 정렬을 하여, mean_nodeimpurity가 최소가 되는 값을 찾았다. (df.sort_values(by='mean_entropy'))

Features에는 'ratings', 'num_of_specs', 'avg_usage_time', 'sum_of_recommend'만 포함시키고, max_depth는 6, min_sample_leaf는 1일 때, mean_entropy가 0.180551로 mean_node impurity 가장 작았다.

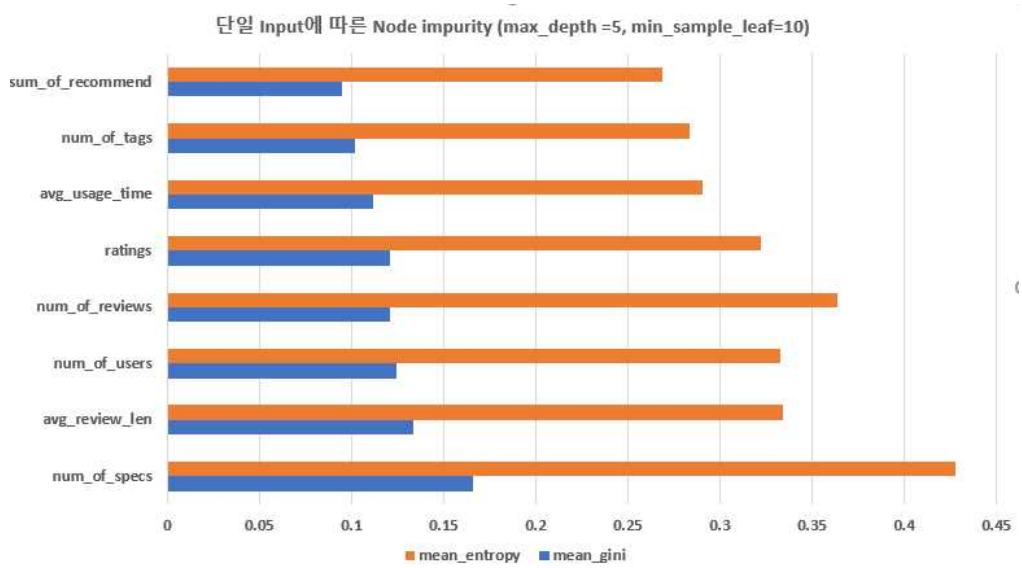
```
df.sort_values(by='mean_entropy')
```

	ratings	num_of_specs	num_of_tags	num_of_users	avg_usage_time	num_of_reviews	sum_of_recommend	avg_review_len	max_depth	min_samples_leaf	mean_gini	mean_entropy
4236	O	O	X	X	O	X	O	X	6	1	0.061852	0.180551
4237	O	O	X	X	O	X	O	X	6	5	0.062026	0.181058
4238	O	O	X	X	O	X	O	X	6	10	0.062372	0.182171
4233	O	O	X	X	O	X	O	X	5	1	0.068732	0.183065
4234	O	O	X	X	O	X	O	X	5	5	0.068926	0.183181
...	...	...	...	...	...	...	...	...	...	...	...	...
1340	X	O	X	X	X	X	X	X	6	10	0.166306	0.428838
1339	X	O	X	X	X	X	X	X	6	5	0.166341	0.428891
1341	X	O	X	X	X	X	X	X	7	1	0.167607	0.429518
1343	X	O	X	X	X	X	X	X	7	10	0.166306	0.429674
1342	X	O	X	X	X	X	X	X	7	5	0.166341	0.429727

5355 rows × 12 columns

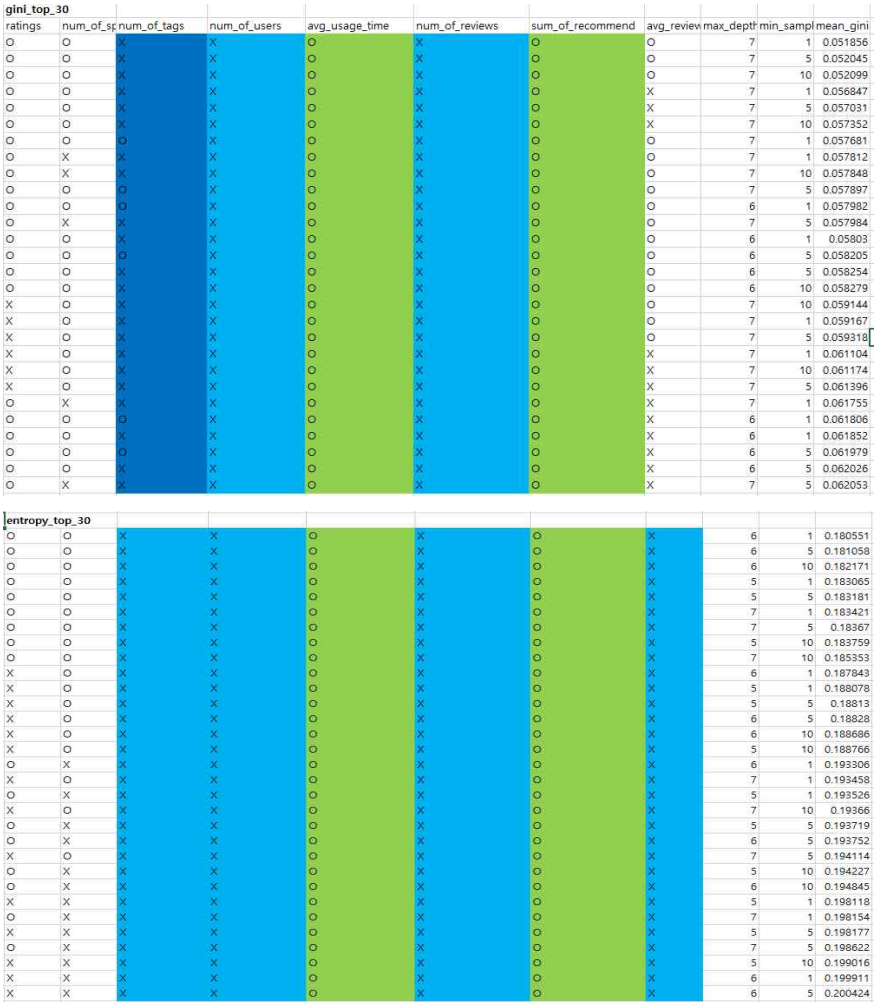
<그림 4> mean_entropy로 내림차순 정렬한 data frame

(3) 단일 input에 따른 mean_node impurity 측정 및 상관관계에 대한 정성적 분석



<그림 5> input에 따른 mean_node impurity 측정 및 상관관계 분석 결과

위의 그래프는 주어진 input들을 하나씩만 사용하여 DT를 생성하고, node_impurity를 측정했을 때의 결과를 나타냈다. node_impurity가 작을수록 해당 input만을 사용했을 때 class를 잘 구분한다고 볼 수 있으며, sum_of_recommend가 단일 input으로써 가장 class를 잘 구분하고 num_of_specs는 가장 class를 잘 구분하지 못하는 것으로 볼 수 있다.



gini_top_30

ratings	num_of_sp	num_of_tags	num_of_users	avg_usage_time	num_of_reviews	sum_of_recommend	avg_review	max_depth	min_sampl	mean_gini
O	O	X	X	O	X	O	O	7	1	0.051856
O	O	X	X	O	X	O	O	7	5	0.052045
O	O	X	X	O	X	O	O	7	10	0.052099
O	O	X	X	O	X	O	X	7	1	0.056847
O	O	X	X	O	X	O	X	7	5	0.057031
O	O	X	X	O	X	O	X	7	10	0.057352
O	O	O	X	O	X	O	O	7	1	0.057681
O	X	X	X	O	X	O	O	7	1	0.057812
O	X	X	X	O	X	O	O	7	10	0.057848
O	O	O	X	O	X	O	O	7	5	0.057897
O	O	O	X	O	X	O	O	6	1	0.057982
O	X	X	X	O	X	O	O	7	5	0.057984
O	O	X	X	O	X	O	O	6	1	0.05803
O	O	O	X	O	X	O	O	6	5	0.058205
O	O	X	X	O	X	O	O	6	5	0.058254
O	O	X	X	O	X	O	O	6	10	0.058279
X	O	X	X	O	X	O	O	7	10	0.059144
X	O	X	X	O	X	O	O	7	1	0.059167
X	O	X	X	O	X	O	O	7	5	0.059318
X	O	X	X	O	X	O	X	7	1	0.061104
X	O	X	X	O	X	O	X	7	10	0.061174
X	O	X	X	O	X	O	X	7	5	0.061396
O	X	X	X	O	X	O	X	7	1	0.061755
O	O	O	X	O	X	O	X	6	1	0.061806
O	O	X	X	O	X	O	X	6	1	0.061852
O	O	O	X	O	X	O	X	6	5	0.061979
O	O	X	X	O	X	O	X	6	5	0.062026
O	X	X	X	O	X	O	X	7	5	0.062053

entropy_top_30

O	O	X	X	O	X	O	X	6	1	0.180551
O	O	X	X	O	X	O	X	6	5	0.181058
O	O	X	X	O	X	O	X	6	10	0.182171
O	O	X	X	O	X	O	X	5	1	0.183065
O	O	X	X	O	X	O	X	5	5	0.183181
O	O	X	X	O	X	O	X	7	1	0.183421
O	O	X	X	O	X	O	X	7	5	0.18367
O	O	X	X	O	X	O	X	5	10	0.183759
O	O	X	X	O	X	O	X	7	10	0.185353
X	O	X	X	O	X	O	X	6	1	0.187843
X	O	X	X	O	X	O	X	5	1	0.188078
X	O	X	X	O	X	O	X	5	5	0.18813
X	O	X	X	O	X	O	X	6	5	0.18828
X	O	X	X	O	X	O	X	6	10	0.188686
X	O	X	X	O	X	O	X	5	10	0.188766
O	X	X	X	O	X	O	X	6	1	0.193306
X	O	X	X	O	X	O	X	7	1	0.193458
O	X	X	X	O	X	O	X	5	1	0.193526
X	O	X	X	O	X	O	X	7	10	0.19366
O	X	X	X	O	X	O	X	5	5	0.193719
O	X	X	X	O	X	O	X	6	5	0.193752
X	O	X	X	O	X	O	X	7	5	0.194114
O	X	X	X	O	X	O	X	5	10	0.194227
O	X	X	X	O	X	O	X	6	10	0.194845
X	X	X	X	O	X	O	X	5	1	0.198118
O	X	X	X	O	X	O	X	7	1	0.198154
X	X	X	X	O	X	O	X	5	5	0.198177
O	X	X	X	O	X	O	X	7	5	0.198622
X	X	X	X	O	X	O	X	5	10	0.199016
X	X	X	X	O	X	O	X	6	1	0.199911
X	X	X	X	O	X	O	X	6	5	0.200424

<그림 6> gini index와 entropy를 기준으로 했을 때, mean node_purity top 30

그러나 앞서 (1)의 결과에서처럼 단일 input의 node_impurity가 작은 feature들만 사용한다고 해서 node_impurtiy가 작아지는 것이 아님을 위의 두 표로부터 알 수 있다. 각 표는 gini index와 entropy를 기준으로 node_impurtiy를 측정하여 작은 impurity를 30개씩 나열한 것이다. 이때, 초록색으로 칠해진 input feature(sum_of_recommend, avg_usage_time)는 top 30개의 결과에서 모두 사용된 input feature이며, 파란색으로 칠해진 input feature들은 사용되지 않은 input feature들이다.

이는 각 input 간의 상관관계가 있다고 볼 수 있는데, sum_of_recommend를 기준으로 다른 input feature들을 하나씩 추가했을 때의 node_impurity를 측정하면, 아래의 표와 같다.

ratings	num_of_specs	num_of_tags	num_of_users	avg_usage_time	num_of_reviews	sum_of_recommend	avg_review_len	mean_gini	mean_entr	label(등수)
X	X	X	X	O	X	O	X	0.075594	0.199016	30
X	O	X	X	X	X	O	X	0.092602	0.262056	1379
O	X	X	X	X	X	O	X	0.096439	0.266511	1758
X	X	X	O	X	X	O	X	0.096592	0.267733	1905
X	X	X	X	X	X	O	O	0.095645	0.268085	1920
X	X	X	X	X	X	O	X	0.094813	0.268642	1938
X	X	O	X	X	X	O	X	0.096551	0.27008	2016
X	X	X	X	X	O	O	X	0.09697	0.271794	2089

<그림 7> sum_of_recommend와 다른 input feature를 추가하여 DT 생성 시 impurity 측정

이로부터, 각 input feature의 조합에 따라 node impurity의 값이 더 감소하거나 증가할 수 있음을 알 수 있으며, 본 데이터에서는 sum_of_recommend와 avg_usage_time만 input feature로 사용하였을 때 가장 node_impurity가 작게 하는 feature임을 확인할 수 있다.

[PART 2]

0. 문제 정의

본 part 2 은 사이트 A의 bundle 간의 연관 분석을 목표로 한다.

1. R2-1: bundle_score view 생성

포함된 아이템의 수, 관련된 장르의 수, 포함된 아이템을 이용한 사용자의 수가 많은 상위 30개의 bundle로 bundle_score라는 이름을 가진 view를 생성한다.

```
CREATE VIEW bundle_score AS
    SELECT
        a.bundle_id,
        bundle_name,
        num_item,
        IFNULL(num_genre, 0) AS num_genre,
        num_user,
        (num_item + IFNULL(num_genre, 0) + num_user) AS score
    FROM
1   (SELECT
        id AS bundle_id, bundle_name
    FROM
        bundle) a
        NATURAL JOIN
2   (SELECT
        bundle_id, COUNT(item_id) * 100 AS num_item
    FROM
        bundle_item
    GROUP BY bundle_id) b
        NATURAL JOIN
3   (SELECT
        bundle_id, (COUNT(user_id) / COUNT(DISTINCT d.item_id)) AS num_user
    FROM
        bundle_item d, user_item e
    WHERE
        d.item_id = e.item_id
    GROUP BY bundle_id) c
        LEFT JOIN
4   (SELECT
        bundle_id, COUNT(DISTINCT genre_id) * 100 AS num_genre
    FROM
        bundle_genre
    GROUP BY bundle_id) f ON a.bundle_id = f.bundle_id
    ORDER BY score DESC
    LIMIT 30
```

<코드 4> bundle_score view 생성을 위한 코드

bundle에 bundle_item, user_item, bundle_genre을 join하여 bundle_score view를 생성하였다.

① bundle에서 bundle_id, bundle_name column을 추출하여 table a로 명명하였다.

② bundle_item에서 bundle_id, num_item(해당 번들에 포함된 item의 수*100)을 column으로 추출하여 table b로 명명하였다.

③ bundle_item과 user_item을 item_id를 기준으로 join시켜 bundle_id, num_user(해당 번들에 포함된 아이템들의 사용자 수 평균)을 추출하여 table c로 명명하였다.

④ bundle_genre에서 bundle_id, num_genre(해당 번들과 관련된 장르의 수)를 추출하여 table f로 명명하였다.

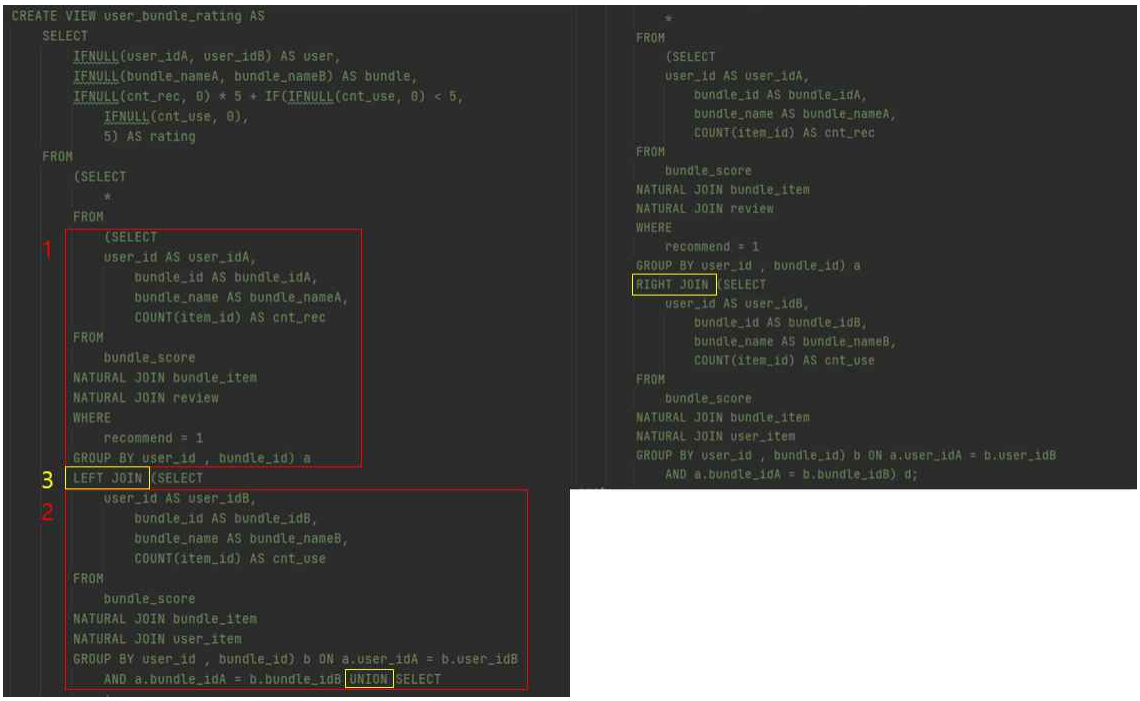
table a, b, c를 natural join으로 연결하였고, 이에 table f는 left join으로 연결하였다. 번들의 총 개수는 515개이고, table f에 포함된 번들은 500개이기 때문에 정보가 없는 15개의 번들이 누락 되는 것을 방지하기 위하여 table f를 left join으로 연결하였다. join된 table에서 bundle_id, bundle_name, num_item, num_genre, num_user, score를 추출하여 score로 내림차순 정리하였고 상위 30개만 bundle_score로 생성하였다. 이때, ifnull() 함수를 이용하여 num_genre의 null값을 0으로 처리해주었다. bundle_score는 <표 2-1>과 같다.

bundle_id	bundle_name	num_item	num_genre	num_user	score
236	Counter-Strike Complete	200	100	33641.5	33941.5
233	Left 4 Dead Bundle	200	100	21594.5	21894.5
234	Portal Bundle	200	200	21388.5	21788.5
232	Valve Complete Pack	1600	200	14089.25	15889.25
312	Dont Starve MEGA PACK	100	300	15385	15785
575	Sid Meiers Civilization V: Complete	100	100	15119	15319
240	Source Multiplayer Pack	300	100	14854.33	15254.33
612	Killing Floor 1 Complete Your Set!	100	100	14046	14246
257	Borderlands Triple Pack	300	200	11836.67	12336.67
1100	New World Collection	100	300	11227	11627
572	BioShock Triple Pack	300	200	10305.33	10805.33
231	Half-Life Complete	600	100	9550.5	10250.5
712	Borderlands Take Over Your Life Bundle	400	300	9437.25	10137.25
374	Goat Simulator: GOATY	100	300	9337	9737
250	Awesomenauts - Mega Value Pack	100	400	8603	9103
166	Grand Theft Auto V & Great White Shark Cash Card	100	200	8532	8832
265	Grand Theft Auto V & Megalodon Shark Cash Card	100	200	8532	8832
264	Grand Theft Auto V & Whale Shark Cash Card	100	200	8532	8832
727	The Witcher Trilogy	200	200	7763.5	8163.5
262	Guns of Icarus Online + Alliance Collectors Edition Bundle	100	400	7039	7539

574	Sid Meiers Civilization Anthology	300	100	6831.333	7231.333
307	World of Magicka Bundle	200	300	6675	7175
237	Half-Life 1 Anthology	400	100	6570.75	7070.75
362	Eidos Anthology	3500	500	2898.2	6898.2
435	SpeedRunners Deluxe Pack	100	500	5281	5881
139	The Binding of Isaac : Rebirth + Afterbirth Bundle	100	100	5451	5651
616	Tripwire Complete Bundle	500	500	4511	5511
983	Supergiant Collection	200	300	4710.5	5210.5
1243	Bad Company 2 Bundle	100	100	4796	4996
383	Telltale Collection	2900	400	1434.828	4734.828

<표 2> bundle_score view

2. R2-2: user_bundle_rating view 및 partial_user_bundle_rating view 생성
각 user가 bundle에 대해 가지는 관심 정도를 rating으로 정의한다.
(1) user_bundle_rating view 생성



<코드 5> user_bundle_rating view 생성을 위한 코드

① bundle_score에 bundle_item과 review를 recommend=1 조건을 주어 natural join 하여 user_id, bundle_id, bundle_name, cnt_rec column을 추출하여 table a로 명명하였다.

② bundle_score에 bundle_item과 user_item을 natural join하여 bundle_id,

bundle_name, cnt_use column을 추출하여 table b로 명명하였다.

③ table a와 table b를 손실되는 데이터가 없도록 Full outer join을 하였다. left join, union, right join으로 full outer join을 구현하였다. full outer join의 결과를 table d로 명명하였다.

table d에서 ifnull() 함수를 이용하여 null값을 처리하였고, user, bundle, rating column을 추출하여 user_bundle_rating view를 생성하였다.

(2) partial_user_bundle_rating view 생성

```
CREATE VIEW partial_user_bundle_rating AS
SELECT
    *
FROM
    user_bundle_rating
        NATURAL JOIN
    (SELECT
        user
    FROM
        user_bundle_rating
    GROUP BY user
    HAVING COUNT(*) >= 20) a;
```

<코드 6> partial_user_bundle_rating view 생성을 위한 코드

user_bundle_rating에서 rating 개수가 20개 이상인 user를 추출하여 partial_user_bundle_rating으로 생성하였다. partial_user_bundle의 일부는 <표 2-2>와 같다.

user	bundle	rating
8062369021	Counter-Strike Complete	7
7995125763	Counter-Strike Complete	7
7995472865	Counter-Strike Complete	7
8026487694	Counter-Strike Complete	7
8048491617	Counter-Strike Complete	7
8056199072	Counter-Strike Complete	7
8063432096	Counter-Strike Complete	7
8040597024	Counter-Strike Complete	7
8018495407	Counter-Strike Complete	7
8053196362	Counter-Strike Complete	12
8016078117	Counter-Strike Complete	12
7973612806	Counter-Strike Complete	7
8027368512	Counter-Strike Complete	12

<표 3> partial_user_bundle의 일부

3. R2-3: horizontal table 생성

연관분석을 위해 R2-2의 partial_user_bundle_rating을 horizontal table로 만든다.

- partial_user_bundle_rating을 pandas의 DataFrame으로 저장하고, DataFrame의 column 명은 partial_user_bundle_rating의 column명과 일치하도록 변경하고, 'user'를 index로 사용한다.
- rating column의 값을 제값으로 나누어, 기존에 rating 정보가 존재할 경우 1의 값을 갖도록 설정했다.
- 이를 horizontal view로 바꾸기 위해, pandas의 pivot_table 함수를 사용했다. index는 user, column은 bundle, value는 rating으로 두었고, fill_value=0으로 설정하여 null값이 생길 경우 0으로 대체하였다.
- 위의 과정을 통해 연관분석에 적절한 DataFrame을 생성하였다. DataFrame에서 user는 transaction의 역할을, bundle은 item의 역할을 한다.

```
cursor.execute('SELECT * FROM partial_user_bundle_rating')
df = pd.DataFrame(cursor.fetchall())
df.columns = cursor.column_names
df = df.set_index('user')

df['rating'] = df['rating'] / df['rating']
hor_view = pd.pivot_table(df, index='user', columns='bundle', values='rating', fill_value=0)

filename = 'DMA_project2_team06_part2_horizontal.pkl'
hor_view.to_pickle(filename)
```

<코드 7> horizontal table 생성

4. R2-4: 연관분석

R2-3의 dataframe을 사용하여 연관분석을 수행한다.

(1) frequent itemset 및 association rule 생성

- apriori 함수를 사용하여 frequent itemset을 생성한다. 최소 support를 0.35로 설정하여 전체 transaction에서 35% 이상 등장하는 itemset만을 frequent itemset로 설정했다.
- association_rules 함수를 사용하여 association rule을 생성한다. metric은 lift로 두고, 최소 lift는 2로 설정했다.

```
frequent_itemsets = apriori(hor_view, min_support = 0.35, use_colnames = True)
rules = association_rules(frequent_itemsets, metric = 'lift', min_threshold = 2)
```

<코드 8> training DB 생성을 위한 Nested query

(2) 연관분석 결과

- (1)의 과정을 통해 총 199369개의 association rule이 생성되었고, 각 rule에 대해 아래의 표와 같이 총 7가지 척도를 확인할 수 있다.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	frozenset(X)	frozenset(Y)	0.458046	0.460701	0.433617	0.946667	2.05484	0.222594	10.11186

<표 4> association rule 예시

- association rule을 (X->Y) 라고 할 때, 각 column의 설명은 아래와 같다.

antecedents	itemset X
consequents	itemset Y
antecedent support	전체 transaction 중 X를 포함하는 비율
consequent support	전체 transaction 중 Y를 포함하는 비율
support	전체 transaction 중 X,Y를 모두 포함하는 비율
confidence	X를 포함하는 transaction 중 Y를 포함하는 비율 (support / antecedent support)
lift	confidence와 consequenct support의 비율 (confidence / consequent support or support / antecedent support * consequent support)
leverage	support - antecedent support * consequent support
conviction	(1 - consequenct support) / (1-confidence)

<표 5> association rule 평가 척도

- support로 전체 transaction에서 association rule이 등장하는 정도를, confidence로 두 itemset 간의 연관성을, lift로 두 itemset 간의 상관관계를 알아볼 수 있다. leverage, conviction은 support, confidence, lift와 관련 있는 척도이므로, support, confidence, lift만을 고려하였다.
- 현재 association rule은 (1)에서 support를 0.35 이상으로 설정하여 transaction에서 드물게 발생하는 rule을 제외하고, lift를 2 이상으로 설정하여 양의 상관관계를 갖는 rule만을 선택한 결과이다. 또한 support는 (0.350306, 0.456977) 사이의 값을, confidence는 (0.747079,1) 사이의 값을, lift는 (2.115319, 2.313879) 사이의 값을 갖는다. lift의 값은 rule 간에 큰 차이를 보이지 않았고, 따라서 support와 confidence의 비교를 통해 rule의 중요도 및 유용성을 평가해볼 수 있다.
- 예를 들어, 현재 association rule에 support 0.456977, confidence 1을 갖는 rule이 다수 존재한다. 아래의 표는 그 중 일부 rule의 antecedents와 consequents를 나타낸다.

	antecedents	consequents
0	'Sid Meiers Civilization V: Complete', 'Grand Theft Auto V & Great White Shark Cash Card'	'Grand Theft Auto V & Whale Shark Cash Card', 'Sid Meiers Civilization Anthology', 'Grand Theft Auto V & Megalodon Shark Cash Card'
1	'Grand Theft Auto V & Whale Shark Cash Card', 'Sid Meiers Civilization V: Complete', 'Grand Theft Auto V & Great White Shark Cash Card'	'Grand Theft Auto V & Whale Shark Cash Card', 'Sid Meiers Civilization Anthology', 'Grand Theft Auto V & Megalodon Shark Cash Card', 'Valve Complete Pack'
2	'Grand Theft Auto V & Whale Shark Cash Card', 'Sid Meiers Civilization V: Complete', 'Valve Complete Pack'	'Sid Meiers Civilization Anthology', 'Grand Theft Auto V & Great White Shark Cash Card'

<표 6> 최대 support, confidence를 갖는 rule의 일부

- 위의 rule은 최대 support, confidence를 갖는다. 따라서 두 itemset 간의 연관성이 매우 높고 양의 상관관계를 가지며, 전체 transaction에서의 발생 빈도 또한 높으므로 유용하고 영향력이 큰 rule이라고 평가할 수 있다. 실제로 최대 support, confidence를 갖는 rule들을 살펴보면 itemset(Sid Meiers Civilization series, Grand Theft Auto V series, Valve Complete Pack)의 관계를 나타내는 rule이 많고, 따라서 해당 item간의 연관성이 높을 것이라 생각해볼 수 있다.
- 이와 같은 방법으로 관련 있는 itemset을 도출해낼 수 있고, 이 외에도 상황에 따라 중요시하는 척도를 우선순위로 두고 적절한 rule을 찾아보거나, 혹은 관심 있는 item이 있다면 해당 item과 관련된 rule을 찾아봄으로써 연관분석을 활용할 수 있다.

[PART 3]

0. 문제 정의

본 part 3 는 user에게 bundle을 추천하는 추천시스템 구현을 목적으로 한다.

각 세부 part에 대한 requirements 는 아래 <표 7>과 같다.

PART	Requirements
R3-1	- 점수 예측 결과 중 top-n개를 반환하는 get_top_n 함수 작성
R3-2	- 주어진 users에 대한 top-5 bundle 추출 <알고리즘> ① KNNBasic (유사도 : cosine) ② KNNWithMeans (유사도 : pearson) - User-based recommendation에서 여러 알고리즘과 유사도 적용 후, cross validation (k=5, random_state=0)을 기준으로 가장 좋은 성능의 모델 선정
R3-3	- 주어진 bundles에 대한 top-10 user 추출 <알고리즘> ① KNNBasic (유사도 : cosine) ② KNNWithMeans (유사도 : pearson) - Item-based recommendation에서 여러 알고리즘과 유사도 적용 후, cross validation (k=5, random_state=0)을 기준으로 가장 좋은 성능의 모델 선정
R3-4	- 주어진 users에 대한 top-5 bundle 추출 <알고리즘> ① SVD (n_factors=100, n_epoch=50, biased=False) ② SVD (n_factors=200, n_epoch=100, biased=True) ③ SVD++ (n_factors=100, n_epoch=50) ④ SVD++ (n_factors=100, n_epoch=100) - Matrix-based recommendation에서 여러 알고리즘과 유사도 적용 후, cross validation (k=5, random_state=0)을 기준으로 가장 좋은 성능의 모델 선정

<표 7>

주어진 user와 bundle자료는 <표 8>와 같다.

PART	자료	비고
3-2	'8051826169', '8027368512', '7998746368', '8054453794', '8030770479'	5개
3-3	'World of Magicka Bundle', 'Borderlands Triple Pack', 'Tripwire Complete Bundle', 'Grand Theft Auto V & White Shark Cash Card', 'Killing Floor 1 Complete Your Set!'	5개
3-4	'8051826169', '8027368512', '7998746368', '8054453794', '8030770479'	5개

<표 8>

1. R3-1: get-top-n 함수 작성

- user_based=True인 경우와 user_based=False 인 경우에 대해 get-top-n 함수를 정의하였다.

- user_based=True인 경우의 코드는 아래 <코드 9>와 같다.

```
# PART 3:
# TODO: Requirement 3-1. WRITE get_top_n
def get_top_n(algo, testset, id_list, n, user_based=True):
    results = defaultdict(list)

    if user_based:
        # TODO: testset의 데이터 중에 user id가 id_list 안에 있는 데이터만 따로 testset_id로 저장
        # Hint: testset은 (user_id, bundle_name, default_rating)의 tuple을 요소로 갖는 list

        testset_id = []
        print("makes a testset_id list...")
        for i in testset:
            if i[0] in id_list:
                testset_id.append(i)
            else:
                pass

        predictions = algo.test(testset_id)
        print("make a prediction results...")

        # TODO: results는 user_id를 key로, [(bundle_name, estimated_rating)의 tuple이 모인 list]를 value로 갖는 dictionary
        for uid, bname, true_r, est, _ in predictions:
            results[uid].append((bname, est))
```

<코드 9> get-top-n 함수 중 user_based=True인 경우의 코드

- ① testset 의 데이터 중 user_id가 주어진 id_list에 존재하면 testset_id 리스트에 삽입하였다.
- ② testset_id 리스트에 대해 예측을 진행하였다.
- ③ user_id를 key로 하고, 그에 대한 (bundle_name, estimated_rating)들의 리스트를 value로 갖는 dictionary를 생성하였다.
- user_based=False 인 경우, ① user_id가 아닌 bundle_name의 존재 여부를 따졌다.
- user_based=False 인 경우, ③ bundle_name을 key로 하고, 그에 대한 (user_id, estimated_rating) 들의 리스트를 value로 갖는 dictionary를 생성하였다.
- 예측 결과인 result 딕셔너리에서 rating이 높은 순서대로 정렬하고(reverse=True) 상위 n개를 남겼다.

2. R3-2: user_based recommendation 진행

(1) 알고리즘 3-2-①: KNNBasic (유사도 : cosine)

- algorithm의 정의는 <코드 10>과 같다.

```
sim_options = {'name': 'cosine', 'user_based': True}
algo = surprise.prediction_algorithms.knns.KNNBasic(sim_options=sim_options)
```

<코드 10> 알고리즘 3-2-①

- 본 알고리즘을 따라 도출된 결과값의 일부는 아래 <그림 8>과 같다.

```
User ID 7998746368 top-5 results
Bundle NAME Sid Meiers Civilization Anthology
        score 1.7021921479492306
Bundle NAME Grand Theft Auto V & Megalodon Shark Cash Card
        score 1.5008325540188168
Bundle NAME Grand Theft Auto V & Whale Shark Cash Card
        score 1.5008325540188168
Bundle NAME Grand Theft Auto V & Great White Shark Cash Card
        score 1.5008325540188168
Bundle NAME The Binding of Isaac : Rebirth + Afterbirth Bundle
        score 1.4934661745854763
```

<그림 8> 알고리즘 3-2-① 결과값 中 user_id '7998746368'에 대한 결과값

(2) 알고리즘 3-2-②: KNNWithMeans (유사도 : pearson)

- algorithm의 정의는 <코드 11>과 같다.

```
sim_options = {'name': 'pearson', 'user_based': True}
algo = surprise.prediction_algorithms.knns.KNNWithMeans(sim_options=sim_options)
```

<코드 11> 알고리즘 3-2-②

- 본 알고리즘을 따라 도출된 결과값의 형식은 <그림 8>을 따른다.

(3) User_based 알고리즘 실험 후 cross validation 시행

- KFold의 n_split=5, random_state=0으로 설정하였다.
- 실험한 알고리즘은 KNNBasic, KNNWithMeans, KNNBaseline으로 총 3가지이다.
- KNNBasic과 KNNWithMeans의 경우 각 4가지의 유사도를 변경해가며 실험을 진행하였다.
- KNNBaseline의 경우 유사도는 'pearson_baseline'으로 고정하고 'method'로는 'als'를 사용하였으며, (n_epochs, reg_u, reg_i)를 3가지로 변경해가며 실험을 진행하였다. (KNNBaseline 알고리즘은 대개 pearson_baseline 유사도를 이용하기 때문에 고정하였다.)
- Cross validation은 RMSE값을 기준으로 설정하였다.
- 각 실험당 5개 fold의 RMSE는 acc 리스트에 담기도록 했으며, acc 리스트의 평균값을 각 알고리즘의 mean_acc_list에 삽입하였다. mean_acc_list는 <표 9>와 같다.

List Name	내용
KBs_mean_acc_list	알고리즘 KNNBasic에 대한 acc 리스트 평균값
KW_mean_acc_list	알고리즘 KNNWithMeans에 대한 acc 리스트 평균값
KBl_mean_acc_list	알고리즘 KNNBaseline에 대한 acc 리스트 평균값

<표 9>

- KNNBasic Algorithm을 이용한 실험의 코드는 <코드 12>와 같다.
- 다른 알고리즘에 대한 cross validation도 <코드 12> 형식에 맞춰 코딩되었다.
- User_based=True로 설정하였다.

```
print("Cross validation...")
np.random.seed(0)
kf = surprise.model_selection.split.KFold(n_splits=5, random_state=0)

KBs_sim_list = ['cosine', 'pearson', 'MSD', 'pearson_baseline']
KBs_mean_acc_list = []

for x in KBs_sim_list:
    acc = []
    sim_options = {'name': x, 'user_based': True}
    algo = surprise.KNNBasic(sim_options=sim_options)

    for i, (trainset, testset) in enumerate(kf.split(data)):
        algo.fit(trainset)
        predictions = algo.test(testset)
        acc.append(surprise.accuracy.rmse(predictions, verbose=True))
    A = np.mean(acc)
    KBs_mean_acc_list.append(A)
print("KNNBasic Done")
```

<코드 12> user-based recommendation에서 KNNBasic에 대한 cross validation

- 실험의 cross validation에 대한 결과값은 <코드 13>과 같으며, <표 10>으로 정리된다.

```
print(KBs_mean_acc_list)
print(KW_mean_acc_list)
print(KBl_mean_acc_list)
```

```
[1.0112282026979647, 1.0216741648055838, 1.032343446934029, 0.9493713079129439]
[1.0238696641161662, 1.0302935234393056, 1.0486242003685746, 0.9616064943570086]
[0.9443915469016767, 0.9453234797449589, 0.9459466123769327]
```

<코드 13>

Algorithm	변경값	RMSE 평균
KNNBasic	cosine	1.0112
KNNBasic	pearson	1.0216
KNNBasic	MSD	1.0323
KNNBasic	pearson_baseline	0.9493
KNNWithMeans	cosine	1.0238
KNNWithMeans	pearson	1.0302
KNNWithMeans	MSD	1.0486
KNNWithMeans	pearson_baseline	0.9616
KNNBaseline	n_epochs:5, reg_u:10, reg_i=5	0.9443
KNNBaseline	n_epochs:10, reg_u:20, reg_i=15	0.9453
KNNBaseline	n_epochs:20, reg_u:30, reg_i=20	0.9459

<표 10>

- 실험 결과, RMSE가 가장 작게 나오는 best model은 KNNBaseline (유사도:pearson_baseline, n_epochs:5, reg_u:10, reg_i=5)이다.

3. R3-3: item_based recommendation 진행

(1) 알고리즘 3-3-①: KNNBasic (유사도 : cosine)

- algorithm의 정의는 <코드 10>과 같으나, user_based=False로 설정하였다.
- 본 알고리즘을 따라 도출된 결과값의 일부는 아래 <그림 9>와 같다.

```
Bundle NAME Borderlands Triple Pack top-10 results
User ID 8027368512
        score 3.852004306804694
User ID 7961040696
        score 3.733905223103874
User ID 8095204217
        score 3.188582531780235
User ID 8070632131
        score 2.8401860927451734
User ID 8053431839
        score 2.692104697078125
User ID 8054433999
        score 2.599298601983482
User ID 8035567225
        score 2.5759587682060587
User ID 8036934637
        score 2.5256376165942007
User ID 8042119839
        score 2.3961460375643573
User ID 8057371706
        score 2.371123022025473
```

<그림 9> 알고리즘 3-3-① 결과값 中
bundle_name 'Borderlands Triple Pack'에 대한 결과값

(2) 알고리즘 3-3-②: KNNWithMeans (유사도 : pearson)

- algorithm의 정의는 <코드 11>과 같으나, user_based=False로 설정하였다.
- 본 알고리즘을 따라 도출된 결과값의 형식은 <그림 9>를 따른다.

(3) Item_based 알고리즘 실험 후 cross validation 시행

- 알고리즘 실험과 cross validation 실행의 형식은 2-(3)과 동일하다.

- 단, User_based=False로 변경하였다.
- mean_acc_list는 <표 11>과 같다.

List Name	내용
KBs_mean_acc_list2	알고리즘 KNNBasic에 대한 acc 리스트 평균값
KW_mean_acc_list2	알고리즘 KNNWithMeans에 대한 acc 리스트 평균값
KBl_mean_acc_list2	알고리즘 KNNBaseline에 대한 acc 리스트 평균값

<표 11>

- 실험의 cross validation에 대한 결과값은 <코드 14>와 같으며, <표 12>로 정리된다.

```
print(KBs_mean_acc_list2)
print(KW_mean_acc_list2)
print(KBl_mean_acc_list2)
```

```
[1.5914670332993544, 1.623269011232803, 1.423689057726444, 1.6791994298693207]
[1.0794503093779855, 1.0241653485391737, 1.055274827219639, 1.0522692566176044]
[1.0510589395668501, 1.0534694037513368, 1.054111168417108]
```

<코드 14>

Algorithm	변경값	RMSE 평균
KNNBasic	cosine	1.5914
KNNBasic	pearson	1.6232
KNNBasic	MSD	1.4236
KNNBasic	pearson_baseline	1.6791
KNNWithMeans	cosine	1.0794
KNNWithMeans	pearson	1.0241
KNNWithMeans	MSD	1.0552
KNNWithMeans	pearson_baseline	1.0522
KNNBaseline	n_epochs:5, reg_u:10, reg_i=5	1.0510
KNNBaseline	n_epochs:10, reg_u:20, reg_i=15	1.0534
KNNBaseline	n_epochs:20, reg_u:30, reg_i=20	1.0541

<표 12>

- 실험 결과, RMSE가 가장 작게 나오는 best model은 KNNWithMeans (유사도: pearson)이다.

4. R3-4: Matrix-factorization recommendation 진행

(1) 알고리즘 3-4-①: SVD (n_factors=100, n_epoch=50, biased=False)
& 알고리즘 3-4-②: SVD (n_factors=200, n_epoch=100, biased=True)

- SVD algorithm의 정의 형식은 <코드 15>과 같다.

```
algo = surprise.prediction_algorithms.matrix_factorization.SVD(n_factors=100, n_epochs=50, biased=False)
```

<코드 15> 알고리즘 3-4-①

- 본 알고리즘들을 따라 도출된 결과값의 형식은 <그림 8>을 따른다.

(2) 알고리즘 3-4-③: SVD++ (n_factors=100, n_epoch=50)
& 알고리즘 3-4-④: SVD++ (n_factors=100, n_epoch=100)

- SVD++ algorithm의 정의 형식은 <코드 16>과 같다.

```
algo = surprise.prediction_algorithms.matrix_factorization.SVDpp(n_factors=100, n_epochs=50)
```

<코드 16> 알고리즘 3-4-③

- 본 알고리즘들을 따라 도출된 결과값의 형식은 <그림 8>을 따른다.

(3) Matrix-factorization 알고리즘 실험 후 cross validation 시행

- KFold의 n_split=5, random_state=0으로 설정하였다.
- 실험한 알고리즘은 SVD, SVD++, NMF로 총 3가지이다.
- SVD와 NMF의 경우 (n_factors, n_epochs, biased)를 4가지로 변경해가며 실험을 진행하였다.
- SVD++의 경우 (n_factors, n_epochs)를 4가지로 변경해가며 실험을 진행하였다.
- Cross validation은 RMSE값을 기준으로 설정하였다.
- 각 실험당 5개 fold의 RMSE는 acc 리스트에 담기도록 했으며, acc 리스트의 평균값을 각 알고리즘의 mean_acc_list에 삽입하였다. mean_acc_list는 <표 13>과 같다.

List Name	내용
SVD_mean_acc_list	알고리즘 SVD에 대한 acc 리스트 평균값
SVDpp_mean_acc_list	알고리즘 SVDpp에 대한 acc 리스트 평균값
NMF_mean_acc_list	알고리즘 NMF에 대한 acc 리스트 평균값

<표 13>

- SVD algorithm을 이용한 실험의 코드는 <코드 17>과 같다.
- 다른 알고리즘에 대한 cross validation도 <코드 17> 형식에 맞춰 코딩되었다.

```
print("Cross validation...")
np.random.seed(0)
kf = surprise.model_selection.split.KFold(n_splits=5, random_state=0)

SVD_sim_list = [(100, 50, True), (100, 50, False), (200, 100, True), (200, 100, False)]
SVD_mean_acc_list = []

for x, y, z in SVD_sim_list:
    acc = []
    algo = surprise.prediction_algorithms.matrix_factorization.SVD(n_factors=x, n_epochs=y, biased=z)

    for i, (trainset, testset) in enumerate(kf.split(data)):
        algo.fit(trainset)
        predictions = algo.test(testset)
        acc.append(surprise.accuracy.rmse(predictions, verbose=True))
    A = np.mean(acc)
    SVD_mean_acc_list.append(A)
print("SVD Done")
```

<코드 17> Matrix-factorization recommendation에서 SVD에 대한 cross validation

- 실험의 cross validation에 대한 결과값은 <코드 18>과 같으며, <표 14>로 정리된다.

```
print(SVD_mean_acc_list)
print(SVDpp_mean_acc_list)
print(NMF_mean_acc_list)
```

```
[0.9395740232322712, 0.9466833818671556, 0.933572802037377, 0.9443051150444992]
[0.9405960898342322, 0.9306261232610623, 0.9278415922400518, 0.925706974094835 3]
[1.3224434540746204, 0.9286685005186538, 1.2989565516085162, 0.9434709186977234]
```

<코드 18>

Algorithm	변경값	RMSE 평균
SVD	n_factors=100, n_epoch=50, biased=True	0.9395
SVD	n_factors=100, n_epoch=50, biased=False	0.9466
SVD	n_factors=200, n_epoch=100, biased=True	0.9335
SVD	n_factors=200, n_epoch=100, biased=False	0.9443
SVDpp	n_factors=70, n_epoch=30	0.9405
SVDpp	n_factors=100, n_epoch=50	0.9306
SVDpp	n_factors=150, n_epoch=100	0.9278
SVDpp	n_factors=150, n_epoch=150	0.9257
NMF	n_factors=100, n_epoch=50, biased=True	1.3224
NMF	n_factors=100, n_epoch=50, biased=False	0.9286
NMF	n_factors=200, n_epoch=100, biased=True	1.2989
NMF	n_factors=200, n_epoch=100, biased=False	0.9434

<표 14>

- 실험 결과, RMSE가 가장 작게 나오는 best model은 SVDpp (n_factors=150, n_epoch=150)이다.