# PROJECT #2

: DB mining & Automated Recommendation System

Team 03 - 김민영, 김해찬, 오태양, 하창우

# PROJECT #2

PART I :  의사 결정 나무

PART II :  연관 분석

PART III :  추천 시스템

## PART I : 의사결정 나무

Requirement 1-1 : item table 에 best_item column 추가

Requirement 1-2 : item에 대한 추가 정보 획득

Requirement 1-3 : Decision Tree 모델 학습

Requirement 1-4 : New best 아이템 선정 기준 설정 및 평가

## PART I : 의사결정 나무

### Requirement 1-1

```python
# TODO: Requirement 1-1. MAKE best_item column
try:
    print('adding new column \'best_item\'')
    cursor.execute('ALTER TABLE item ADD best_item TINYINT(1) not null default 0;')
except mysql.connector.errors.ProgrammingError:
    print('best_item column already exists')

# create 'best_item_list' with txt given
best_item_list=[]
with open('best_item_list.txt', 'r', encoding='utf-8') as best_item_data:
    while True:
        line=best_item_data.readline()
        if not line:break
        best_item_list.append(line.strip())
    print('best_item_list : ', best_item_list)

# about the list 'best_item_list', update mySQL DB
print('Updating best_item column in item table')
for best_item in best_item_list:
    sql_update = '''UPDATE item
    SET best_item = 1
    WHERE id=%s''' % best_item
    cursor.execute(sql_update)
    cnx.commit()
```

Step 1:
best_item column 삽입

Step 2:
best_item datatype 설정

Step 3:
best_item_list.txt 에 속하는 item tuple의
best_item 값을 1로 설정

## PART I : 의사결정 나무

### Requirement 1-2

```
# TODO: Requirement 1-2. WRITE MYSQL QUERY AND EXECUTE. SAVE to .csv file
print('Collecting information about items on DB...')
sql_item_detail='''SELECT I.item_id, best_item, ratings, COALESCE(num_of_specs, 0) as num_of_specs, COALESCE(num_of_tags, 0) as num_of_tags,\
    COALESCE(num_of_users, 0) as num_of_users, COALESCE(avg_usage_time, 0) as avg_usage_time, COALESCE(num_of_reviews, 0) as num_of_reviews,\
        COALESCE(sum_of_recommend, 0) as sum_of_recommend, COALESCE(avg_review_len, 0) as avg_review_len
FROM(SELECT id as item_id, best_item, ratings FROM item) AS I
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_specs FROM item_specs GROUP BY item_id) AS S
ON I.item_id=S.item_id
LEFT JOIN(SELECT item_id, count(*) as num_of_tags FROM tag GROUP BY item_id) AS T
ON I.item_id=T.item_id
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_users, avg(usagetime_total) as avg_usage_time FROM user_item GROUP BY item_id) AS UI
ON I.item_id=UI.item_id
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_reviews, SUM(recommend) as sum_of_recommend, AVG(body) as avg_review_len FROM review GROUP BY item_id)
ON I.item_id=R.item_id
'''
cursor.execute(sql_item_detail)
item_detail = pd.DataFrame(cursor.fetchall())
item_detail.columns = cursor.column_names
item_detail.to_csv('DMA_project2_team%02d_part1.csv' % team, index=False)
```

Step : Requirement에서 주어진 10개의 column을 확인

(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)

## PART I : 의사결정 나무

### Requirement 1-2

```
# TODO: Requirement 1-2. WRITE MYSQL QUERY AND EXECUTE. SAVE to .csv file
print('Collecting information about items on DB...')
sql_item_detail='''SELECT I.item_id, best_item, ratings, COALESCE(num_of_specs, 0) as num_of_specs, COALESCE(num_of_tags, 0) as num_of_tags,\
    COALESCE(num_of_users, 0) as num_of_users, COALESCE(avg_usage_time, 0) as avg_usage_time, COALESCE(num_of_reviews, 0) as num_of_reviews,\
        COALESCE(sum_of_recommend, 0) as sum_of_recommend, COALESCE(avg_review_len, 0) as avg_review_len
FROM(SELECT id as item_id, best_item, ratings FROM item) AS I
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_specs FROM item_specs GROUP BY item_id) AS S
ON I.item_id=S.item_id
LEFT JOIN(SELECT item_id, count(*) as num_of_tags FROM tag GROUP BY item_id) AS T
ON I.item_id=T.item_id
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_users, avg(usagetime_total) as avg_usage_time FROM user_item GROUP BY item_id) AS UI
ON I.item_id=UI.item_id
LEFT JOIN(SELECT item_id, COUNT(*) as num_of_reviews, SUM(recommend) as sum_of_recommend, AVG(body) as avg_review_len FROM review GROUP BY item_id)
ON I.item_id=R.item_id
'''
cursor.execute(sql_item_detail)
item_detail = pd.DataFrame(cursor.fetchall())
item_detail.columns = cursor.column_names
item_detail.to_csv('DMA_project2_team%02d_part1.csv' % team, index=False)
```

Step : 각각의 table에서 가공후 item table에서 LEFT OUTER JOIN -> view 생성

## PART I : 의사결정 나무

### Requirement 1-3

```python
# TODO: Requirement 1-3. MAKE AND SAVE DECISION TREE
# gini file name: DMA_project2_team##_part1_gini.pdf
# entropy file name: DMA_project2_team##_part1_entropy.pdf

# Feature_names list

dt_feature_names = []
with open('DMA_project2_team%02d_part1.csv' % team, 'r', encoding='utf-8') as i_header:
    header = i_header.readline()
    header = header.strip().split(sep=',')
    for dt_feature_name in header:
        dt_feature_names.append(dt_feature_name)
del dt_feature_names[:2]

# Data Preprocessed
dt_class = item_detail.drop(dt_feature_names, axis=1)
dt_class = dt_class.drop(['item_id'], axis=1)
dt_feature = item_detail.drop(['item_id', 'best_item'], axis=1)
dt_class = dt_class.to_numpy()
dt_feature = dt_feature.to_numpy()
print(dt_class)

# Decision Tree_gini
dt_gini = tree.DecisionTreeClassifier(criterion='gini', min_samples_leaf=10, max_depth=5)
dt_gini.fit(X=dt_feature, y=dt_class)
print('DT gini parpameter : ', dt_gini.get_params())

graph_dt_gini = tree.export_graphviz(dt_gini, out_file=None,
                                     feature_names=dt_feature_names, class_names=['normal', 'BEST'])
graph_dt_gini = graphviz.Source(graph_dt_gini)
graph_dt_gini.render('DMA_project2_team%s_part1_gini' % team, view=True)

# Decision Tree_entropy
dt_entropy = tree.DecisionTreeClassifier(criterion='entropy', min_samples_leaf=10, max_depth=5)
dt_entropy.fit(X=dt_feature, y=dt_class)
print('DT entropy parpameter : ', dt_entropy.get_params())

graph_dt_entropy = tree.export_graphviz(dt_entropy, out_file=None,
                                        feature_names=dt_feature_names, class_names=['normal', 'BEST'])
graph_dt_entropy = graphviz.Source(graph_dt_entropy)
graph_dt_entropy.render('DMA_project2_team%s_part1_entropy' % team, view=True)
```

Step 1: Data 전처리

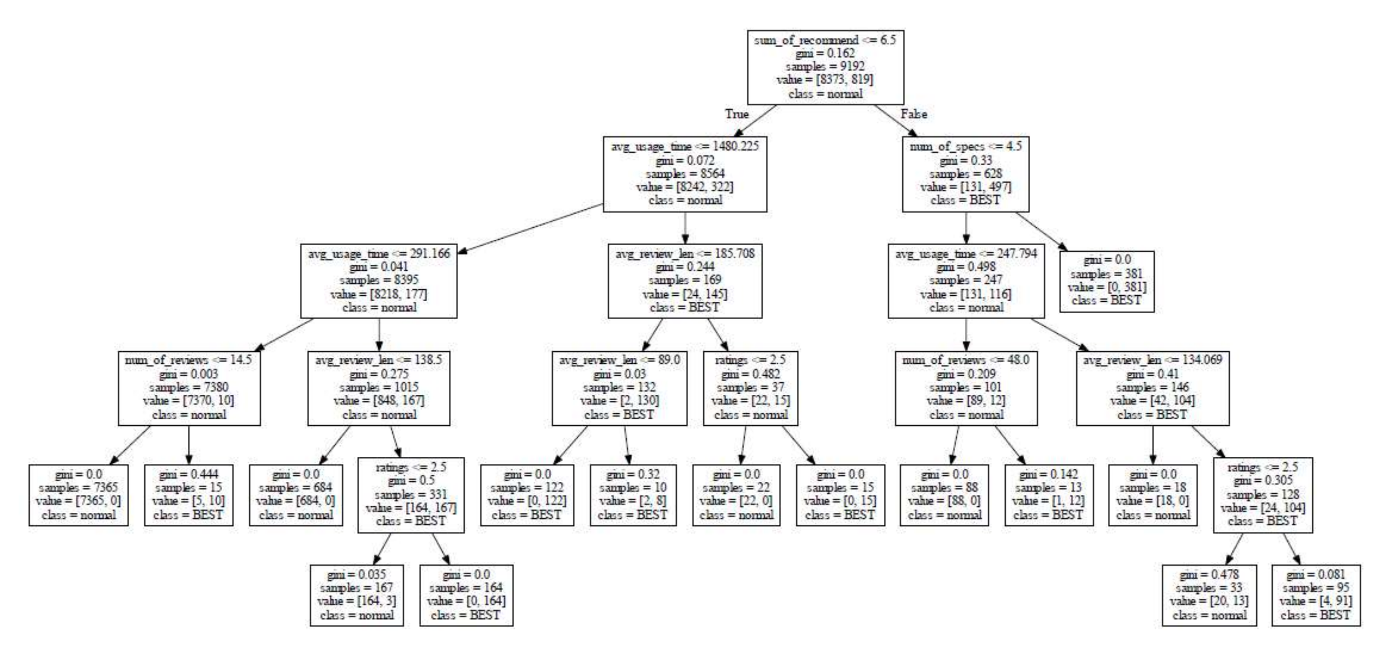
Step 2: Decision Tree - Gini

Step 3: Decision Tree - Entropy

## PART I : 의사결정 나무

**Requirement 1-3 :** Gini

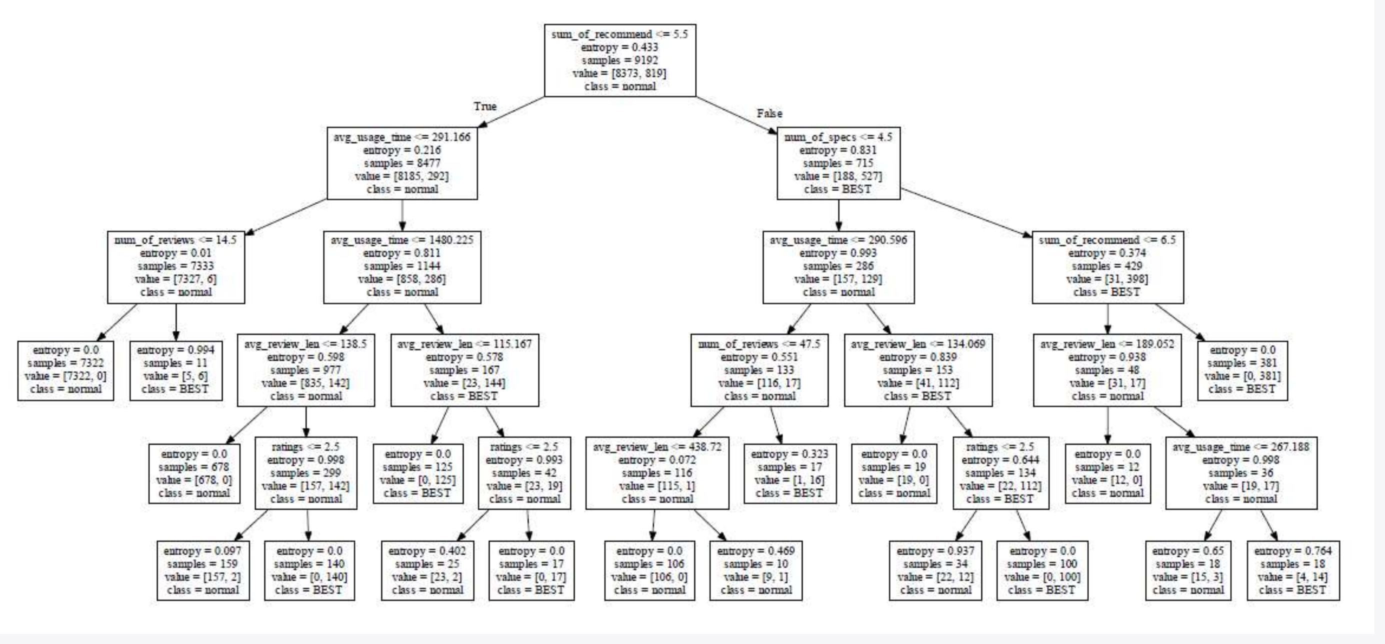
| DT gini parpameter | |
|---|---|
| 'ccp_alpha' | 0.0 |
| 'class_weight' | None |
| 'criterion' | 'gini' |
| 'max_depth' | 5 |
| 'max_features' | None |
| 'max_leaf_nodes' | None |
| 'min_impurity_decrease' | 0.0 |
| 'min_impurity_split' | None |
| 'min_samples_leaf' | 10 |
| 'min_samples_split' | 2 |
| min_weight_fraction_leaf' | 0.0 |
| 'random_state' | None |
| 'splitter' | 'best' |



## PART I : 의사결정 나무

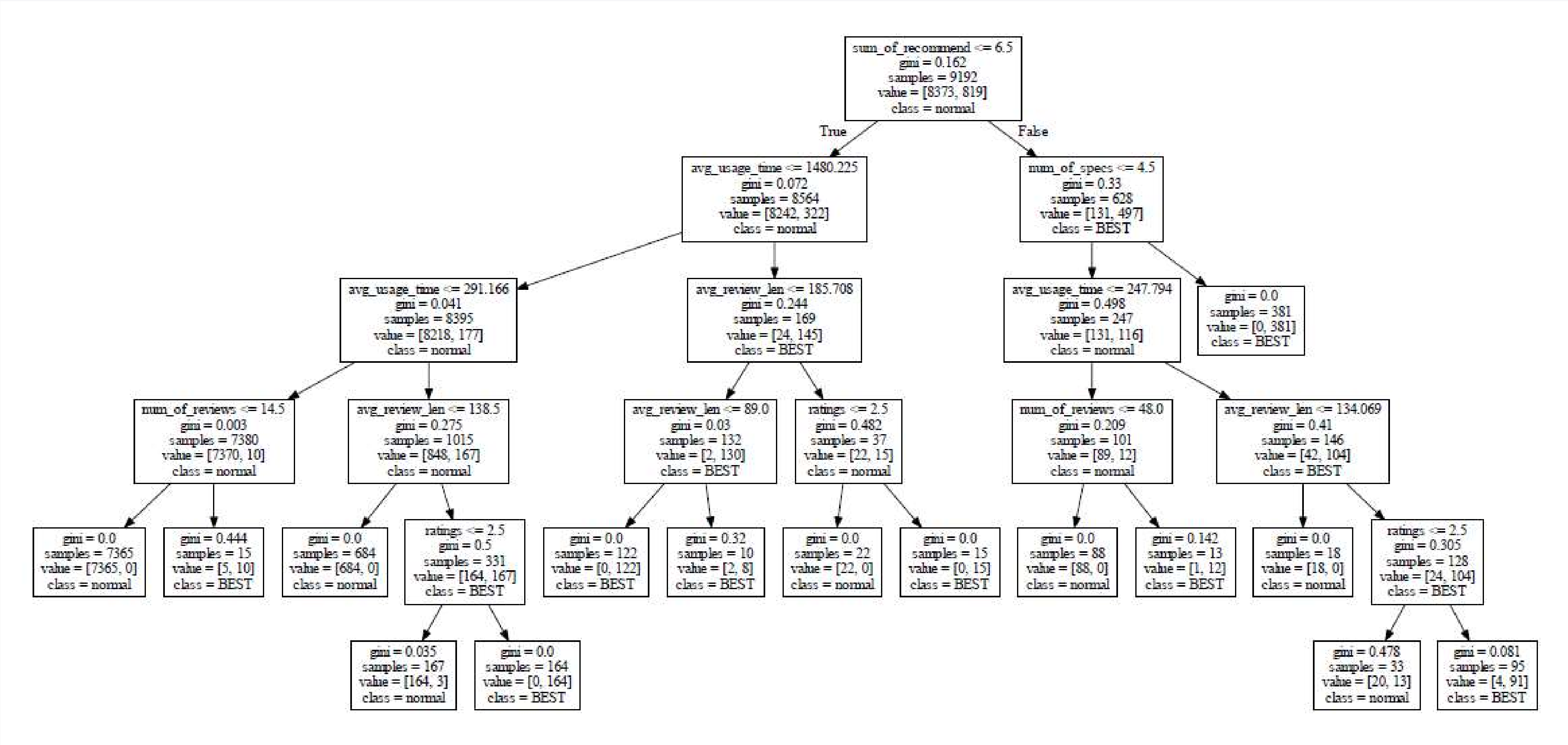
### Requirement 1-3 : Entropy

| DT entropy parpameter | |
|---|---|
| 'ccp_alpha' | 0.0 |
| 'class_weight' | None |
| 'criterion' | 'entropy' |
| 'max_depth' | 5 |
| 'max_features' | None |
| 'max_leaf_nodes' | None |
| 'min_impurity_decrease' | 0.0 |
| 'min_impurity_split' | None |
| 'min_samples_leaf' | 10 |
| 'min_samples_split' | 2 |
| min_weight_fraction_leaf' | 0.0 |
| 'random_state' | None |
| 'splitter' | 'best' |



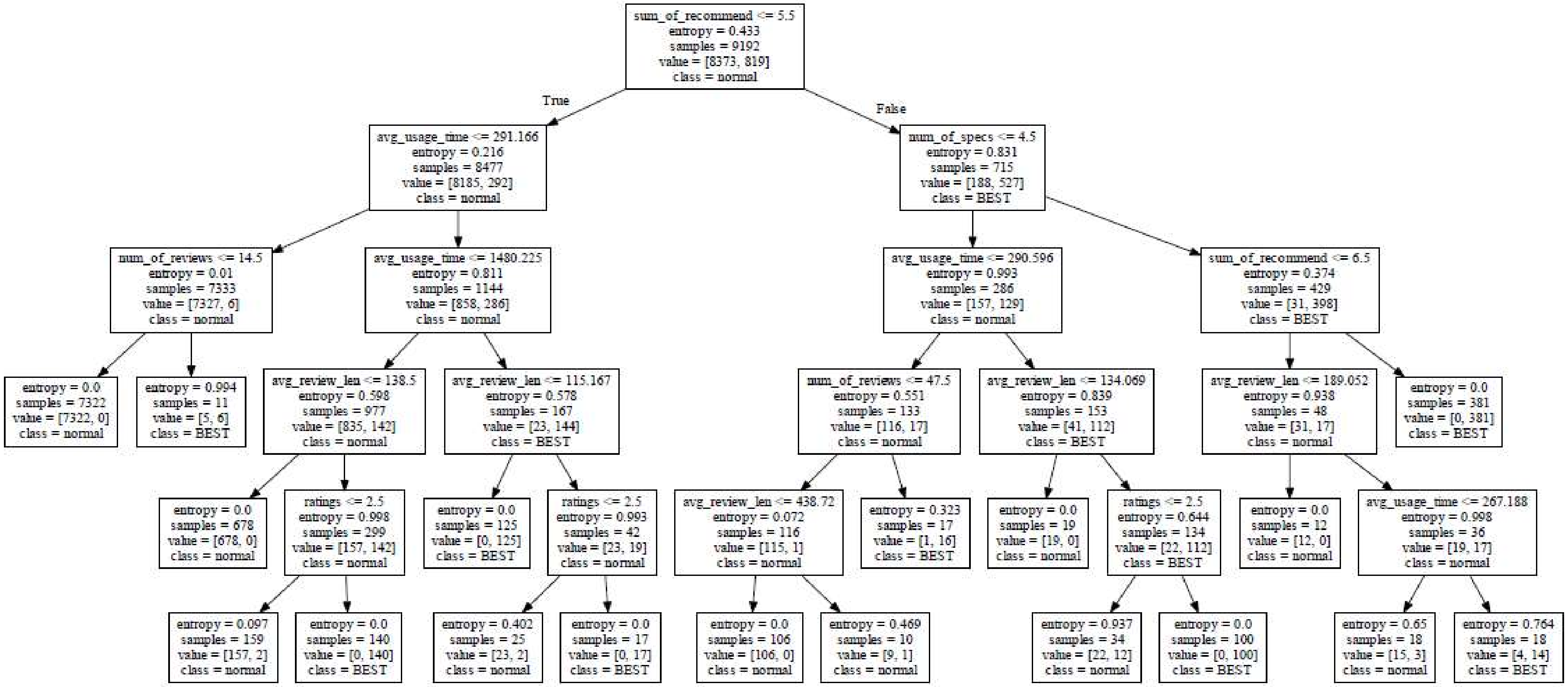
# PART I : 의사결정 나무

## Requirement 1-3 : Gini



# PART I : 의사결정 나무

## Requirement 1-3 : Entropy



## PART I : 의사결정 나무

### Requirement 1-4

Step I: 입력값
- num_of_tag 제외
- sum_of_recommend 보정

Step 2: Decision Tree 파라미터
- Gini criterion 사용
- max_depth = 5 유지
- min_split_criterion=0.03 추가

Step 3: 논리의 타당성
- num_of_specs 입력값 제거

| best_item | avg(num_of_tags) |
|---|---|
| 1 | 16.5507 |
| 0 | 7.7831 |

FIG. 3: 통계치

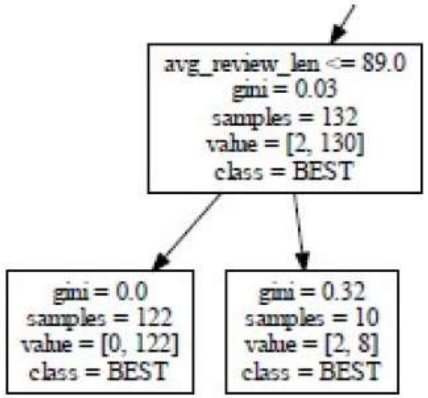



FIG. 4: gini-node

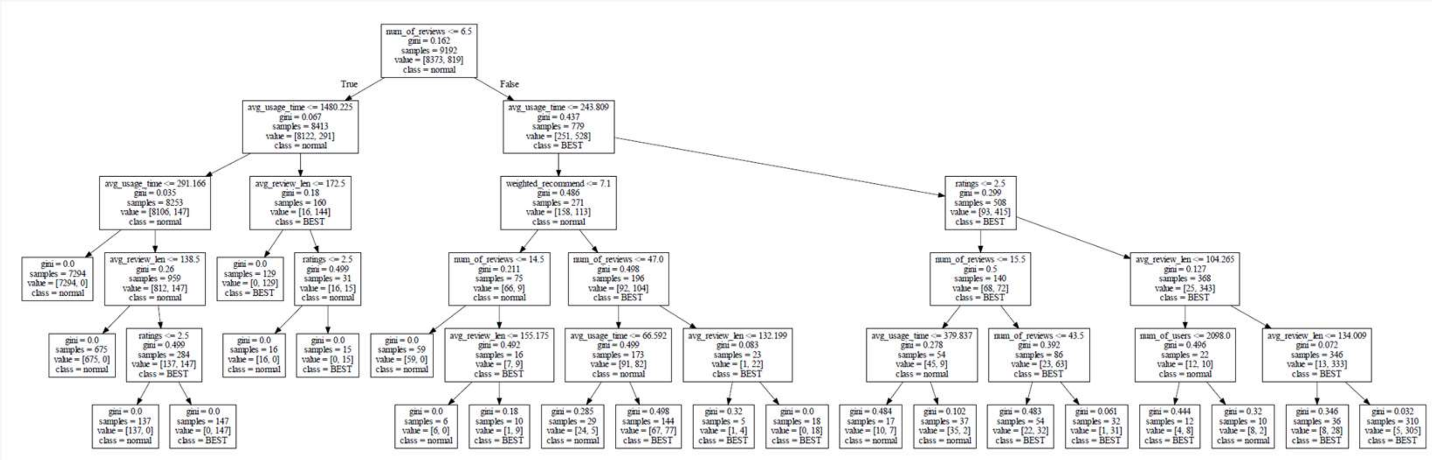
## PART I : 의사결정 나무

### Requirement 1-4

Step 1: 입력값
- num_of_tag 제외
- sum_of_recommend 보정

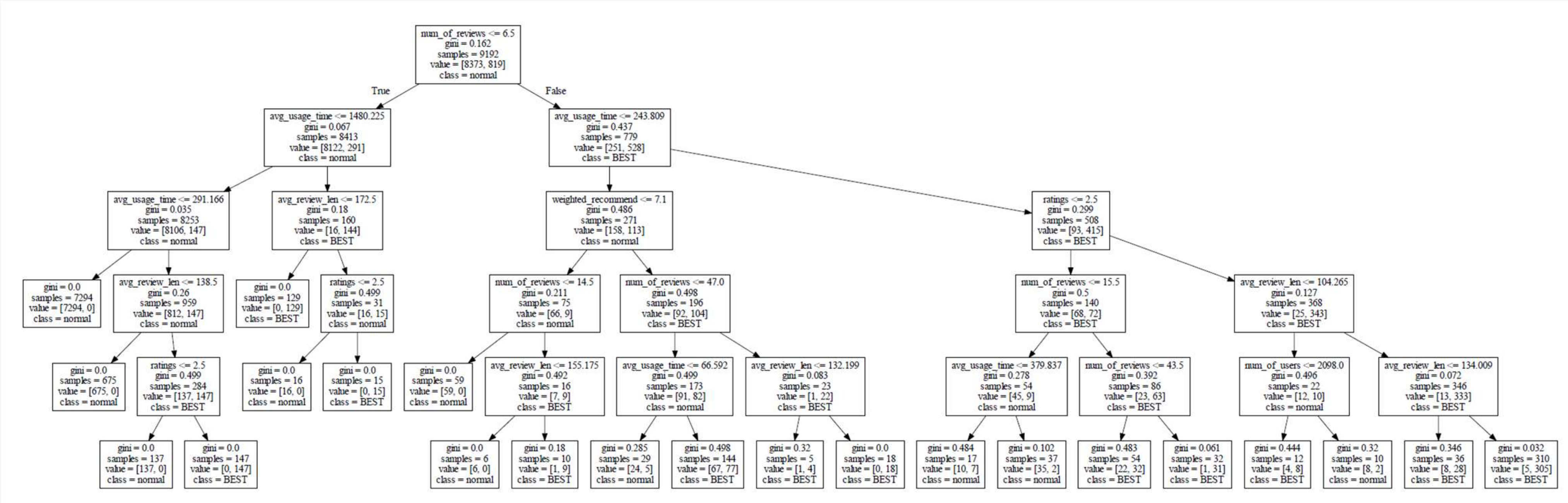
Step 2: Decision Tree 파라미터
- Gini criterion 사용
- max_depth = 5 유지
- min_split_criterion=0.03 추가

Step 3: 논리의 타당성
- num_of_specs 입력값 제거



# PART I : 의사결정 나무

## Requirement 1-4



## PART II : 연관 분석

Requirement 2-1 : bundle_score view 생성

Requirement 2-2 : user_bundle_rating(UBR) view 생성

Requirement 2-3 : partial_user_bundle_rating의 horizontal table 변환 및 DataFrame 생성

Requirement 2-4 : frequent itemset 생성 및 연관분석, 정성적 & 정량적 평가

# PART II : 연관 분석

## Requirement 2-1

```python
# TODO: Requirement 2-1. CREATE VIEW AND SAVE to .csv file

print("2-1. Making User-bundle score...")

fopen = open('DMA_project2_team%02d_part2_bundle.csv' % team, 'w', encoding='utf-8')

bundle_score_columns = ['bundle_id', 'bundle_name', 'num_item', 'num_genre','num_user','score']

make_bundle_score = '''
SELECT *, num_item+num_genre+num_user as score FROM(
SELECT B.bundle_id, B.bundle_name, 100*BI.num_item as num_item, coalesce(100*BG.num_genre,0) as num_genre, num_user/BI.num_item as num_user
FROM (SELECT id as bundle_id, bundle_name FROM bundle) AS B
LEFT JOIN (SELECT bundle_id, COUNT(*) as num_item FROM bundle_item GROUP BY bundle_id) AS BI
ON B.bundle_id = BI.bundle_id
LEFT JOIN (SELECT bundle_id, COUNT(*) as num_genre FROM bundle_genre GROUP BY bundle_id) AS BG
ON B.bundle_id = BG.bundle_id
LEFT JOIN (SELECT bundle_id, SUM(num_user) as num_user FROM(
SELECT bundle_id, bundle_item.item_id, COUNT(*) as num_user FROM (bundle_item left join user_item on bundle_item.item_id=user_item.item_id)\
GROUP BY bundle_id*bundle_item.item_id) AS A
GROUP BY bundle_id ) AS BU
ON B.bundle_id = BU.bundle_id) AS F;
'''

cursor.execute(make_bundle_score)

bundle_score = pd.DataFrame(cursor.fetchall())
bundle_score.columns = cursor.column_names

bundle_score = bundle_score.sort_values('score', ascending=False)
bundle_score = bundle_score[0:30]
bundle_score.to_csv('DMA_project2_team%02d_part2_bundle.csv' % team, index=False)
fopen.close()
```

Step 1:
bundle, bundle_item, bundle_genre, user_item table을 활용하여
Bundle_score 구현

Step 2:
Score 상위 30개 filtering
-> bundle_score view 구현

## PART II : 연관 분석

### Requirement 2-2

```python
# TODO: Requirement 2-2. CREATE 2 VIEWS AND SAVE partial one to .csv file

print("2-2. Making User-bundle partial rating score...")

#top_30_bundle_list derived from bundle_score
top_bundle_list = bundle_score.bundle_id.unique()
top_bundle_list_str = "\', \'".join(map(str,top_bundle_list))
top_bundle_list_str = "\'"+top_bundle_list_str+"\'"

make_user_bundle_rating = '''
SELECT user_id, bundle_name, num_recommend+num_use as rating
FROM
(SELECT user_id, bundle_name, if(count(item_id)>5,5,count(item_id)) as num_use , 5*count(if(recommend=1,1,null)) as num_recommend
FROM
(SELECT user_id, bundle_name, item_id, coalesce(recommend, 0) as recommend
FROM (select bundle_id, item_id from bundle_item where bundle_id in (%s) as BI
LEFT JOIN (select id, bundle_name from bundle) AS B
ON bundle_id=id
LEFT JOIN (select user_id, item_id as item_id2 from user_item) as UI
ON item_id = item_id2
LEFT JOIN (select user_id as user_id3, item_id as item_id3, recommend from review) as R
ON user_id = user_id3 AND item_id = item_id3) as T
GROUP BY user_id*bundle_name) as F;
'''%(top_bundle_list_str)

cursor.execute(make_user_bundle_rating)
user_bundle_rating = pd.DataFrame(cursor.fetchall())
user_bundle_rating.columns = ['user', 'bundle_name', 'rating']

#partial_version
partial_user_bundle_rating = user_bundle_rating.copy()
freq_user = user_bundle_rating.user.value_counts()
freq_user = freq_user[freq_user>19]
partial_user_bundle_rating = partial_user_bundle_rating[partial_user_bundle_rating['user'].apply(lambda x : x in freq_user)]

fopen = open('DMA_project2_team%02d_part2_UBR.csv' % team, 'w', encoding='utf-8')
partial_user_bundle_rating.to_csv('DMA_project2_team%02d_part2_UBR.csv' % team, index=False)
fopen.close()
```

Step 2:
bundle_item, user_item, review
User_id와 item_id를 기준으로 LEFT OUTER JOIN

Rating equation에 따라 rating

Step 3:
freq_user > 19 만족하는 UBR을 filtering
-> partial_user_bundle_rating view 구현

## PART II : 연관 분석

### Requirement 2-3

```python
# TODO: Requirement 2-3. MAKE HORIZONTAL VIEW
# file name: DMA_project2_team##_part2_horizontal.pkl
# use to_pickle(): df.to_pickle(filename)

print("2-3. Making Horizontal View...")
horizontal = np.zeros((len(freq_user), len(top_bundle_list)), dtype=bool)
horizontal = pd.DataFrame(horizontal)
horizontal.columns = bundle_score.bundle_name.unique()
horizontal.index = freq_user.index


for b_id in bundle_score.bundle_name.unique():
    series_of_user = partial_user_bundle_rating[partial_user_bundle_rating.bundle_name==b_id].user
    series = horizontal[b_id]
    for user in series_of_user:
        series[user] = True
    horizontal[b_id] = series


horizontal.to_pickle("DMA_project2_team03_part2_horizontal.pkl")
```

Step 1:
Req 2-1에서 구현한 bundle_score를 통해
top_bundle_list 구현

Step 2:
bundle_item, user_item, review
User_id와 item_id를 기준으로 LEFT OUTER JOIN

Rating equation에 따라 rating

## PART II : 연관 분석

### Requirement 2-4

```
# TODO: Requirement 2-4. ASSOCIATION ANALYSIS
# filename: DMA_project2_team##_part2_association.pkl (pandas dataframe)
print("2-4. frequent_itemset...")
frequent_itemset = apriori(horizontal, min_support = 0.35, use_colnames=True)
print(frequent_itemset)
print("2-4. association rules...")
rules = association_rules(frequent_itemset, metric='lift', min_threshold=2)
print(rules.to_string())
rules.to_pickle("DMA_project2_team03_part2_association.pkl")
cursor.close()
```

Step :
Frequent itemset 생성

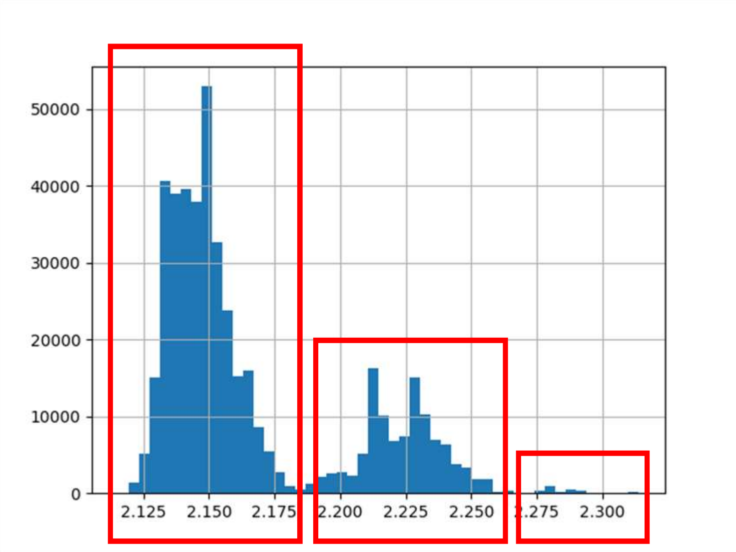
## PART II : 연관 분석

### Requirement 2-4

Rule data의 통계량 ( 445544개의 rule )

| | antecedent | consequent | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|
| count | 446544 | 446544 | 446544 | 446544 | 446544 | 446544 | 446544 |
| mean | 0.418 | 0.418 | 0.377 | 0.904 | 2.166 | 0.203 | inf |
| std | 0.027 | 0.027 | 0.020 | 0.057 | 0.036 | 0.010 | NaN |
| min | 0.350 | 0.350 | 0.350 | 0.747 | 2.115 | 0.185 | 2.57E+00 |
| 25% | 0.399 | 0.399 | 0.358 | 0.865 | 2.140 | 0.194 | 4.45E+00 |
| 50% | 0.420 | 0.420 | 0.373 | 0.912 | 2.150 | 0.202 | 6.54E+00 |
| 75% | 0.438 | 0.438 | 0.390 | 0.947 | 2.177 | 0.210 | 1.06E+01 |
| max | 0.469 | 0.469 | 0.457 | 1.000 | 2.314 | 0.243 | inf |

Rule의 lift value 분포-histogram



## PART II : 연관 분석

### Requirement 2-4

| | antecedent | consequent | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|
| count | 446544 | 446544 | 446544 | 446544 | 446544 | 446544 | 446544 |
| mean | 0.418 | 0.418 | 0.377 | 0.904 | 2.166 | 0.203 | inf |
| std | 0.027 | 0.027 | 0.020 | 0.057 | 0.036 | 0.010 | NaN |
| min | 0.350 | 0.350 | 0.350 | 0.747 | 2.115 | 0.185 | 2.57E+00 |
| 25% | 0.399 | 0.399 | 0.358 | 0.865 | 2.140 | 0.194 | 4.45E+00 |
| 50% | 0.420 | 0.420 | 0.373 | 0.912 | 2.150 | 0.202 | 6.54E+00 |
| 75% | 0.438 | 0.438 | 0.390 | 0.947 | 2.177 | 0.210 | 1.06E+01 |
| max | 0.469 | 0.469 | 0.457 | 1.000 | 2.314 | 0.243 | inf |

Rule data의 통계량
( 445544개의 rule )

| | antecedent | consequent | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|
| count | 2592 | 2592 | 2592 | 2592 | 2592 | 2592 | 2592 |
| mean | 0.395 | 0.395 | 0.356 | 0.903 | 2.287 | 0.200 | 8.367 |
| std | 0.020 | 0.020 | 0.003 | 0.046 | 0.010 | 0.002 | 4.936 |
| min | 0.364 | 0.364 | 0.354 | 0.832 | 2.276 | 0.198 | 3.774 |
| 25% | 0.376 | 0.376 | 0.354 | 0.858 | 2.280 | 0.199 | 4.397 |
| 50% | 0.392 | 0.392 | 0.354 | 0.903 | 2.283 | 0.200 | 6.351 |
| 75% | 0.413 | 0.413 | 0.357 | 0.946 | 2.291 | 0.201 | 10.800 |
| max | 0.425 | 0.425 | 0.364 | 0.974 | 2.314 | 0.205 | 21.828 |

Rule(lift>2.27) data의 통계량
( 2592개의 rule )

-> 더 강한 상관관계를 가지는 Rule

## PART II : 연관 분석

### Requirement 2-4

| Antecedents | Consequents |
| --- | --- |
| 'Half-Life Complete'<br>'Sid Meiers Civilization V: Complete'<br>'Grand Theft Auto V & Megalodon Shark Cash Card'<br>'Borderlands Triple Pack' | 'Borderlands Take Over Your Life Bundle'<br>'Sid Meiers Civilization Anthology'<br>'Source Multiplayer Pack'<br>'Grand Theft Auto V & Great White Shark Cash Card' |

TABLE I: Case1) confidence = 0.97, lift = 2.28

| Antecedents | Consequents |
| --- | --- |
| 'Grand Theft Auto V & Megalodon Shark Cash Card'<br>'Half-Life Complete'<br>'Borderlands Triple Pack'<br>'Sid Meiers Civilization V: Complete' | 'Eidos Anthology'<br>'Grand Theft Auto V & Whale Shark Cash Card'<br>'Grand Theft Auto V & Great White Shark Cash Card'<br>'Valve Complete Pack'<br>'Source Multiplayer Pack'<br>'Sid Meiers Civilization Anthology'<br>'Borderlands Take Over Your Life Bundle' |

TABLE II: Case2) confidence = 0.94, lift = 2.28

| Antecedents | Consequents |
| --- | --- |
| 'Half-Life Complete'<br>'Grand Theft Auto V & Whale Shark Cash Card'<br>'Borderlands Triple Pack'<br>'Sid Meiers Civilization Anthology' | 'Eidos Anthology', 'Grand Theft Auto V & Megalodon Shark Cash Card'<br>'Sid Meiers Civilization V: Complete'<br>'Grand Theft Auto V & Great White Shark Cash Card'<br>'Valve Complete Pack'<br>'Source Multiplayer Pack'<br>'Borderlands Take Over Your Life Bundle' |

TABLE III: Case3) confidence = 0.92, lift = 2.28

Antecedents에 해당하는 번들을 이용한 이력이 있을 경우,
Consequents에 해당하는 번들을 이용할 확률이 매우 높음

⇒ 유저에게 타겟 마케팅으로 활용 가능!

## PART III : 추천 시스템

Requirement 3-1 : get_top_n function

Requirement 3-2 : top-5 bundle. txt by using algorithms(KNNBasic, KNNWithMeans) / Select best algorithms

Requirement 3-3 : top-10 user.txt by using algorithms(KNNBasic, KNNWithMeans) / Select best algorithms

Requirement 3-4 : top-5 bundle.txt by using algorithms(SVD, etc) / Select best algorithms

## PART III : 추천 시스템

### Requirement 3-1

```python
# 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 = []
        for tup in testset:
            uid = tup[0]
            if uid in id_list:
                testset_id.append(tup)
        predictions = algo.test(testset_id)
        for uid, bname, true_r, est, _ in predictions:
            # TODO: results는 user_id를 key로, [(bundle_name, estimated_rating)의 tuple이 모인 list]를 value로 갖는 dictionary
            results[uid] = results[uid] + [(bname, est)]
    else:
        # TODO: testset의 데이터 중 bundle name이 id_list 안에 있는 데이터만 따로 testset_id라는 list로 저장
        # Hint: testset은 (user_id, bundle_name, default_rating)의 tuple을 요소로 갖는 list
        testset_id = []
        for tup in testset:
            name = tup[1]
            if name in id_list:
                testset_id.append(tup)
        predictions = algo.test(testset_id)
        for uid, bname, true_r, est, _ in predictions:
            # TODO: results는 bundle_name를 key로, [(user_id, estimated_rating)의 tuple이 모인 list]를 value로 갖는 dictionary
            results[bname] = results[bname] + [(uid, est)]
    for id_, ratings in results.items():
        results[id_] = sorted(results[id_], key=lambda x: x[1], reverse=True)[:n]

    return results

    # TODO: rating 순서대로 정렬하고 top-n개만 유지

    return ret
```

Step 1:
User_based case

Step 2:
Item_based case

Step 3:
Choose top-n

## PART III : 추천 시스템

### Requirement 3-2

```python
# TODO: Requirement 3-2. User-based Recommendation
uid_list = ['8051826169', '8027368512', '7998746368', '8054453794', '8030770479']
# TODO: set algorithm for 3-2-1
sim_options={'name':'cosine', 'user_based':True, 'min_support':1}
algo = surprise.KNNBasic(sim_options=sim_options)
algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-2-1.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')

# TODO: set algorithm for 3-2-2
sim_options = {'name': 'pearson', 'user_based': True, 'min_support': 1, 'shrinkage':0}
algo = surprise.KNNWithMeans(sim_options=sim_options)
algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-2-2.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')
```

Step 1:
KNNBasic algorithm

Step 2:
KNNWithMeans algorithm

# PART III : 추천 시스템

## Requirement 3-2

KNN Basic & KNNWithMeans algorithm

$$KNNBasic : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sin(u,v) \cdot r_{vi}}{\sum_{v \in N_i^K(u)} sin(u,v)}$$

$$KNNWithMeans : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sin(u,v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^K(u)} sin(u,v)} + \mu_v$$

KNN Baseline algorithm

$$KNNbaseline : \hat{r}_{ui} = \frac{\sum_{v \in N_i^K(u)} sin(u,v) \cdot (r_{vi} - b_{ui})}{\sum_{v \in N_i^K(u)} sin(u,v)} + b_{ui}$$

## PART III : 추천 시스템

### Requirement 3-2

```python
# TODO: 3-2-3. Best Model
best_algo_ub = surprise.KNNBaseline(sim_options={'name':'pearson_baseline', 'user_based':True, 'min_support':1, 'shrinkage':1})
```

평가 지표: RMSE(Root mean square error)

| 유사도함수/알고리즘 | KNNBasic | KNNWithMeans | KNNBaseline |
|---|---|---|---|
| Pearson | 1.026 | 1.034 | 1.01 |
| Cosine | 1.016 | 1.028 | 1.004 |
| MSD | 1.037 | 1.052 | 1.025 |
| Pearson_Baseline | 0.958 | 0.974 | 0.954 |

Best algorithm

## PART III : 추천 시스템

### Requirement 3-3

```
# TODO: Requirement 3-3. Item-based Recommendation
bname_list = ['World of Magicka Bundle',
              'Borderlands Triple Pack',
              'Tripwire Complete Bundle',
              'Grand Theft Auto V & Great White Shark Cash Card',
              'Killing Floor 1 Complete Your Set!']
# TODO - set algorithm for 3-3-1
sim_options = {'name': 'cosine', 'user_based': False, 'min_support': 1}
algo = surprise.KNNBasic(sim_options=sim_options)
algo.fit(trainset)
results = get_top_n(algo, testset, bname_list, n=10, user_based=False)
with open('3-3-1.txt', 'w') as f:
    for bname, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('Bundle NAME %s top-10 results\n' % bname)
        for uid, score in ratings:
            f.write('User ID %s\n\tscore %s\n' % (uid, str(score)))
        f.write('\n')

# TODO: set algorithm for 3-3-2
sim_options = {'name': 'pearson', 'user_based': False, 'min_support': 1, 'shrinkage': 0}
algo = surprise.KNNWithMeans(sim_options=sim_options)
algo.fit(trainset)
results = get_top_n(algo, testset, bname_list, n=10, user_based=False)
with open('3-3-2.txt', 'w') as f:
    for bname, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('Bundle NAME %s top-10 results\n' % bname)
        for uid, score in ratings:
            f.write('User ID %s\n\tscore %s\n' % (uid, str(score)))
        f.write('\n')
```

Step 1:
KNNBasic algorithm

Step 2:
KNNWithMeans algorithm

## PART III : 추천 시스템

### Requirement 3-3

```
# TODO: 3-3-3. Best Model
best_algo_ib = surprise.KNNWithMeans(sim_options={'name':'pearson', 'user_based':False, 'min_support':1})
```

평가 지표: RMSE(Root mean square error)

| 유사도함수/알고리즘 | KNNBasic | KNNWithMeans | KNNBaseline |
|---|---|---|---|
| Pearson | 1.621 | 1.021 | 1.021 |
| Cosine | 1.593 | 1.081 | 1.081 |
| MSD | 1.426 | 1.057 | 1.057 |
| Pearson_Baseline | 1.680 | 1.058 | 1.056 |

Best algorithm

## Requirement 3-4

```python
# TODO: Requirement 3-4. Matrix-factorization Recommendation
# TODO: set algorithm for 3-4-1
algo = surprise.SVD(n_factors=100, n_epochs=50, biased=False)
algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-4-1.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')

# TODO: set algorithm for 3-4-2
algo = surprise.SVD(n_factors=200, n_epochs=100, biased=False)
algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-4-2.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')
```

Step 1:
SVD algorithm
(n_factors=100, n_epoch=50, biased=False)

Step 2:
SVD algorithm
(n_factors=200, n_epoch=100, biased=True)

## PART III : 추천 시스템

### Requirement 3-4

```
# TODO: set algorithm for 3-4-3
algo = surprise.SVDpp(n_factors=100, n_epochs=50)

algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-4-3.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')

# TODO: set algorithm for 3-4-4
algo = surprise.SVDpp(n_factors=100, n_epochs=100)
algo.fit(trainset)
results = get_top_n(algo, testset, uid_list, n=5, user_based=True)
with open('3-4-4.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-5 results\n' % uid)
        for bname, score in ratings:
            f.write('Bundle NAME %s\n\tscore %s\n' % (bname, str(score)))
        f.write('\n')
```

Step 1:
SVD++ algorithm
(n_factors=100, n_epoch=50)

Step 2:
SVD++ algorithm
(n_factors=100, n_epoch=100)

## PART III : 추천 시스템

### Requirement 3-3

```
# TODO: 3-4-5. Best Model
best_algo_mf = surprise.SVDpp(n_factors=200, n_epochs=200)
```

평가 지표: RMSE(Root mean square error)

| (n_factors,n_epochs)/알고리즘 | SVD | SVD++ | NMF |
|---|---|---|---|
| (100, 50) | 1.621 | 1.021 | 1.021 |
| (200, 100) | 1.593 | 1.081 | 1.081 |
| (200, 200) | 1.426 | 1.057 | 1.057 |

Best algorithm

# PROJECT #2

Team 03

감사합니다