DB Mining and Recommendation Using Python

팀: 9조

이름: 강명훈 고우성 김원재 김영헌

Part 1. DB MINING: DECISION TREE

- part1() 코드가 작동되지 않을 시 주의사항 :

windows 기준, 코드 실행 전에, C:₩ProgramData₩MySQL₩MySQL Server 8.0₩my.ini 파일을 열어서, [mysqld] 아래 칸에 local_infile=1 을 입력 후 저장합니다. 또한 mysql workbench 및 터미널 콘솔에서 mysql 로그인 후 다음 쿼리를 실행합니다 mysql> set global local_infile=1;
그 후 services.msc에서 MySQL80을 재시작합니다. 또한 이 파이썬 코드와 같은 위치에 best_restaurant.txt가 포함된 dataset 폴더가 존재해야 합니다. 그래도 작동하지 않을 시 아래 cnx = mysql.connector.connect(host=HOST, user=USER, password=PASSWORD, port = 3306, allow_local_infile=True) 에서 allow_local_infile=True 을 추가.
(R1-1)

DB construction을 통해 DB 설계 후 Requirement에 만족하는 Table을 만들어 restaurant으로 선정한 식당 기준에 해당하는 의사 결정나무를 만들었다.
Requirement 1에서 Restaurant 테이블에 best_restaurant이라는 새로운 column 추가, best_restaurant의 데이터 타입은 TINYINT(1)이며 default 값은 0 조건을 만족하기 위해 아래와 같은 식으로 코드를 작성하였다.

```
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
```

Best_restaurant와 best_restaurant_id라는 칼럼이 Restaurant table에 존재하면 drop 하고 존재하지 않으면 best_restaurant와 best_restaurnat_id를 쿼리문을 통해 각각 TINYINT(1) DEFAULT 0과 VARCHAR(255)의 조건을 가지도록 각각 만들어주었다.

식당의 id가 best_restaurant에 포함되면 best_restuarnat column에 1을 저장하기 위해서,
아래와 같은 코드를 사용하였다.

```
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 '.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()
```

Temp_best_restaurant라는 table이 있을 경우 drop하고, 없을 경우 temp_id VARCHAR(255)인 칼럼을 생성하여 bestrestaurant.txt파일을 해당 칼럼에 INFILE하였다. 그 후 best_restaurant_id와 temp_id가 완전히 일치하는 text를 찾기 위해 마지막에 띄어쓰기가 한칸있는 best_restaurnat_id는 left(,22)를 활용하여 left(,22)를 마찬가지로 활용한 temp_id 칼럼과 같은 칼럼을 찾아 best_restaurant_id에 저장하였다. 그후 best_restaurnat_id와 restaurant_id가 일치하면 best_restaurant에 1을 저장하였다.

Review 테이블상에서 사용자가 세부점수(맛, 서비스, 분위기)를 입력하지 않고 종합점수를 직접 입력한 경우 (맛, 서비스, 분위기 점수가 모두 0이지만 종합 점수는 0이 아닌 경우)는 세부점수 값을 모두 종 합점수와 같은 값으로 대체하는 query를 작성하고, 조건을 만족하기 위해 아래와 같은 코드를 작성하였다.

```
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()
```

taste_score, service_score, mood_score가 decimal(11)로 저장되어 소수점이하의 숫자가 저장되지 않기 때문에 Decimal(11,1)을 활용하여 column을 modify 했다. 이를 통해 total_score가 4.5이고 세부점수는 0일 때 세부점수를 4가 아닌 4.5로 정확히 저장될 수 있게 만들었다. 다음으로 Review 테이블을 업데이트해서 total score가 0이아니고 taste_score, service_score, mood_score가 0이 아니면 total score값으로 세부점수로 바꿔주는 쿼리를 만들었다.
이를 통해 requirement 1을 만족하는 코드를 만들었다.

(R1-2)

requirement 1-2를 만족하기 위해 nested query를 이용하여 하나의 SQL 문장으로 작성하여 결과를 DMA_project2_team09_part1.csv에 저장하는 코드를 작성하였다.

1. restaurant_id: 식당의 id
2. best_restaurant: 식당의 BEST restaurant 선정 여부

3. avg_total_score: 식당이 사용자들에게 받은 종합 점수의 평균
4. avg_taste_score: 식당이 사용자들에게 받은 맛 점수의 평균
5. avg_service_score: 식당이 사용자들에게 받은 서비스 점수의 평균
6. avg_mood_score: 식당이 사용자들에게 받은 분위기 점수의 평균
7. num_of_reviews: 식당에 달린 리뷰의 수
8. num_of_collections: 식당을 컬렉션에 포함시킨 사용자의 수
9. 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("./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)
```

위와 같은 코드를 통해 restaruant에서 restaurant_id, best_restaurant column을 select하고, review table에서 avg를 사용하여 total_score, taste_score, service_score, mood_score를 각각 avg_total_score, avg_taste_score, avg_service_score, avg_mood_score로 저장하였다. 다음으로 review테이블의 restaurant칼럼에서 count를 사용하여 num_of_review를 작성하였다. review table의 restaurant와 restaurant table의 restaurant_id가 같을때 left join을 사용하여 위의 reiview table에서 뽑은 column에 대한 정보를 restaurant table에 left join하였다. 또한 collection table의 restaurant id와 restaurant의 restaurant id가 같을 때 collection table의 user id개수를 세어 저장하여 각 restaurant이 몇 개의 사용자 콜렉션에 포함되어있는지를 저장하였다. 마지막으로 category에서

name column을 사용하여 restaurant table에 category_name 칼럼으로 저장하였다. category table의 category_id와 restaurant table의 category_id가 같으면 이때 category table의 이름을 저장하였다. 마지막으로 Group by r.restaurant_id를 활용해 레스토랑 ID별로 결과를 그룹화하며, 이를 통해 각 레스토랑별로 계산된 평균 점수와 리뷰 수, 콜렉션 수 등을 집계하였다.
이렇게 만들어진 sql query file을 cursor.fetchall()을 활용하여 data 변수에 저장하였고 open함수를 사용해 정해진 directory에 utf-8-sig로 저장하여 해당 파일 핸들을 f_handle 변수에 저장하였다. 그리고 csdv.writer(f_handle)을 사용하여 csv 파일에 쓸 write를 생성하여 writer.writerow함수를 사용하여 csv 파일 첫줄에 열 제목을 작성한다. 마지막으로 data에 저장된 각행을 반복하여 csv파일에 작성한다. 이는 for row in data를 통해 각 행에 접근하고 writer.writerow(row)를 통해 행 데이터를 파일에 쓴다. 해당 데이터는 "./DMA_project2_team09_part1.csv" 디렉토리에 저장되었다.

(R1-3)

Requirement3는 다음과 같다.

1. categorical data인 category_name은 one-hot encoding으로 처리한다
2. avg_total_score, avg_taste_score, avg_service_score, avg_mood_score의 결측치는 3으로 대체한다.
3. 사용 라이브러리: sklearn.tree.DecisionTreeClassifier
4. Node impurity criterion: gini / entropy
5. 결과 파일명: node impurity criterion에 따라 DMA_project2_team##_part1_gini.pdf, DMA_project2_team##_part1_entropy.pdf 로 구분함.
6. 분석 목표: BEST restaurant 선정 기준
7. min_samples_leaf: 10 max_depth: 5 feature names: restaurant_id, best_restaurant, avg_total_score, avg_taste_score, avg_service_score, avg_mood_score, num_of_review, num_of_collection, 그리고 category_name을 one-hot encoding으로 바꿔 생성한 각 column의 name
8. class names: normal, BEST

이 조건들을 만족하기 위해 아래와 같은 코드를 사용하였다.

```
df = pd.read_csv("./DMA_project2_team09_part1.csv")
df[['avg_total_score', 'avg_taste_score', 'avg_service_score',
'avg_mood_score']] = df[
    ['avg_total_score', 'avg_taste_score', 'avg_service_score',
'avg_mood_score']].fillna(3)

# OneHotEncoder 초기화 및 적용
encoder = OneHotEncoder(sparse_output=False)
```

```
encoded_categories = encoder.fit_transform(df[['category_name']])
category_columns = encoder.get_feature_names_out(['category_name'])
df_encoded = pd.DataFrame(encoded_categories, columns=category_columns)

df = pd.concat([df.drop('category_name', axis=1), df_encoded], axis=1)
```

먼저 얻은 csv 파일을 활용하여 avg_total_score, avg_taste_score, avg_service_score, avg_mood_score 의 결측치는 3으로 대체하는 코드를 작성하였다. 위와 같이 fillna(3)을 활용하였다. 다음으로 category name을 ONHOTENCODER을 사용하여 encoder.get_features_names_out (['category_name'])을 사용하여 .인코딩된 칼럼의 이름을 가져온다. 다음으로 pd.DataFrame (encoded_categories, columns=category_columns)를 사용하여 인코딩된 데이터를 Data Frame 형태로 나타낸다. 마지막으로 원본 DataFrame에서 category_name 칼럼을 제거하고 pd.concat를 사용하여 기존 DataFrame에서 삭제된 category_name칼럼을 제외한 데이터와 새로운 onehotencoding된 dataframe을 수평방향으로 병합한다.

```
# 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('-------------------------------------------------')
```

다음으로 best_restaurant와 restrestaurant_id를 df에서 drop해서 requirement를 만족하는 feature를 만들고 best_restaurant를 class를 만들어 featrue와 class를 만든다. 이후 의사결정나무 모델 생성 및 학습을 시작하는데 먼저 DT_gini를 tree.DecisionTreeClassifier를 활용해 gini criterion과 min_samples_leaf=10, max_dept=5를 활용해 의사결정나무모델을 만든다. DT_gini.fit(X=features, y=classes)로 데이터를 학습한다. 다음으로 Entropy 모델도 같은방식으로 조작하였다.

```
def pdf_export_font_change(decision_tree, file_name_prefix, feature_names):
    dot_file_path = f'./{file_name_prefix}.dot'
    export_graphviz(decision_tree, out_file=dot_file_path,
                    feature_names=feature_names, class_names=['normal',
'BEST'], filled=True)
    font_change_dot_file(dot_file_path)
    return dot_file_path

def font_change_dot_file(dot_path):
    modified_lines = []
    with open(dot_path, 'r', encoding='utf-8') as file:
        lines = file.readlines()

    for line in lines:
        if 'fontname=' in line:
            line = line.replace('fontname="helvetica"', 'fontname="Sans"')
        modified_lines.append(line)

    with open(dot_path, 'w', encoding='utf-8') as file:
```

```
        file.writelines(modified_lines)

def export_pdf_from_dot(dot_path, output_pdf_path):
    dot_graph = graphviz.Source.from_file(dot_path)
    dot_graph.render(filename=output_pdf_path, format='pdf', cleanup=True)

dot_path_gini = pdf_export_font_change(DT_gini,
'DMA_project2_team09_part1_gini', features.columns)
output_path_gini = "./DMA_project2_team09_part1_gini"
export_pdf_from_dot(dot_path_gini, output_path_gini)

dot_path_entropy = pdf_export_font_change(DT_entropy,
'DMA_project2_team09_part1_entropy', features.columns)
output_path_entropy = "./DMA_project2_team09_part1_entropy"
export_pdf_from_dot(dot_path_entropy, output_path_entropy)
dot_graph.render(filename=output_pdf_path, format='pdf', cleanup=True)
```

pdf_export_font_change함수와 font_change_dot_file 함수, export_pdf_from_dot 함수를 활용하여 dot file형태의 gini와 entropy 파일을 얻었고 font name을 Sans로 바꾸어 한글이 깨짐없게 dot 파일에 다시 저장하였다. 마지막으로 이를 pdf파일로 변환하여. "./DMA_project2_team09_part1_gini"와 "./DMA_project2_team09_part1_entropy"로 저장하였다. 해당 결과에서 얻을 수 있는 점은 다음과 같았다.

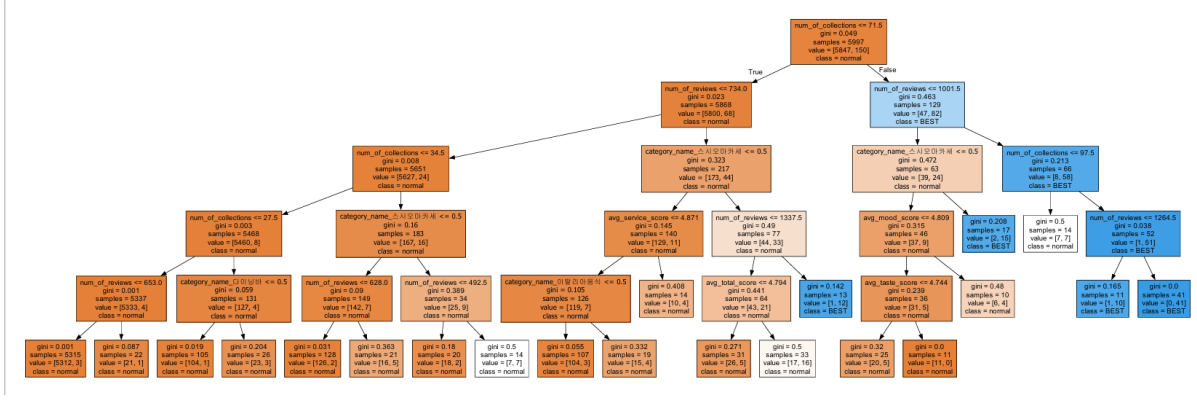


Fig.1. gini_pdf 파일

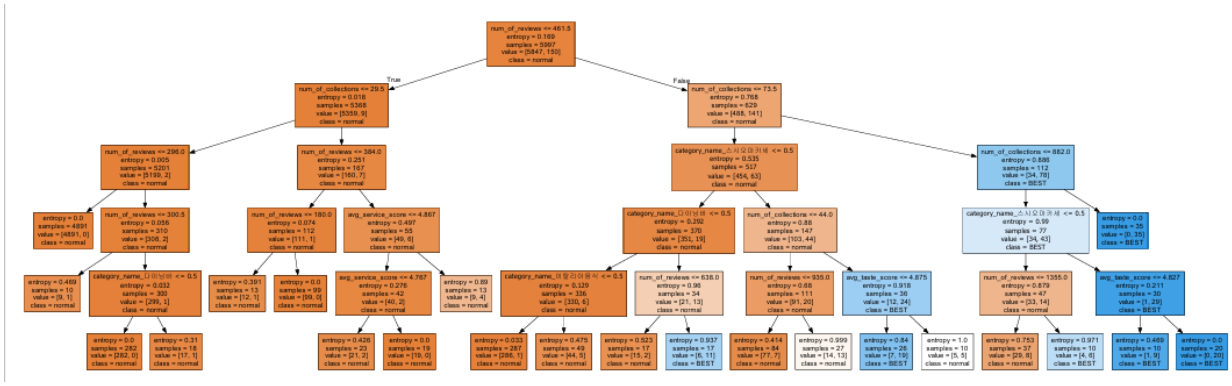


Fig.2. entropy_pdf 파일

두 의사결정나무(Gini와 Entropy)는 BEST 레스토랑을 식별하는 데 사용된 주요 변수와 분류 기준에 있어서 유사한 경향을 보인다. 먼저 Gini Tree에서 'num_of_collections`와 `num_of_reviews`는

매우 중요한 변수로, 많은 리뷰와 콜렉션 수를 가진 레스토랑이 BEST로 분류될 가능성이 높다. Collection이 97.5개이상 리뷰수가 10001개이상 이면 BEST로 분류되었고 리뷰수가 1264개를 넘으면 가장 pure하게 BEST였다. 이는 가장 오른쪽 node들의 파란색 best를 보면 확인할 수 있다. 또한 스시오마카세 레스토랑들 중 BEST로 분류될 확률이 높았다. 콜렉션 개수는 71.5보다 작고 리뷰수는 1337.5보다 높거나 콜렉션은 71.5보다 크고 리뷰수는 1001.5보다 작을때 스시오마카세 레스토랑으로 분류되면 BEST가 되었다. 이중 가장 다수의 sample을 가지고 pure한 것은 콜렉션 97.5개이상 리뷰수 1001개이상 일 때였다.

다음으로 entropy tree에서도 `num_of_collections`와 `num_of_reviews`가 결정적인 변수로 작용하며, 리뷰 수가 많은 레스토랑은 일반적으로 더 높은 등급을 받는 것으로 나타났다. Review수가 461.5개이상 collection수가 73.5개 이상일 때 BEST의 빈도가 훨씬 높고 pure했다. 특히 collection이 882개 이상이 되면 전부 BEST였다. 또한 해당 레스토랑중 스시오마카세 레스토랑이면 BEST일 확률이 높았다. 이때 avg_taste_score까지 좋으면 BEST였다. 가장 pure한것은 마찬가지로 collection 73.5개 review수가 882개이상인 restaurant이었다.

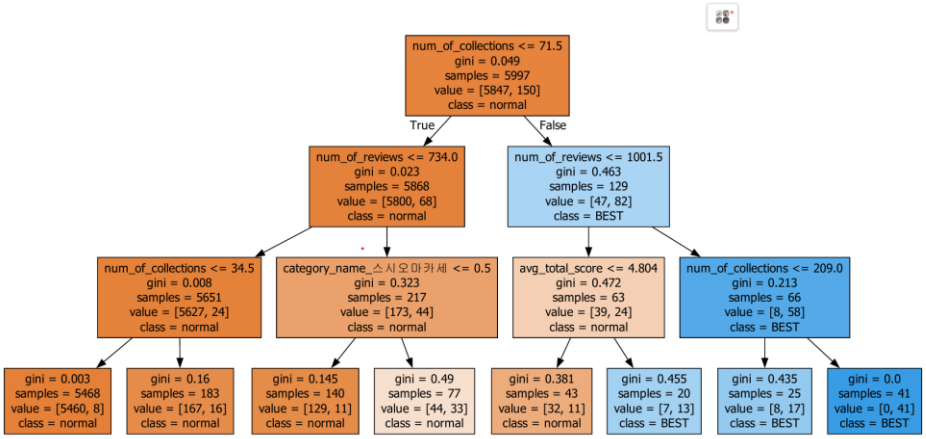
결론적으로 두 나무 모두에서 리뷰 수와 콜렉션 수가 높으면 BEST로 분류될 확률이 매우 높았다.

(R1-4)

```
#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()
```

Requirement 1-3의 code를 활용하여 작성하였다. 먼저 input features에 대해 requirement 1-3에서 criterion으로 사용되지 않은 avg_service_score와 avg_mood_score을 더 drop하였다. 또한 node impurity criterion은 gini를 사용하였고 sklearn.tree.DecisoinTReeclassifer에 min_samples_leaf를 기존의 10에서 20으로 증가하였고 max_depth또한 5에서 3으로 낮추어 사용하였다. Categorial data는 기존과 같은 one hot encoder를 사용하였다. 이는 범주간 서열이 없기 때문에 one hot encoding이 제일 적절하다고 판단하였다. 이를 활용하여 gini_modified를 pdf파일로 얻으면 다음과 같았다.



Gini modified 파일은 위와 같고 normal 클래스와 best 클래스로 분류되었다. 사용되지 않은 특성이 제거되었는데 이를 통해 모델이 간결해졌다. 이를 통해 중요한 특성에 더 집중할 수 있게 되어 특성 중요도가 명확해졌다. Max depth를 낮추어 모델의 깊이가 제한되어 트리의 복잡성이 감소되었다. 이로 인해 과적합의 위험이 줄어들었고 모델의 일반화 능력이 향상되었다. 이로 인해 num of collections와 num of reviews가 높으면 best로 선정됨을 더욱 명확히 알 수 있었다.

하지만 기존에 존재하였던 스시오마카세 점수가 높았을 때 best로 선정되는 규칙이 발견되지 않았다. 이는 구조 중 하나를 놓쳤다고 볼 수 있다.

다음으로 min_samples_leaf는 leaf_node에 필요한 .최소 샘플 수인데 이를 증가시킴으로 노드가 더 많은 데이터를 포함하게 되어 트리의 민감도가 감소한다. 이를 통해 데이터 소량 변동이나 노이즈에 대해 모델이 큰 영향을 받지 않아 과적합이 감소하고 모델 안정성이 향상된다. 하지만 마찬가지로 기존에 존재하였던 Best인 스시오마카세에 관련한 규칙을 놓쳤다.

이를 통해 Max depth가 5에서 3이되고 min_sample_leaf가 10에서 20이 되면 일반화 능력이 향상되어 num of collection과 num of review가 높으면 best로 선정됨을 명확히 알수있지만 스시오마카세에 관련한 규칙을 놓침이 확인되었다.

Part2. 연관 분석

(R2-1)

각 식당에 대한 컬렉션 수, 리뷰 수, 해당 식당의 카테고리에 속한 식당의 수, 사용자의 재방문 수, 영향력 있는 사용자가 작성한 리뷰 수를 기반으로 정의되는 점수를 계산하고 상위 300개의 식당 선정

```
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
```

해당 코드는 restaurant_score view를 선언하고 view에 들어갈 column들을 선언하는 코드이다. 프로젝트 조건을 따라 restaurant_id, restaurant_name 부터 score까지 선언하였다. 해당 코드에서 사용된 COALESCE 함수는 integer 혹은 decimal type의 column 값에 'NONE' 값이 있다면 이 값을 0으로 바꾸기 위해 사용하였다. Score를 계산할 때, 특정 column 값에 'NONE'이 존재한다면 다른 세부 점수들이 얼마나 큰지에 상관없이 score 값이 NULL값으로 정해진다. 이 문제를 해결하기 위해 식당 수, 리뷰 수, 재방문 리뷰 수 등의 값들에 모두 함수를 취해주었다.

Num_revisit과 influential_review_score는 다른 column값들과는 다르게 SUM을 취했는데 그 이유는 REVRE.count_revisit은 restaurant_id와 user_id로 group by되어있어, 식당 기준으로 점수를 계산할 땐 해당하는 모든 유저의 값들을 더해줘야 한다.

```
         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;
```

Restaurant 테이블에 여러 column들을 join시켜면서 view를 만들고자 했다. 첫번째로 해당 식당이

컬렉션에 추가된 횟수를 세기위해 restaurant_id를 기준으로 Collection 테이블을 join 시키면서 총 횟수를 count했다. 유사한 방식으로 리뷰에 추가된 횟수, 카테고리에 속하는 식당 수 등을 세어주었다. REVRE는 식당에 대한 전체 유저들의 재방문 리뷰 수를 세기위한 것인데 그 전에 특정 유저의 해당 식당에 대한 재방문 리뷰수를 계산하고 앞선 코드에서 본 것처럼 모든 유저에 대해 더해주었다. 영향력 있는 사용자를 판별하기위해 Follow 테이블에서 followee_id를 기준으로 count해주었고 그 값은 팔로워의 수가 된다. 팔로워 수가 3명 이상일 경우 REVRE, 특정 유저가 특정 식당에 남긴 리뷰 수,에 Left join하였다. 이후 앞선 코드에서 팔로워 수와 리뷰 수를 곱하여 영향력 점수를 계산할 수 있었다. 팔로워 수가 3명 미만인 경우, Fol.num_follower의 값이 NULL값이 되고 영향력점수는 0이 된다.
해당 column을 group by restaurant_id로 묶어주었고 order by score desc로 내림차순으로 정렬하였다. Limit 300으로 점수 상위 300개의 식당을 저장하였다.
구현 시에 JOIN을 사용하지 않고 LEFT JOIN을 주로 사용하였는데 식당에 리뷰가 없는 경우, 식당이 컬렉션에 추가되지 않은 경우 등이 존재한다면 JOIN의 경우 리뷰 또는 컬렉션 테이블에 레스토랑ID가 존재하지 않아 해당 레스토랑이 사라질 가능성이 존재한다. 따라서 LEFT JOIN을 사용해주었고 category는 모든 레스토랑이 가지고 있으므로 JOIN을 사용하였다.

(R2-2)

앞서 구현한 restaurant_score view의 300개 식당에 대해 연관분석을 위한 IntDegree값을 계산하여 user_restaurant_IntDegree view를 만들자.

```
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;
```

먼저 Restaurant 테이블과 User 테이블을 cross join해줌으로써 모든 restaurant, user 쌍을 만들었다. 유저가 어떤 레스토랑의 카테고리에 속하는 식당에 남긴 리뷰 수를 세기 위해 먼저 Review 테이블에 category를 join하여 Review 테이블에 category가 추가된 테이블을 만들었다. 그 리뷰 테이블의 user_id, category가 현재 user, category와 일치하면 중복되지 않는 restaurant_id를 count하였다. 이때 리뷰를 적어도 한 개 이상 남긴 경우를 세어주기 위해 rev_num>0이라는 조건을 추가하였다. 컬렉션에 남긴 리뷰 수도 동일한 메커니즘으로 count해주었다. 또 레스토랑에 남긴 유저의 평균 total score를 계산하기 위해 리뷰 테이블에서 레스토랑 id와 user id가 같은 경우 total score의 평균을 계산했고, 유저의 평균 total score를 계산하기 위해 user_id가 같은 경우를 계산하였다. 마지막으로 앞선 상위 300개의 레스토랑에 대한 정보만 남기기 위해 restaurant_score를 join시켜주었다.

```
CREATE VIEW partial_user_restaurant_IntDegree AS
                    SELECT
                        U.user AS user,
                        U.restaurant AS restaurant,
                        U.IntDegree AS IntDegree
                    FROM
                        user_restaurant_IntDegree AS U
                    JOIN
                         (SELECT user, COUNT(*) AS num FROM user_restaurant_IntDegree group by user) AS
UU on UU.user=U.user AND num>=25;
```

25개 이상의 정보를 가지고 있는 유저들에 대해 partial_user_restaurant_IntDegree view를 생성하였다. user_restaurant_IntDegree 뷰의 모든 유저들에 대해 count했을 경우 25가 넘는 유저들을 user_id기준으로 join시켰다. FROM의 user_restaurant_IntDegree에 user, restaurant, IntDegree의 값들이 저장되고 UU를 이용하여 IntDegree정보 25미만의 유저들을 탈락시켰다고 볼 수 있다.

```
    cursor.execute('select * from partial_user_restaurant_IntDegree;')
    pd_partial_user_restaurant_IntDegree = pd.DataFrame(cursor.fetchall())
    pd_partial_user_restaurant_IntDegree.columns = cursor.column_names
    pd_partial_user_restaurant_IntDegree.to_csv('DMA_project2_team%02d_part2_UAI.csv' % team, sep=',',
na_rep='NaN', index=False)
```

Pandas를 이용하여 excel파일로 partial_user_restaurant_IntDegree의 정보를 저장하였다.

(R2-3)

```
   restaurant_id = []
    cursor.execute('select restaurant from partial_user_restaurant_IntDegree;')
```

```
    while True:
        row = cursor.fetchone()
        if row == None:
            break
        if row[0] not in restaurant_id:
            restaurant_id.append(row[0])
```

Unique한 restaurant array를 만들기 위해 paritial_user_restaurant_IntDegree에서 모든 row에 대해 restaurant값을 읽고 for문을 이용하여 중복을 없애주었다. 물론 중복 restaurant은 존재하지 않을 것으로 보이나 확인을 위해 구현하였다.

```
    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
```

Horizontal data생성을 위해 sql1에 콤마를 기준으로 각 restaurant에 대해 user가 정보를 가지는지에 대한 내용을 저장하고, select문을 이용하여 horizontal data를 생성해준다.
그 결과를 pandas를 이용하여 pkl파일로 저장해준다.

(R2-4)

```
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)
```

효율적인 계산을 위해 0,1을 Bool type으로 변환해주었다. 그 후 apriori를 사용하여 min support 0.2 이상의 frequent_itemset을 생성하고, association_rules를 이용하여 2 이상 lift 값을 가지는 rule들을 생성한다. 그후 pkl파일로 저장한다.
생성된 pkl 파일로 정량적, 정성적 평가) – part2() 안에 코드를 삽입하였으나, part2() 함수 실행 시 약 2시간이 걸리므로, python 코드와 같은 폴더에 DMA_project2_team09_part2_association.pkl을 같이 두면, 코드 마지막 부분을 실행하여 좀 더 빠르게 결과를 확인할 수 있도록 하였다.

```
# part2 - data frame and data analysis (part2_association.pkl)
df = pd.read_pickle(f'DMA_project2_team{team_number:02d}_part2_association.pkl')
# Total number of association rules
print(f"Total number of association rules: {len(df)}")
```

```
# Top 10 rules sorted by lift
print("\nTop 10 rules sorted by lift:")
print(df.sort_values('lift', ascending=False).head(10))
# Quantitative Analysis
print("\nQuantitative Analysis:")
# Descriptive statistics of support, confidence, and lift
print(df[['support', 'confidence', 'lift']].describe())
```

<정량적 정성적 분석 결과>

```
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-85O-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-85O-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-85O-9kCVZC4Q)
2683766   (vmbRKLQMlWYcD2-5lketpA, rRBP1l5Hq-85O-9kCVZC4Q)
2683768   (rRBP1l5Hq-85O-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
```

Part3. 추천 시스템

(R3-1)

점수 예측 결과 top-n개 결과를 반환하는 get_top_n 함수를 작성

(1) 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))
```

이 경우 점수 예측 결과는 user id 기반으로 측정하므로, user id 를 key 로 하고 restaurant id 와 default training 을 튜플로 갖는 results list 를 만든다.

(2) Restaurant 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))
```

이 경우 점수 예측 결과는 restaurant id 기반으로 측정하므로, restaurant id 를 key 로 하고 user id 와 default training 을 튜플로 갖는 results list 를 만든다.

(3) Rating Sorting

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

result 결과를 top-n개만 반환해야 하므로 위와 같이 설정하였다.

(R3-2)

(1) R3-2-1 : KNNBasic, cosine

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

2bZ4xxQGSln_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. 불래 - 5.0	3. Bogl(보글) - 4.7	3. 스시카나에 - 5.0
4. 야키토리스미카 - 5.0	4. 어물전 청 도산점 - 5.0	4. 북덕방막걸리집 - 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

표 3.1. TOP-5 식당 (KNNBasic, cosine, user based), 소수점 둘째 자리까지 표현.

(2) R3-2-2 : KNNWithMeans, pearson

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

2bZ4xxQGSln_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 스시즈바사 - 3.06	1. 불래 - 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 朝牛 - 3.79	3. 조우 朝牛 - 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

표 3.2. TOP-5 식당 (KNNWithMeans, pearson, user based), 소수점 둘째 자리까지 표현.

(R3-2-3) 알고리즘별 성능 평가 비교 및 최적 모델 산정

<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
```

KNNBASIC, KNNWithMeans, KNNWithMeans, KNNWithZScore(각 유저의 z-점수 정규화를 이용하는 것), KNNBaseline, 각 거리를 잴 때 사용하는 유사도는 msd, cosine, pearson, pearson_baseline 유사도를 각각 활용하여, 총 16가지 조합에 대해서 rmse 값을 KFold(n=5)로 분할하여 평균치를 내서 가장 성능이 뛰어난 알고리즘을 도출한다. 도출 결과, KNNBasic, pearson_baseline 유사도를 설정하는 것이 가장 좋은 성능을 보였다. RMSE는 0.9305325338631297로 도출되었다.

(R3-3)

(1) (R-3-3-1) : 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. Ap3sazzjG3qGt6SvoEIAVA - 4.7
3. AAysJBAlbbyARMRrON-91A - 5.0	3. 9cnPhusg4PPG98EEhTJ7Rw - 4.7	3. Yfdk6FzjuVpeiTS6a6FIvA - 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

표 3.5. TOP-5 User (KNNBasic, cosine, Item based), 소수점 둘째 자리까지 표현.

(2) (R3-3-2) : 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-a1KuIOyMgA – 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

표 3.6. TOP-5 User (KNNWithMeans, pearson, Item based), 소수점 둘째 자리까지 표현.

(3) (R3-3-3) 알고리즘별 성능 평가 비교 및 최적 모델 산정

<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
```

KNNBASIC, KNNWithMeans, KNNWithMeans, KNNWithZScore, KNNBaseline, 각 거리를 잴 때 사용하는 유사도는 msd, cosine, pearson, pearson_baseline 유사도를 각각 활용하여, 총 16가지 조합에 대해서 rmse 값을 KFold(n=5)로 분할하여 평균치를 내서 가장 성능이 뛰어난 알고리즘을 도출한다. 도출 결과, KNNBasic, pearson 유사도를 설정하는 것이 가장 좋은 성능을 보였다. RMSE는 0.932072132214174로 도출되었다.

(R3-4)

(1) (R3-4-1) : SVD(n_factors=100, n_epoch=50, biased=False)

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

2bZ4xxQGSln_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1lDyY5FA
1. 창창 - 1.94	1. 팩피 (FAGP) - 2.52	1. 괴르츠 - 2.07	1. 웨스틴조선서울 아리아 - 2.92	1. 이속우화 - 2.37
2. 고든램지버거 롯데월드몰점 - 1.79	2. 만감 - 2.17	2. 본앤브레드 신관 - 1.95	2. 레에스티우 - 2.22	2. 무드서울 - 1.95
3. 더 키친 일뽀르노 광화문점 - 1.67	3. 괴르츠 - 2.08	3. 고든램지버거 롯데월드몰점 - 1.62	3. 스시우미 잠실 - 1.87	3. 마타헬로 - 1.92
4. 권숙수 - 1.67	4. Truffle di Alba (트러플 디 알바) - 1.89	4. 휘닉스 제주 코지 디너 뷔페 - 1.54	4. 팔레드 신 - 1.71	4. 뜨라또리아 로마냐 (ROMAGNA) - 1.92
5. 스시이로 - 1.67	5. 수퍼판 - 1.81	5. 모도우 광화문점 - 1.46	5. 갓포쿠모 - 1.54	5. 웨스틴조선서울 아리아 - 1.91

표 3.9. TOP-5 식당 SVD(n_factors=100, n_epoch=50, biased=False), 소수점 둘째 자리까지 표현.

(2) (R3-4-2) : SVD(n_factors=200, n_epoch=100, biased=True)

```
algo = surprise.SVD(n_factors=200, n_epochs=100, biased=True)
```

2bZ4xxQGSln_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 어물전 청 한남점 - 3.75	1. 익스퀴진 - 3.90	1. 스시우미 잠실 - 3.80	1. 스시상현 - 3.92	1. 만감 - 3.50
2. 갓포우오 - 3.73	2. 멜팅샵X치즈룸 광화문 디타워 - 3.77	2. 우오하나 - 3.70	2. 디라이프스타일키친 동탄점 - 3.73	2. 우피치바 - 3.46
3. 다이닝고 - 3.67	3. 한우물 - 3.75	3. 디라이프스타일키친 동탄점 - 3.69	3. 미호 오마카세스시 - 3.70	3. 대산 한우안동갈비 - 3.43
4. 수블 - 3.64	4. 비스트로 앤트로 - 3.70	4. 스시히로아키 - 3.68	4. 쥬에 - 3.59	4. 쁘띠따블 - 3.42
5. 조선팰리스 호텔 강남 더 그레이트 홍연 - 3.62	5. 페어링룸 - 3.67	5. 오부이용 - 3.60	5. 어물전 청 한남점 - 3.55	5. 디라이프스타일키친 동탄점 - 3.41

표 3.10. TOP-5 식당 SVD(n_factors=200, n_epoch=100, biased=True), 소수점 둘째 자리까지 표현.

(3) (R3-4-3) : SVD++(n_factors=100, n_epoch=50)

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

2bZ4xxQGSln_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 음 세이로무시(상암점) - 3.75	1. 스시우미 잠실 - 3.70	1. 스키당 스키야키 - 3.68	1. 쥬에 - 4.03	1. 코르바니 선릉 - 3.55
2. 웨스틴조선부산 셔블 - 3.73	2. 이로도리 - 3.67	2. 만감 - 3.65	2. 정식당 - 3.82	2. 허교수스시 - 3.51
3. 모담 라피아노 삼송 직영점 - 3.68	3. 스시키 - 3.63	3. 스시잇센 - 3.64	3. 개성철렵 - 3.75	3. 수블 - 3.51
4. 마그넷 - 3.60	4. 스시무카 - 3.61	4. 도쿠센 - 3.63	4. 마노디셰프 명동점 - 3.64	4. 디해방 D HAEBANG - 3.50
5. 온리고 별내점 - 3.60	5. 소우트(sought) - 3.61	5. 시오(SIIO) - 3.61	5. 익스퀴진 - 3.59	5. 스시키 - 3.49

표 3.11. TOP-5 식당 SVD++(n_factors=100, n_epoch=50), 소수점 둘째 자리까지 표현.

(4) (R3-4-4) : SVD++(n_factors=100, n_epoch=100)

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

2bZ4xxQGSln_LpRVG0smoQ	WHA89VBJuWWkRAID0C6zCw	iqtGbG_2mBe_TVWDQZJmNQ	kN5wr4aaOJWIRvbJZamO6Q	sHp92HPhfEused1IDyY5FA
1. 스시히로아키 - 3.88	1. VINHO - 3.85	1. 스시이로 - 3.86	1. 보스캣 - 3.81	1. 디라이프스타일키친 동탄점 - 3.65
2. 쥬에 - 3.87	2. 라핀부쉬 - 3.83	2. 스시하쿠야 - 3.80	2. 라파예트 클럽하우스 - 3.81	2. 오부이용 - 3.50
3. 우브리앙 - 3.80	3. 스시히로아키 - 3.74	3. 스시케이 - 3.75	3. 비놀릭 - 3.81	3. 스시키요시 - 3.44
4. 만감 - 3.80	4. 익스퀴진 - 3.66	4. CESTA 세스타 - 3.68	4. 알아차림 서래마을 - 3.77	4. 브리즈 - 3.44
5. 스시잇센 - 3.79	5. 초승달 - 3.66	5. 수블 - 3.67	5. 한우물 - 3.75	5. 스시 소우카이 - 3.43

표 3.12. TOP-5 식당 SVD++(n_factors=100, n_epoch=100), 소수점 둘째 자리까지 표현.

(1) (R3-4-5) 알고리즘별 성능 평가 비교 및 최적 모델 산정

<사용한 알고리즘 및 파라미터 설정>

```
# 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
```

SVD, SVD++, NMF 알고리즘에 대해서 각각의 n_factor을 100, 150, 200, 250, 300으로 설정하고, n_epochs를 50, 100, 150, 200으로 설정하고, svd의 경우 biased 값이 true인지 false인지를 설정 후, 각각의 파라미터를 골라서 어떤 조합이 가장 값이 적게 나오는지 탐색한다. 이보다 더 촘촘하게 그리드를 구할 경우 시간이 굉장히 오래 걸리기 때문에 50 단위씩 맞춰서 설정하였다.

<조합에 대한 값 도출 및 최적의 조합 탐색 수행>

```
# 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']
```

그리드 검색으로 각 모델에 대해 파라미터 검색을 수행하여 가장 좋은 모델을 도출한 결과, SVD++ 알고리즘, n_factors = 300, n_epochs = 50으로 설정하여 RMSE가 0.9885273757330861으로 나왔다.