DMA P#2

DB mining & Automated Recommendation System

2024-1 데이터 관리와 분석
9조 강명훈 고우성 김원재 김영헌

INDEX

GOAL

✓ 프로젝트 #1의 방법으로 DB를 구축하고 이를 이용하여 DB mining 및 Automated Recommendation System 구현을 목적으로 한다.

PART 1

사이트 A에서 BEST restaurant으로 선정한 식장 기준에 대한 의사결정 나무를 만드는 것을 목표로 한다.

PART 2

사이트 A의 식당들 간의 연관분석을 목표로 한다.

PART 3

사용자들에게 제품을 추천하는 추천 시스템 구현을 목표로 한다.

PART 1

사이트 A의 BEST restaurant 선정기준

```
cursor.execute("SHOW COLUMNS FROM Restaurant LIKE 'best_restaurant';")
result = cursor.fetchone()
if result:
    cursor.execute("ALTER TABLE Restaurant DROP COLUMN best_restaurant;")
cursor.execute("SHOW COLUMNS FROM Restaurant LIKE 'best_restaurant_id';")
result = cursor.fetchone()
if result:
    cursor.execute("ALTER TABLE Restaurant DROP COLUMN 
best_restaurant_id;")

cursor.execute("ALTER TABLE Restaurant ADD best_restaurant TINYINT(1) 
DEFAULT 0;")

cursor.execute("ALTER TABLE Restaurant ADD best_restaurant_id
```

```
cursor.execute("DROP TABLE IF EXISTS temp_best_restaurant;")
cursor.execute("CREATE TABLE temp_best_restaurant (temp_id VARCHAR(255));")
cursor.execute(
    "LOAD DATA LOCAL INFILE 
'C:/Users/user/Desktop/DMA_project2/dataset/best_restaurant.txt' INTO TABLE 
temp_best_restaurant;")
cursor.execute("UPDATE temp_best_restaurant SET temp_id = 
left(temp_id,22);")
cursor.execute('''UPDATE Restaurant R, temp_best_restaurant T
               SET R.best_restaurant_id = temp_id
               WHERE LOWER(left(R.restaurant_id,22)) = 
LOWER(T.temp_id);''')
cursor.execute(
    '''UPDATE Restaurant SET best_restaurant = 1 WHERE best_restaurant_id = 
restaurant_id;''')
cnx.commit()
```

```
cursor.execute("ALTER TABLE review MODIFY COLUMN taste_score 
DECIMAL(11,1);")
cursor.execute("ALTER TABLE review MODIFY COLUMN service_score 
DECIMAL(11,1);")
cursor.execute("ALTER TABLE review MODIFY COLUMN mood_score 
DECIMAL(11,1);")

cursor.execute("""
    UPDATE Review
    SET taste_score = total_score,
        service_score = total_score,
        mood_score = total_score
    WHERE taste_score = 0 AND service_score = 0 AND mood_score = 0 AND 
total_score != 0;
""")
cnx.commit()
```

- ✓ Restaurant table에 best_restauran라는 새로운 column 추가, bes_ restauran의 데이터 타입은 TINYINT(1)이고 default 값은 0이다.
- ✓ 식당의 id가 best_restaurant에 포함되면 best_restaurant column에 1을 저장
- ✓ Review table에서 사용자가 세부점수를 입력하지 않고 종합점수를 입력한 경우 모든 세부점수를 종합점수와 같은 값으로 대체한다

사이트 A의 BEST restaurant 선정기준

- restaurant_id: 식당의 id
- best_restaurant: 식당의 BEST restaurant 선정여부
- avg_total_score: 식당이 사용자들에게 받은 종합 점수의 평균
- avg_taste_score: 식당이 사용자들에게 받은 맛 점수의 평균
- avg_service_score: 식당이 사용자들에게 받은 서비스 점수의 평균
- avg_mood_score: 식당이 사용자들에게 받은 분위기 점수의 평균
- num_of_reviews: 식당에 달린 리뷰의 수
- num_of_collections: 식당을 컬렉션에 포함시킨 사용자의 수
- category_name: 제품이 속한 카테고리 이름

```
cursor.execute('''SELECT
    r.restaurant_id,
    r.best_restaurant,
    AVG(re.total_score) AS avg_total_score,
    AVG(re.taste_score) AS avg_taste_score,
    AVG(re.service_score) AS avg_service_score,
    AVG(re.mood_score) AS avg_mood_score,
    COUNT(re.restaurant) AS num_of_reviews,
    (SELECT COUNT(c.user_id) FROM Collection c WHERE c.restaurant_id =
r.restaurant_id) AS num_of_collections,
    ca.name AS category_name
FROM
    Restaurant AS r
LEFT JOIN
    Review AS re ON r.restaurant_id = re.restaurant
LEFT JOIN
    Category AS ca ON r.category = ca.category_id
GROUP BY
    r.restaurant_id;
            ''')
    data = cursor.fetchall()
    print(data)
    with open("C:/Users/user/Desktop/DMA_project2_team09_part1.csv", 'w',
encoding='utf-8-sig', newline='') as f_handle:
        writer = csv.writer(f_handle)
        writer.writerow(
        ['restaurant_id', 'best_restaurant', 'avg_total_score',
'avg_taste_score', 'avg_service_score', 'avg_mood_score',
             'num_of_reviews', 'num_of_collections', 'category_name'])
        for row in data:
          writer.writerow(row)
```

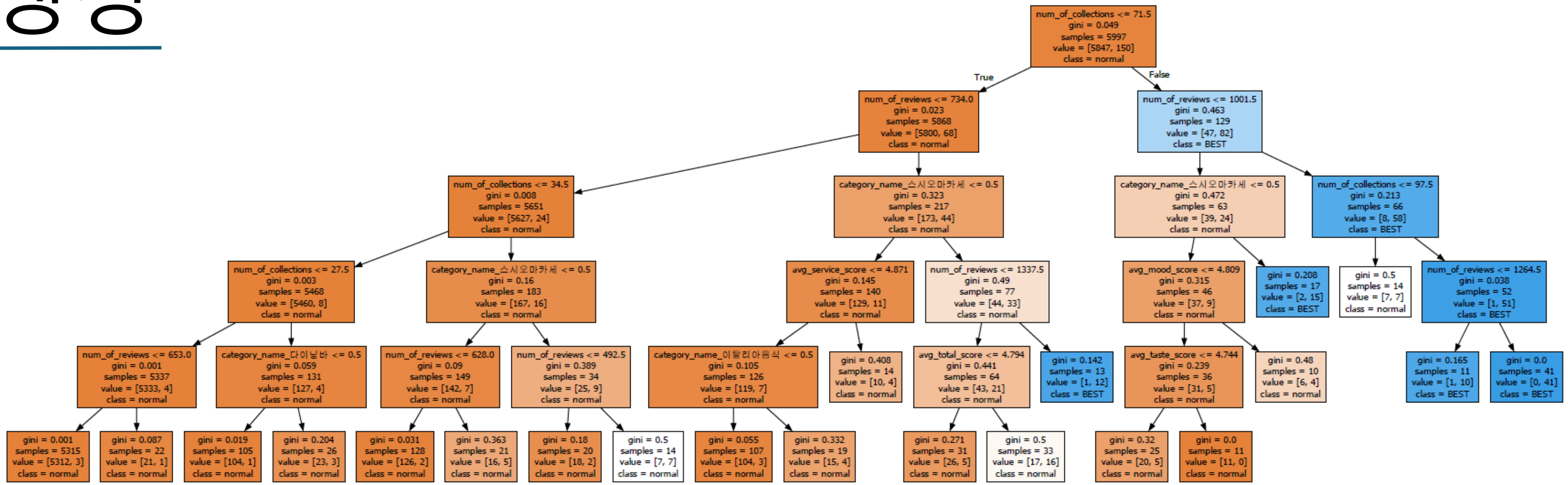
의사결정나무 생성

```python
# 의사결정나무 모델 생성 및 학습 make gini tree
features = df.drop(['best_restaurant', 'restaurant_id'], axis=1)
classes = df['best_restaurant']
DT_gini = tree.DecisionTreeClassifier(criterion='gini',
min_samples_leaf=10, max_depth=5)
DT_gini.fit(X=features, y=classes)
print(DT_gini.get_params())
print('-----------------------------------------------')
# make entropy tree
DT_entropy = tree.DecisionTreeClassifier(criterion='entropy',
min_samples_leaf=10, max_depth=5)
DT_entropy.fit(X=features, y=classes)
print(DT_entropy.get_params())
print('-----------------------------------------------')
feature_names = list(features.columns)
output_directory = "C:/Users/user/Desktop"
# DOT 파일 경로 설정
dot_file_path_gini = os.path.join(output_directory,
'DMA_project2_team09_part1_gini.dot')
# DOT 파일로 gini 기준 의사결정나무 시각화 저장
export_graphviz(DT_gini, out_file=dot_file_path_gini,
feature_names=features.columns,
              class_names=['normal', 'BEST'], filled=True)
# DOT 파일로 Entropy 기준 의사결정나무 시각화 저장
dot_file_path_entropy = os.path.join(output_directory,
'DMA_project2_team09_part1_entropy.dot')
dot_data_entropy = export_graphviz(DT_entropy,
out_file=dot_file_path_entropy, feature_names=features.columns,
                                  class_names=['normal', 'BEST'], filled=True)
```

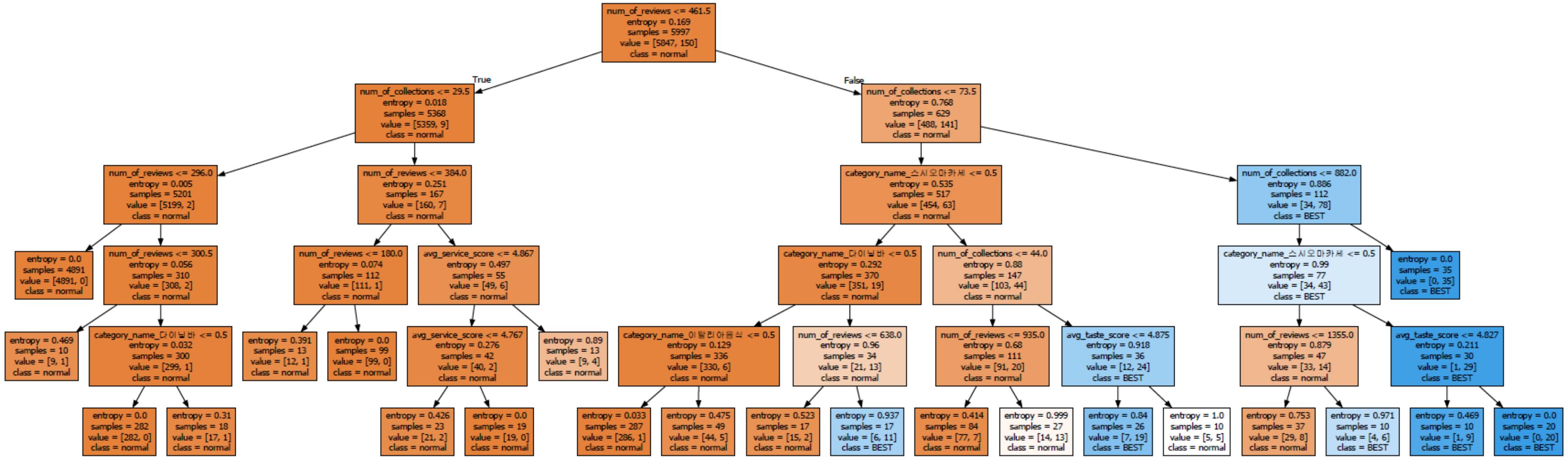
```python
dot_path = dot_file_path_gini
with open(dot_path, 'r', encoding='utf-8') as file:
    dot_content = file.read()
# Graphviz 객체 생성
dot_graph = graphviz.Source(dot_content)
pdf_file_path_gini = os.path.join("C:/Users/user/Desktop",
'DMA_project2_team09_part1_gini')
# PDF로 변환 및 저장
output_pdf_path = pdf_file_path_gini
dot_graph.render(filename=output_pdf_path, format='pdf', cleanup=True)

dot_path = dot_file_path_entropy
with open(dot_path, 'r', encoding='utf-8') as file:
    dot_content = file.read()
# Graphviz 객체 생성
dot_graph = graphviz.Source(dot_content)
pdf_file_path_entropy = os.path.join("C:/Users/user/Desktop",
'DMA_project2_team09_part1_entropy')
# PDF로 변환 및 저장
output_pdf_path = pdf_file_path_entropy
dot_graph.render(filename=output_pdf_path, format='pdf', cleanup=True)
```

의사결정나무 생성

gini tree

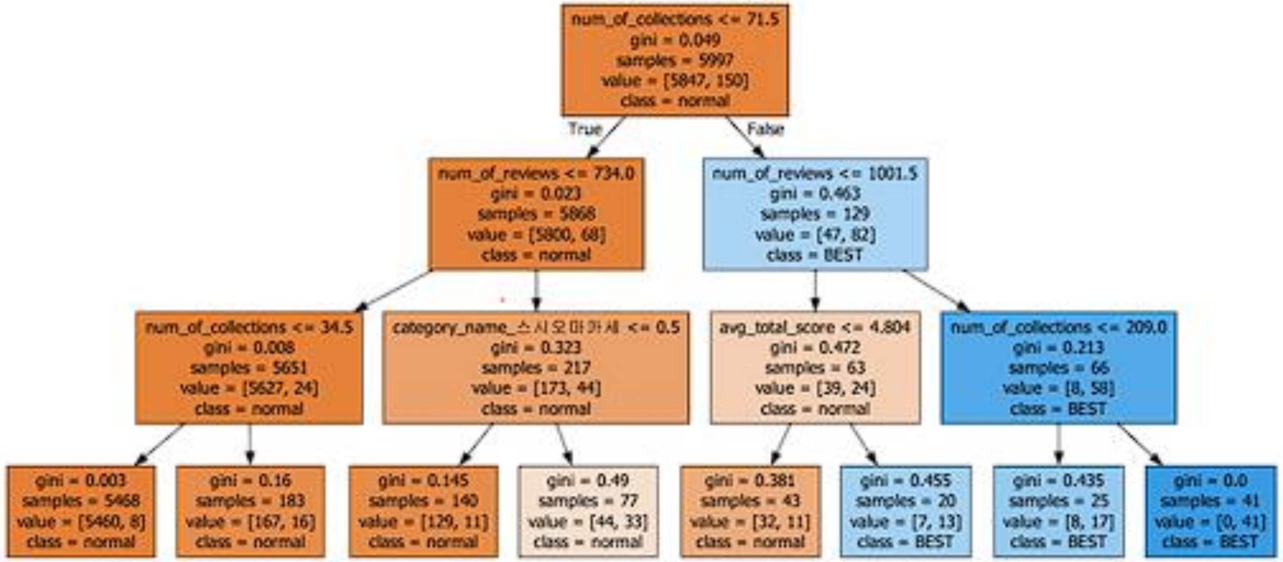


entropy tree



의사결정나무 생성

```
#using df from 1-3
features = df.drop(['best_restaurant',
'restaurant_id', 'avg_service_score',
'avg_mood_score'], axis=1)
classes = df['best_restaurant']
DT_gini =
tree.DecisionTreeClassifier(criterion='gini',
min_samples_leaf=20, max_depth=3)
DT_gini.fit(X=features, y=classes)
print(DT_gini.get_params())
print('-----------------------------------------
-')

dot_path_gini = pdf_export_font_change(DT_gini,
'DMA_project2_team09_part1_gini_modified',
features.columns)
output_path_gini =
"./DMA_project2_team09_part1_gini_modified"
export_pdf_from_dot(dot_path_gini, output_path_gini)
cursor.close()
```



PART 2

연관분석을 위한 VIEW 생성

```
CREATE VIEW restaurant_score AS
                    SELECT
                     RES.restaurant_id AS restaurant_id,
                     RES.restaurant_name AS restaurant_name,
                     COALESCE(COL.num_collection,0) AS num_collection,
                     COALESCE(REV.num_review,0) AS num_review,
                     COALESCE(CAT.num_category_restaurant,0) AS num_category_restaurant,
                     COALESCE(SUM(REVRE.count_revisit - 1),0) AS num_revisit,
                     COALESCE(SUM(FOL.num_follower*REVRE.count_revisit), 0) AS influential_review_score,
                      (COALESCE(COL.num_collection,0)*5+COALESCE(REV.num_review,0)+COALESCE(CAT.num_categ
ory_restaurant,0)+COALESCE(SUM(FOL.num_follower*REVRE.count_revisit),0)+COALESCE(SUM(REVRE.count_revis
it - 1),0)*5) AS score
```

- ✓ *Score : num_collection * 5 + num_review + num_category_restaurant + num_revisit * 5 + influential_review_score * 5*
- ✓ *num_revisit 의 예시는 다음과 같다 식당X 에 사용자A 가 5 개의 리뷰 사용자B 가 3 개의 리뷰, 사용자C 가 1 개의 리뷰를 남긴 경우 식당X 의 num_revisit 은 (5 - 1) + (3 - 1) = 6 과 같이 계산한다.*
- ✓ *Influential_review_score 의 예시는 다음과 같다 식당X 에 팔로워가 100 명인 사용자A 가 5 개의 리뷰를 남기고 팔로워가 2 명인 사용자B 가 2 개의 리뷰를 남긴 경우, 식당X 의 influential_review_score 는 5*100 = 500 과 같이 계산한다*

```
          FROM   Restaurant AS RES
                    LEFT JOIN
                            (SELECT restaurant_id, COUNT(*) AS num_collection FROM Collection GROUP BY
restaurant_id) AS COL ON RES.restaurant_id = COL.restaurant_id
                    LEFT JOIN
                         (SELECT restaurant, COUNT(*) AS num_review FROM Review GROUP BY restaurant) AS
REV ON RES.restaurant_id = REV.restaurant
                    JOIN
                         (SELECT category, COUNT(*) AS num_category_restaurant FROM Restaurant GROUP BY
category) AS CAT ON RES.category = CAT.category
                    LEFT JOIN
                           (SELECT restaurant, user_id, COUNT(*) AS count_revisit FROM Review GROUP BY
restaurant, user_id) AS REVRE ON RES.restaurant_id = REVRE.restaurant
                    LEFT JOIN
                        (SELECT followee_id, COUNT(*) AS num_follower FROM Follow GROUP BY followee_id)
AS FOL ON REVRE.user_id = FOL.followee_id AND num_follower>=3
                    GROUP BY
                        RES.restaurant_id
                    ORDER BY
                        score DESC

                   LIMIT 300;
```

연관분석을 위한 VIEW 생성

```
CREATE VIEW user_restaurant_IntDegree AS
                   SELECT
                       U.user_id AS user,
                       R.restaurant_id AS restaurant,
                       FLOOR(R4.res_avg/R5.user_avg+T1.col_num+G1.rev_num) AS IntDegree
                       FROM
                           Restaurant AS R
                       CROSS join
                           User as U
                       JOIN
                             (select G.user_id, G.category, COUNT(distinct G.restaurant_id) AS rev_num
FROM (select RE.user_id, R3.restaurant_id, R3.category FROM Review AS RE JOIN Restaurant AS R3 on
R3.restaurant_id=RE.restaurant) AS G group by G.user_id, G.category HAVING rev_num>0) AS G1 ON
G1.user_id=U.user_id AND G1.category=R.category
                       JOIN
                                   (select T.user_id, T.category, COUNT(distinct T.restaurant_id) AS
col_num FROM (select COL.user_id, R2.restaurant_id, R2.category FROM Collection AS COL JOIN Restaurant
AS R2 on R2.restaurant_id=COL.restaurant_id ) AS T group by T.user_id, T.category) AS T1 ON
T1.category=R.category AND T1.user_id=U.user_id

                       JOIN
                           (SELECT restaurant,user_id, AVG(total_score) AS res_avg FROM Review GROUP BY
restaurant,user_id) AS R4 ON R4.restaurant=R.restaurant_id AND R4.user_id = U.user_id
                       JOIN
                           (SELECT user_id, AVG(total_score) AS user_avg FROM Review GROUP BY user_id)
AS R5 ON R5.user_id=U.user_id
                       JOIN
                                             (select restaurant_id FROM restaurant_score) AS RS ON
RS.restaurant_id=R.restaurant_id;
```

✓ *[IntDegree Rating Equation]*

*IntDegree(user, restaurant) = 5 * (restaurant에 user가 남긴 리뷰 total_score의 평균) / (user가 남긴 모든 리뷰 total_score의 평균) + (restaurant의 category에 속하는 식당 중에서 user가 리뷰를 남긴 식당의 수) + (restaurant의 category에 속하는 식당 중에서 user가 collection에 포함시킨 식당의 수)*

✓ *아래의 column 들을 포함하는 user_restaurant_IntDegree 라는 이름을 가지는 view 를 생성하라.*

- user: *사용자의 id*
- restaurant: *식당의 id (상위 300 개 중 하나)*
- IntDegree: *위에서 정의한 사용자 가 식당 에 가지는 관심 정도*

연관분석을 위한 data 생성

```
sql1 = ''
for i in restaurant_id:
    sql = 'max(if(restaurant=\'{id_1}\', 1, 0)) as \'{id_2}\''.format(id_1=i, id_2=i)
    sql1 = sql1 + ', ' + sql
print('aaaaaaaaaa')

cursor.execute('''select user
    {} from (select user, restaurant from partial_user_restaurant_IntDegree) as a
    group by user;
    '''.format(sql1))
hor_view = pd.DataFrame(cursor.fetchall())
hor_view.columns = cursor.column_names
hor_view = hor_view.set_index('user')
hor_view.to_pickle('DMA_project2_team%02d_part2_horizontal.pkl' % team
```

```
hor_view = hor_view.replace(0, False)
hor_view = hor_view.replace(1, True)

frequent_itemsets = apriori(hor_view, min_support=0.2, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=2)
rules.to_pickle('DMA_project2_team%02d_part2_association.pkl' % team)
```

- ✓ vertical data를 horizontal data로 변환
- ✓ 0,1을 bool type으로 변환
- ✓ apriori를 사용해 0.2 이상의 min support를 갖는 frequent itemset
- ✓ 2이상의 lift 값을 갖는 rule 생성

연관분석 실행, 결과 저장

```
Total number of association rules: 4211578

Top 10 rules sorted by lift:
                                         antecedents  \
2105789  (V0SNE_bpiuk-rbERvpmqbA, euYxK_Z-DuVrEl74arXqk...
2683773  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683760  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683761  (V0SNE_bpiuk-rbERvpmqbA, rRBP1l5Hq-850-9kCVZC4...
2683762  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683763  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683764  (V0SNE_bpiuk-rbERvpmqbA, euYxK_Z-DuVrEl74arXqk...
2683766  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683768  (V0SNE_bpiuk-rbERvpmqbA, 7aORNMXw9Kwl_pshlxb01...
2683769  (V0SNE_bpiuk-rbERvpmqbA, rRBP1l5Hq-850-9kCVZC4...

                                         consequents  \
2105789  (A9Bahp1qqbHF1LaxVEuuOA, -KfR4WkBLCdy0ISb2Lf-k...
2683773   (pg2-pj_qzjw4LsyZCiGBtw, cqbUQADGJCByLzJCX9eDHA)
2683760   (cqbUQADGJCByLzJCX9eDHA, HYnTvmpQyWRQoZ5jNoUuKA)
2683761   (7aORNMXw9Kwl_pshlxb01w, euYxK_Z-DuVrEl74arXqkg)
2683762   (vmbRKLQMlWYcD2-5lketpA, euYxK_Z-DuVrEl74arXqkg)
2683763   (HYnTvmpQyWRQoZ5jNoUuKA, euYxK_Z-DuVrEl74arXqkg)
2683764   (7aORNMXw9Kwl_pshlxb01w, rRBP1l5Hq-850-9kCVZC4Q)
2683766   (vmbRKLQMlWYcD2-5lketpA, rRBP1l5Hq-850-9kCVZC4Q)
2683768   (rRBP1l5Hq-850-9kCVZC4Q, HYnTvmpQyWRQoZ5jNoUuKA)
2683769   (pg2-pj_qzjw4LsyZCiGBtw, 7aORNMXw9Kwl_pshlxb01w)
```

```
         antecedent support  consequent support   support  confidence  lift  \
2105789            0.285714            0.285714  0.285714         1.0   3.5
2683773            0.285714            0.285714  0.285714         1.0   3.5
2683760            0.285714            0.285714  0.285714         1.0   3.5
2683761            0.285714            0.285714  0.285714         1.0   3.5
2683762            0.285714            0.285714  0.285714         1.0   3.5
2683763            0.285714            0.285714  0.285714         1.0   3.5
2683764            0.285714            0.285714  0.285714         1.0   3.5
2683766            0.285714            0.285714  0.285714         1.0   3.5
2683768            0.285714            0.285714  0.285714         1.0   3.5
2683769            0.285714            0.285714  0.285714         1.0   3.5

         leverage  conviction  zhangs_metric
2105789  0.204082         inf            1.0
2683773  0.204082         inf            1.0
2683760  0.204082         inf            1.0
2683761  0.204082         inf            1.0
2683762  0.204082         inf            1.0
2683763  0.204082         inf            1.0
2683764  0.204082         inf            1.0
2683766  0.204082         inf            1.0
2683768  0.204082         inf            1.0
2683769  0.204082         inf            1.0

Quantitative Analysis:
            support    confidence          lift


count  4.211578e+06  4.211578e+06  4.211578e+06
mean   2.860999e-01  9.451540e-01  3.112929e+00
std    7.411988e-03  1.235877e-01  5.493258e-01
min    2.857143e-01  6.666667e-01  2.333333e+00
25%    2.857143e-01  1.000000e+00  2.333333e+00
50%    2.857143e-01  1.000000e+00  3.500000e+00
75%    2.857143e-01  1.000000e+00  3.500000e+00
max    4.285714e-01  1.000000e+00  3.500000e+00
```

PART 3

get_top_n

<User based>

```
testset_id = [x for x in testset if x[0] in id_list]
predictions = algo.test(testset_id)
for uid, rid, true_r, est, _ in predictions:
        results[uid].append((rid, est))
```

<Item based>

```
testset_id = [x for x in testset if x[1] in id_list]
predictions = algo.test(testset_id)
for uid, rid, true_r, est, _ in predictions:
        results[rid].append((uid, est))
```

<Rating Sorting>

```
for id_, ratings in results.items():
  ratings.sort(key=lambda x: x[1], reverse=True)
  results[id_] = ratings[:n]
return results
```

User-based Recommendation

✓ KNNBasic, cosine

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

2bZ4xxQGSIn_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 스시카나에 - 5.0	1. 스시우미 잠실 - 5.0	1. 청담 긴자블루 - 5.0	1. 디해방 D HAEBANG - 5.0	1. 스시우미 잠실 - 5.0
2. CESTA 세스타 - 5.0	2. 스시이로 - 5.0	2. 텐지몽 - 5.0	2. 레스토랑 주은 - 5.0	2. 스시이로 - 5.0
3. 스시잇센 - 5.0	3. 스시카나에 - 5.0	3. 볼래 - 5.0	3. Bogl(보글) - 4.7	3. 스시카나에 - 5.0
4. 야키토리스미카 - 5.0	4. 어물전 청 도산점 - 5.0	4. 복덕방막걸리집 - 5.0	4. 청담나인 NINE live - 4.5	4. 어물전 청 도산점 - 5.0
5. 에노테카오토 - 5.0	5. 비스트로 앤트로 - 5.0	5. 스이세한남 - 4.85	5. 스시코우지 - 4.3	5. CESTA 세스타 - 5.0

✓ KNNWithMeans, Pearson

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

2bZ4xxQGSIn_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 스시츠바사 - 3.06	1. 볼래 - 5.0	1. SOIGNE - 4.03	1. 고든램지버거 롯데월드몰점 - 2.93	1. 스시소라 서초점 - 4.88
2. 고든램지버거 롯데월드몰점 - 2.935	2. 타쿠미곤 - 3.94	2. 타쿠미곤 - 3.88	2. 스시소라 서초점 - 2.93	2. 어물전 청 도산점 - 4.88
3. 스시소라 서초점 - 2.935	3. 조우 鯛牛 - 3.79	3. 조우 鯛牛 - 3.73	3. 키친마이야르 - 2.93	3. 술수 - 4.83
4. 키친마이야르 - 2.935	4. 스키야키 타카 - 3.39	4. 스키야키 타카 - 3.33	4. 로우앤슬로우 - 2.93	4. 비놀로지 VINOLOGY - 4.67
5. 로우앤슬로우 - 2.935	5. 류니끄 - 3.39	5. 류니끄 - 3.33	5. 웨스틴조선서울 아리아 - 2.93	5. Bogl(보글) - 4.61

Item-based Recommendation

✓ KNNBasic, cosine

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

로우앤슬로우	부베트 서울	고든램지버거 롯데월드몰점	산리오 러버스 클럽 홍대점	웨스틴조선서울 아리아
1. alW1KveHXCt2z1oNy9nUDw - 5.0	1. 2yvDE0WnTZ7MEmd2Q_wCYQ - 5.0	1. _S454QrntlCccP_XoANqzA - 5.0	1. LwuhmgftG5NoNcUcWKPlBg - 4.0	1. 4DkuO4_frlzcmU_VK41fqw - 5.0
2. A9iFWYH-hMAZQTZ5gtwaBA - 5.0	2. nSCDQA-oVwtZSYi_IREnWQ - 5.0	2. X3DQfitXuo94UqpJcK8vpA - 5.0	2. 4IQYM2yJqwwElfKPUUSkvQ - 3.0	2. Ap3sazzjG3qGt6SvoElAVA - 4.7
3. AAysJBAlbbyARMRrON-91A - 5.0	3. 9cnPhusg4PPG98EEhTJ7Rw - 4.7	3. Yfdk6FzjuVpeiTS6a6FlvA - 5.0	3. YaV2v2AZhk0XWhrKM_yjmw - 3.0	3. OZhnXLO54X9wTChUUvOzGw - 4.3
4. PoHc5SwtNhAwU0polpZOQQ - 4.8	4. 3mSRd11Gy6foHtqbOeSguw - 4.5	4. qK1AsQhPTY4ocMLsAoXaUw - 5.0	4. TcHyEOEmA46ZVkzn7DpFMw - 2.85	4. HLvDS6NERG4r0iC3QTukkQ - 4.3
5. XqpcP9UzsDY4FKGsVmuMVg - 4.8	5. G3a_e0yePxy67Ez45ruwAw - 4.5	5. 9BDOmPx3ljy-rwHrWmam0A - 5.0	5. 36bimicGZbygpMoWSM6a8g - 2.85	5. Gjr8z8ZLfPM7y5HqD1sPyQ - 4.0

✓ KNNWithMeans, pearson

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

로우앤슬로우	부베트 서울	고든램지버거 롯데월드몰점	산리오 러버스 클럽 홍대점	웨스틴조선서울 아리아
1. TcHyEOEmA46ZVkzn7DpFMw - 2.20	1. TcHyEOEmA46ZVkzn7DpFMw – 2.74	1. gwTlk97lS-AfXMBDYQ9smg – 3.42	1. TcHyEOEmA46ZVkzn7DpFMw – 2.77	1. TcHyEOEmA46ZVkzn7DpFMw - 3.08
2. 36bimicGZbygpMoWSM6a8g - 2.20	2. 36bimicGZbygpMoWSM6a8g – 2.74	2. zHuK3t2wqbC-a1KulOyMgA – 2.51	2. 36bimicGZbygpMoWSM6a8g – 2.77	2. 36bimicGZbygpMoWSM6a8g - 3.08
3. e0RAkMXfq_g2gjg5QmBC5A - 2.20	3. e0RAkMXfq_g2gjg5QmBC5A – 2.74	3. DWvXMEO2bJAufwpYDbQf1A – 2.34	3. e0RAkMXfq_g2gjg5QmBC5A – 2.77	3. e0RAkMXfq_g2gjg5QmBC5A – 3.08
4. G5TCVuV9WdHB1WBMUWNe-Q - 2.20	4. G5TCVuV9WdHB1WBMUWNe-Q – 2.74	4.. _S454QrntlCccP_XoANqzA – 2.34	4.. G5TCVuV9WdHB1WBMUWNe-Q - 2.77	4.. G5TCVuV9WdHB1WBMUWNe-Q – 3.08
5. 4phUB87XOV4hBDelq_JRYg - 2.20	5. 4phUB87XOV4hBDelq_JRYg – 2.74	5. zefQV3ky_7oZBwIOQsHBfg – 2.34	5. 4phUB87XOV4hBDelq_JRYg -2.77	5. 4phUB87XOV4hBDelq_JRYg – 3.08

Cross Validation

✓ User-based

<사용한 알고리즘 및 파라미터 설정>

```
algorithms = [surprise.KNNBasic, surprise.KNNWithMeans,
surprise.KNNWithZScore, surprise.KNNBaseline]

sim_options = [{'name': 'msd','user_based': True}, {'name':
'cosine','user_based': True}, {'name': 'pearson','user_based': True}, {'name':
'pearson_baseline','user_based': True}]
np.random.seed(0)
kf = KFold(n_splits=5)
best_algo = None
best_sim_option = None
best_score = float('inf')
```

<반복문을 활용하여 최적 알고리즘 도출>

```
# Cross valiation to find the best algorithm and similarity option
for algo in algorithms:
  for sim_option in sim_options:
    algo_instance = algo(sim_options=sim_option)
    rmse_scores = []
    for trainset, testset in kf.split(data):
      algo_instance.fit(trainset)
      predictions = algo_instance.test(testset)
      rmse = surprise.accuracy.rmse(predictions, verbose=True)
      rmse_scores.append(rmse)
      mean_rmse = np.mean(rmse_scores)
    if mean_rmse < best_score:
      best_algo = algo
      best_sim_option = sim_option
      best_score = mean_rmse
```

✓ Item-based

<사용한 알고리즘 및 파라미터 설정>

```
algorithms = [surprise.KNNBasic, surprise.KNNWithMeans,
surprise.KNNWithZScore, surprise.KNNBaseline]
sim_options = [{'name': 'msd','user_based': False}, {'name':
'cosine','user_based': False}, {'name': 'pearson','user_based': False},
{'name': 'pearson_baseline','user_based': False}]
np.random.seed(0)
kf = KFold(n_splits=5)
best_algo_ib = None
best_sim_option_ib = None
best_score_ib = float('inf')
```

<반복문을 활용하여 최적 알고리즘 도출>

```
for algo in algorithms:
  for sim_option in sim_options:
    algo_instance = algo(sim_options=sim_option)
    rmse_scores = []
    for trainset, testset in kf.split(data):
      algo_instance.fit(trainset)
      predictions = algo_instance.test(testset)
      rmse = surprise.accuracy.rmse(predictions, verbose=True)
      rmse_scores.append(rmse)
      mean_rmse = np.mean(rmse_scores)
    if mean_rmse < best_score_ib:
      best_algo_ib = algo
      best_sim_option_ib = sim_option
      best_score_ib = mean_rmse
```

Matrix-based Recommendation

```python
# 알고리즘 목록을 정의합니다.
algorithms = [surprise.SVD, surprise.SVDpp, surprise.NMF]

# 매개변수 그리드를 정의합니다.
param_grid_svd = {'n_factors': [100, 150, 200, 250, 300], 'n_epochs': [50,
100, 150, 200], 'biased': [True, False]}
param_grid_svdpp = {'n_factors': [100, 150, 200, 250, 300], 'n_epochs': [50,
100, 150, 200]}
param_grid_nmf = {'n_factors': [100, 150, 200, 250, 300], 'n_epochs': [50,
100, 150, 200]}
best_score_mf = float('inf')
best_algo_mf = None
best_param = None
```

```python
# SVD 에 대해 그리드 검색을 수행합니다.
gs_svd = GridSearchCV(surprise.SVD, param_grid_svd, measures=['rmse'], cv=5)
gs_svd.fit(data)
  if gs_svd.best_score['rmse'] < best_score_mf:
  best_algo_mf = surprise.SVD
  best_param = gs_svd.best_params['rmse']
  best_score_mf = gs_svd.best_score['rmse']

# SVDpp 에 대해 그리드 검색을 수행합니다.
gs_svdpp = GridSearchCV(surprise.SVDpp, param_grid_svdpp, measures=['rmse'],
cv=5)
gs_svdpp.fit(data)
  if gs_svdpp.best_score['rmse'] < best_score_mf:
  best_algo_mf = surprise.SVDpp
  best_param = gs_svdpp.best_params['rmse']
  best_score_mf = gs_svdpp.best_score['rmse']

# NMF 에 대해 그리드 검색을 수행합니다.
gs_nmf = GridSearchCV(surprise.NMF, param_grid_nmf, measures=['rmse'], cv=5)
gs_nmf.fit(data)
  if gs_nmf.best_score['rmse'] < best_score_mf:
  best_algo_mf = surprise.NMF
  best_param = gs_nmf.best_params['rmse']
  best_score_mf = gs_nmf.best_score['rmse']
```

DMAP#2

THANK YOU

2024-1 데이터 관리와 분석
9조 강명훈 고우성 김원재 김영헌