Project 3. DB Mining

(Class : 데이터 관리 및 분석)

10조
나현수, 니키타 김, 박소정, 조현수

Requirement 1st

Code

```
def part1():
    cnx = mysql.connector.connect(host=HOST, user=USER, password=PASSWORD)
    cursor = cnx.cursor()
    cursor.execute('SET GLOBAL innodb_buffer_pool_size=2*1024*1024*1024;')
    cursor.execute('USE %s;' % SCHEMA)

    # TODO: REQUIREMENT 1. WRITE MYSQL QUERY IN EXECUTE FUNCTION BELOW

    cursor.execute("""
            ALTER TABLE gatherings ADD hot_gathering INT(1) DEFAULT 0;;
                """)


    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Hot_Gatherings_List(
        id VARCHAR(22) NOT NULL,
        PRIMARY KEY (id));""")
    #hot_gatherings.txt 파일의 내용을 담기 위한 새로운 테이블을 데이터베이스에 생성


    sql_hot_gatherings_list = """INSERT INTO Hot_Gatherings_List VALUES (%s)"""

    f = open('C:/Users/parks/Desktop/Hot_Gatherings_List.txt', 'r', encoding='utf-8')
```

Explanation

1) gatherings table에 hot_gathering 여부를 나타내기 위한 column을 새로 추가

2) Hot_gathering에 해당되면 1, 해당되지 않으면 0 할당

Requirement 1st

Code

```
sql_hot_gatherings_list = """INSERT INTO Hot_Gatherings_List VALUES (%s)"""

f = open('C:/Users/parks/Desktop/Hot_Gatherings_List.txt', 'r', encoding='utf-8')

while True:
    line_list = []
    line = f.readline()
    line = line.replace('₩n', '')
    line_list.append(line)
    if line:
        cursor.execute(sql_hot_gatherings_list, line_list)
    else:
        break

cnx.commit()
f.close()

cursor.execute("""
    UPDATE gatherings SET hot_gathering = CASE
    WHEN id IN (SELECT id
    FROM Hot_Gatherings_List) THEN 1 ELSE 0 END;
    """)
```

Explanation

1) **Hot_gatherings_list txt file을 연 후, 파일의 레코드를 Hot_Gatherings_List table에 저장**

2) **gatherings에 추가했던 hot_gathering column을 update**

3) **txt파일의 레코드를 추가한 Hot_Gatherings_List 테이블에 gatherings 테이블의 id가 존재하면 1, 존재하지 않으면 0 할당**

Requirement 1st

Code

```
cursor.execute("""
SELECT view5.gathering_id, hot_gathering, age, description, average_rating, number_of_members, number_of_meetings, number_of_recently_joined_members
FROM (SELECT view3.gathering_id, hot_gathering, age, description, average_rating, number_of_members, number_of_meetings
FROM(SELECT view1.gathering_id, hot_gathering, age, description, average_rating, number_of_members
FROM
(SELECT id as gathering_id, hot_gathering, TIMESTAMPDIFF(HOUR, created, '2018-01-01') as age, description, average_rating
FROM gatherings
WHERE year(created) < 2018) as view1 ##여기 수정
LEFT JOIN
(SELECT gathering_id, COUNT(member_id) as number_of_members
FROM member_gathering
GROUP BY gathering_id) as view2
ON view1.gathering_id = view2.gathering_id) as view3
LEFT JOIN
(SELECT gathering_id, COUNT(meeting_place_id) as number_of_meetings
FROM meetings
WHERE year(created) < 2018 ##여기 수정
GROUP BY gathering_id) as view4
ON view3.gathering_id = view4.gathering_id) as view5
LEFT JOIN
(SELECT gathering_id, COUNT(member_id) as number_of_recently_joined_members
FROM member_gathering
WHERE TIMESTAMPDIFF(YEAR, joined, '2018-01-01')<3 ##여기 수정
GROUP BY gathering_id) as view6
ON view5.gathering_id = view6.gathering_id
ORDER BY gathering_id    """)
```

Code

1) gatherings table에서 가져올 수 있는 column들인 gathering_id, hot_gathering, age, description, average_rating을 가져온 후, 이를 view1으로 설정

2) number_of_members를 select하기 위해, member_gathering에서 각 gathering_id별로 member_id를 count한 뒤 이를 view2로 설정

3) view1과 gathering_id가 같은 조건으로 left join을 수행한 뒤 이를 view3로 지정

4) number_of_meetings를 select하기 위해, meetings 테이블에서 각 gathering_id별로 개최한 오프라인 모임의 수를 count하여 view4

5) view3과 gathering_id가 같은 조건에서 left join을 수행해 view5로 지정

6) number_of_recently_joined_members를 select하기 위해 각 그룹별로 오늘과 가입 날짜 간의 년도 차이가 3보다 작은 멤버를 member_gathering 테이블에서 count하여 view6으로 지정

7) view5와 gathering_id가 같은 조건에서 left join을 수행한 후, gathering_id로 ordering

Requirement 2nd

Code

```
fopen = open('project3_team10_data.txt', 'w', encoding='utf8')

rows = cursor.fetchall()
for line in rows:
    for i in range(len(line)):
        fopen.write(str(line[i]))
        if i < len(line) - 1: fopen.write(',')
    if line != rows[len(rows) - 1]: fopen.write('\n')

fopen.close()

# ----------------------------------------------------

# TODO: REQUIREMENT 2. MAKE AND SAVE DECISION TREES

#저장했던 txt파일을 pandas 모듈을 이용해 읽어서 바로 dataframe형식으로 만듦
data_table = pd.read_table('project3_team10_data.txt', delimiter=',',
                           names=['gathering_id', 'hot_gathering', 'age', 'description', 'average_rating',
                                  'number_of_members', 'number_of_meetings', 'number_of_recently_joined_members'])
```

Explanation

1) R2는 R1에서 반환 결과로부터 Hot Gathering 선정 기준에 대한 의사결정 나무를 생성하고자 한다.

2) 저장했던 .txt 파일을 pandas 모듈을 이용해 읽어 dataframe 형식으로 변환

Requirement 2nd

Code

```
#label이 되는 class는 df형식으로 decision tree를 만들 수 없기 때문에 array형식으로 reshape해줌
classes = data_table['hot_gathering'].values.reshape(-1, 1)

#필요한 컬럼만 남기고 모두 없애고, txt파일을 읽어서 null값이 그대로 두지않음 'None'으로 읽히는데, 0가능을 모두 수치 0으로 바꿈. (decisiontreeclassifier에는 수치형만 넣을 수 있음)
features = data_table.drop(['hot_gathering', 'gathering_id'], axis=1)
features['number_of_meetings'] = features['number_of_meetings'].replace('None', 0).astype(int)
features['number_of_members'] = features['number_of_members'].replace('None', 0).astype(int)
features['number_of_recently_joined_members'] = features['number_of_recently_joined_members'].replace('None',0).astype(int)

#CART에 나온 다른 classifier 두 가지 형식으로 생성
DT_gini = tree.DecisionTreeClassifier(criterion='gini', min_samples_leaf=10, max_depth=5)
DT_gini.fit(X=features, y=classes) #samples_leaf ~

graph_gini = tree.export_graphviz(DT_gini, out_file=None,
                                  feature_names=['age', 'description', 'average_rating', 'number_of_members',
                                                 'number_of_meetings', 'number_of_recently_joined_members'],
                                  class_names=['normal', 'HOT'])
graph_gini = graphviz.Source(graph_gini)
graph_gini.render('DM_project3_team10_gini', view=True)
```

Explanation

1) Label이 되는 class는 df형식으로 decision tree를 만들 수 없기 때문에 array 형식으로 reshape

2) 필요한 칼럼만 남기고 모두 없앤 이후 txt 파일이 읽어 null 값이 'None'으로 읽히는 것을 모두 0으로 바꿈.

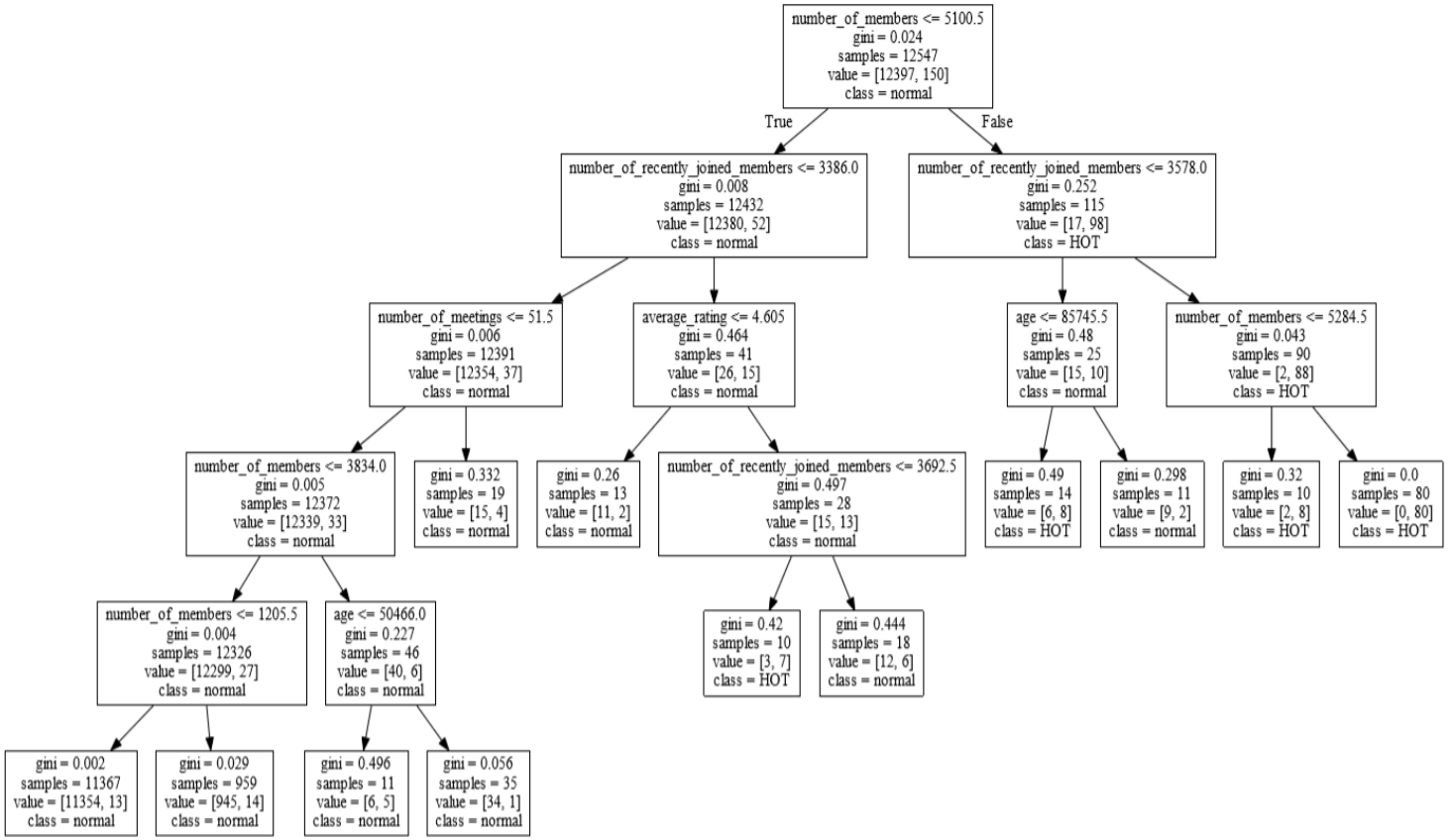
3) node impurity 측정 criterion을 gini로 설정한 tree를 생성하고, min_samples_leaf=10, max_depth=5로 지정

4) 이 때 class는 hot_gatherings가 아니면 normal로, hot_gatherings에 해당되면 HOT으로 설정

Requirement 2nd

Results

1) node impurity 측정방식으로 gini를 사용한 의사결정나무



Requirement 2nd

Results

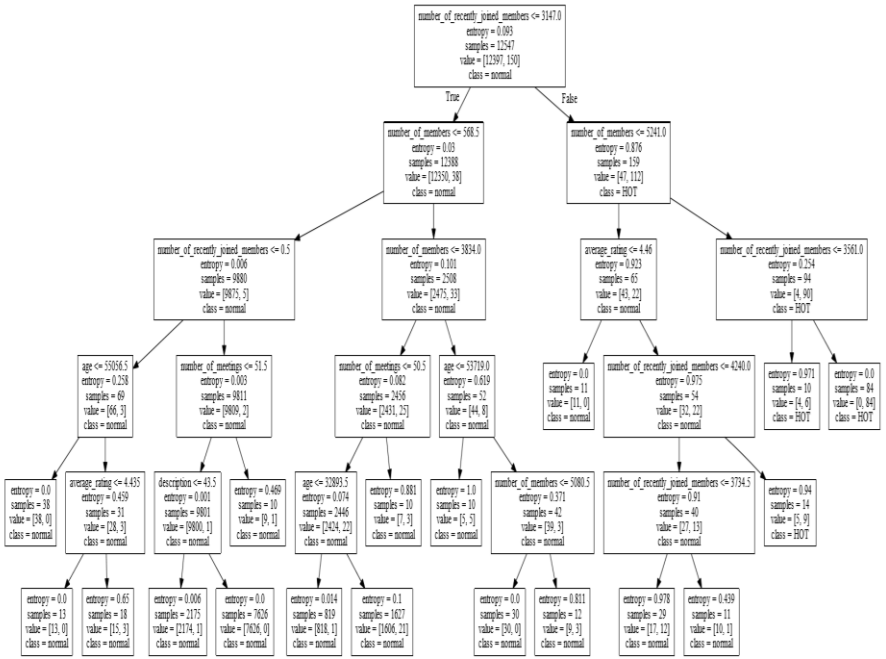
2) node impurity 측정방식으로 entropy를 사용한 의사결정나무

```
DT_entropy = tree.DecisionTreeClassifier(criterion='entropy', min_samples_leaf=10, max_depth=5)
DT_entropy.fit(X=features, y=classes)


graph_entropy = tree.export_graphviz(DT_entropy, out_file=None,
                                     feature_names=['age', 'description', 'average_rating', 'number_of_members',
                                                    'number_of_meetings', 'number_of_recently_joined_members'],
                                     class_names=['normal', 'HOT'])
graph_entropy = graphviz.Source(graph_entropy)
graph_entropy.render('DMA_project3_team10_entropy', view=True)


# ------------------------------------------------

cursor.close()
```



Requirement 3rd

설정 및 이유

- 새롭게 recently_active와 promptness_active라는 column을 만들고 age column 삭제

- recently_active : 오프라인 만남 정보가 최근에 업데이트 된 날짜 사이의 거리가 작을수록 recently active했으므로 hot_gathering 여부에 영향

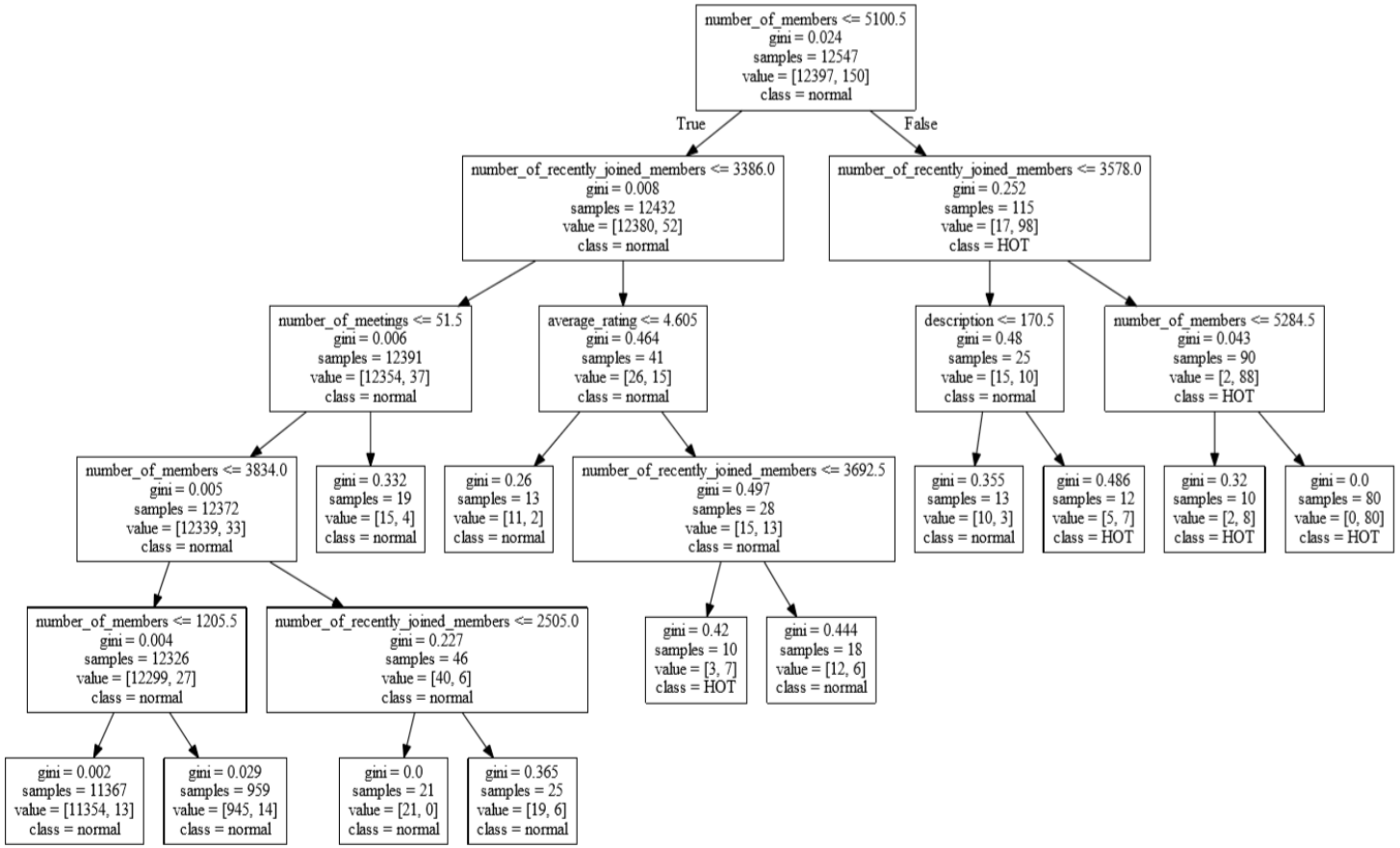
- promptness_active : 오프라인 만남이 처음 정해진 날짜와 실제 모임 날짜 사이의 간격이 작을수록 신속성이 높아 hot_gathering 여부에 영향

- 신생 그룹이어도 최근 활발하면 hot_gathering에 선정될 수 있다 판단해 age 삭제

- Node impurity : gini, entropy 모두 활용

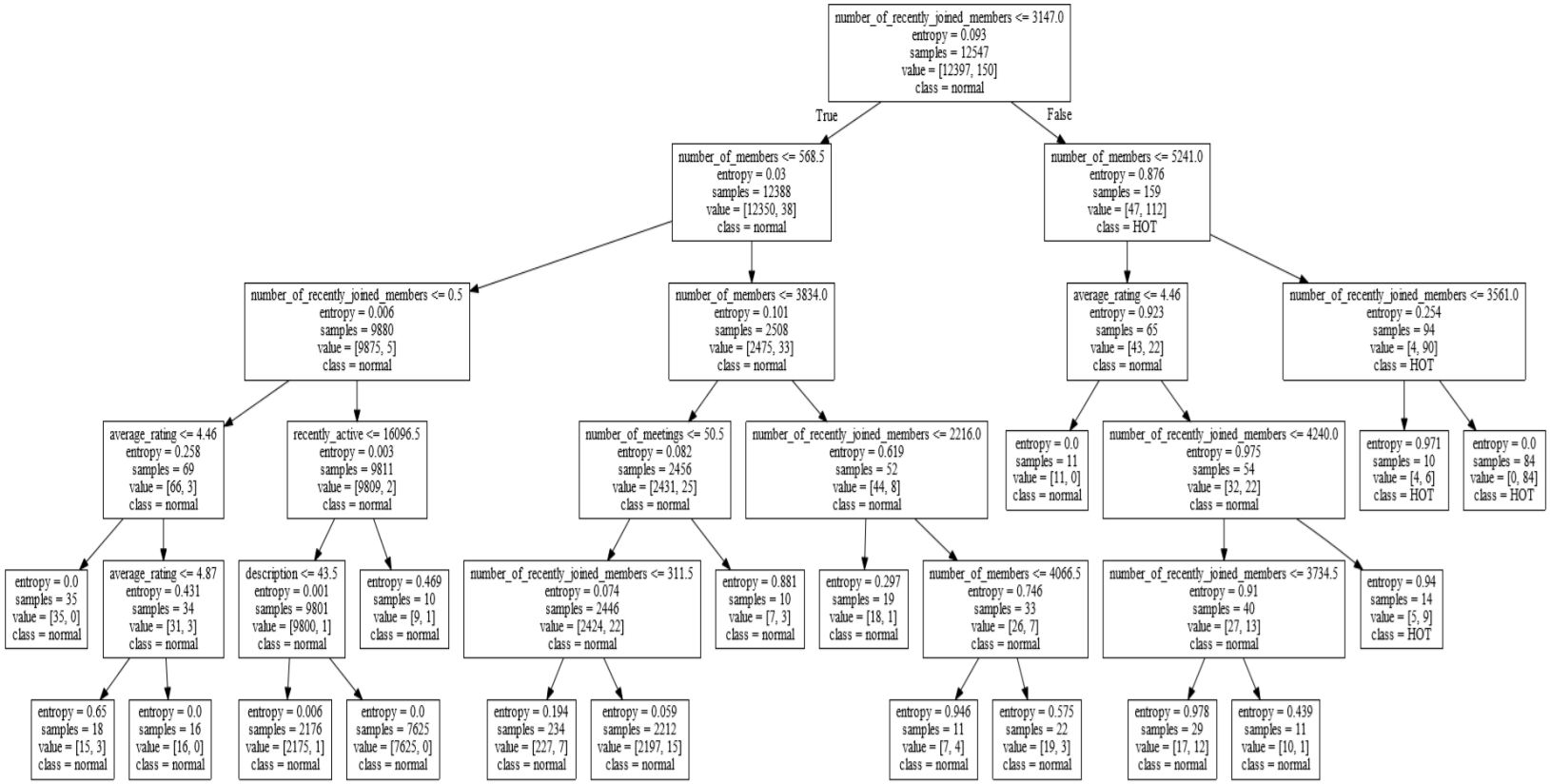
Requirement 3rd

gini Method



Requirement 3rd

entropy Method



Requirement 4th

Code

```
def part2():
    cnx = mysql.connector.connect(host=HOST, user=USER, password=PASSWORD)
    cursor = cnx.cursor()
    cursor.execute('SET GLOBAL innodb_buffer_pool_size=2*1024*1024*1024;')
    cursor.execute('USE %s;' % SCHEMA)
    cursor.execute('DROP VIEW IF EXISTS maybe_NY_gatherings;')
    cursor.execute('DROP VIEW IF EXISTS NY_gatherings;')

# TODO: REQUIREMENT 4. WRITE MYSQL QUERY IN EXECUTE FUNCTIONS BELOW

# Query on Members who live in NY
cursor.execute('''
    CREATE OR REPLACE VIEW M
    AS SELECT id, city_id
    FROM members
    WHERE city_id = 10001
    ''')

# Gatherings that have 99% of members live in NY
# (SUM(CASE WHEN city_id = 10001 THEN 1 ELSE 0 END)) Counts number of people who live in NY
# Count (*) counts total number of people belong to the gathering
cursor.execute('''
    CREATE OR REPLACE VIEW 99_NY
    AS SELECT gathering_id
    FROM member_gathering, members
    WHERE member_gathering.member_id = members.id
    GROUP BY gathering_id
    HAVING (SUM(CASE WHEN city_id = 10001 THEN 1 ELSE 0 END)) >= (COUNT(*) * 0.99)
''')
```

Explanation

1) 뉴욕 주의 뉴욕 시에서 활동하는 사용자들이 활동하는 그룹들 사이 연관 분석

2) 뉴욕에 살고 있는 가입자를 대상으로 수행

3) 구성원 중 99% 이상이 NY 출신인 Gathering을 선별하는 View 99_NY 생성

Requirement 4th

Code

```
# Gatherings and members 99% NY
cursor.execute('''
    CREATE VIEW maybe_NY_gatherings
    AS SELECT member_gathering.gathering_id AS gathering_id, member_id
    FROM 99_NY, member_gathering
    WHERE 99_NY.gathering_id = member_gathering.gathering_id
''')

# Gatherings that have 100% of members live in NY
cursor.execute('''
    CREATE OR REPLACE VIEW 100_NY
    AS SELECT gathering_id as gathering_id
    FROM maybe_NY_gatherings
    LEFT JOIN M ON maybe_NY_gatherings.member_id = M.id
    GROUP BY gathering_id
    HAVING ((SUM(CASE WHEN city_id = 10001 THEN 1 ELSE 0 END)) = (COUNT(*)))
''')

# Gatherings and members of 100% NY
cursor.execute('''
    CREATE VIEW NY_gatherings
    AS SELECT member_gathering.gathering_id AS gathering_id, member_id
    FROM 100_NY, member_gathering
    WHERE 100_NY.gathering_id = member_gathering.gathering_id
''')

# ————————————————————————————————————————————————
```

Explanation

1) 구성원 중 99% 이상이 NY 출신인 Gathering의 gathering_id, member_id를 SELECT

2) 구성원 중 100%가 NY 출신인 Gathering을 선별하는 View 100_NY 생성

3) 구성원 중 100%가 NY 출신인 Gathering을 선별하는 NY_gathering 생성

Requirement 5th

Code

```
# TODO: REQUIREMENT 5. MAKE HORIZONTAL DATAFRAME

# Get non-duplicate (unique) values on Gathering ID
cursor.execute('''
    SELECT DISTINCT gathering_id
    FROM NY_gatherings

''')

gathering_id = cursor.fetchall()
select_columns = ''
```

```
# make select columns : eg max(IF(gathering_id = 10010442, 1, 0)
# For more info, Check TA07 pg 13
for dot in gathering_id:
    select_columns = select_columns + "max(IF(gathering_id='{0}',1,0)) as '{1}',".format(dot[0], dot[0])
select_columns = select_columns[:-1]

# Make Horizontal DataFrame
cursor.execute('''
    SELECT member_id, ''' + select_columns + '''
    FROM NY_gatherings
    GROUP BY member_id
''')

# Save it as pandas
df = pd.DataFrame(cursor.fetchall())
df.columns = cursor.column_names
df = df.set_index('member_id')
print(df)

cnx.close()
cursor.close()
# ----------------------------------------
```

Explanation

1) R5는 NY_gatherings을 horizontal table로 만들고 pandas의 DataFrame으로 저장하는 쿼리

2)row index는 member_id로 GROUP BY

3)DataFrame으로 저장하는 과정에서 member_id를 index로 선언

Results

```
Requirement 5
           10010442  10015542  10052602  ...  9798532  9829012  9849892
member_id                                ...
13355089          0         0         0  ...        0        0        0
13355180          0         0         0  ...        0        0        0
13355330          0         0         0  ...        0        0        0
13355433          0         0         0  ...        0        0        0
13355953          0         0         0  ...        0        0        0
...             ...       ...       ...  ...      ...      ...      ...
13349444          0         0         0  ...        0        0        0
13349947          0         0         0  ...        0        0        0
13352617          0         0         0  ...        0        0        0
13352740          0         0         0  ...        0        0        0
13352770          0         0         0  ...        0        0        0

[71207 rows x 2086 columns]
```

Requirement 6th

Code

```
# TODO: REQUIREMENT 6. SAVE ASSOCIATION RULES

frequent_itemsets = apriori(df, min_support=0.0025, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=0.5)
rules.to_csv('DMA_project3_team10_association.txt')

# ----------------------------------------------
```

Explanation

1) frequent itemset을 만들고 연관분석을 수행하는 쿼리

2) min_support는 0.0025로 support가 0.0025이상인 itemset을 frequent itemset으로 정의

3) lift가 0.5이상인 경우 association rule이라고 보고 Results를 출력

Results

,antecedents,consequents,antecedent support,consequent support,support,confidence,lift,leverage,conviction