Project #3
DB Mining

교수: 박종현 교수님

조교: 윤동현 조교님

제출일: 2019-12-11

10 조

2013-13441 나현수
2017-12525 니키타 김
2018-12530 박소정
2014-12615 조현수

<목차>

프로젝트 개요

본 프로젝트는 주어진 데이터를 사용하여 연관 분석 및 의사결정 나무 모델링, 그리고 이의 결과를 분석하는 것을 목적으로 수행한다. 프로젝트에 사용될 데이터는 프로젝트 #2 에서 진행했던 사이트 A 의 데이터를 기반으로 하려고 한다.

해당 프로그램은 Python 및 MySQL 사용을 통한 구현을 권장한다. 만족해야 할 요구 조건은 총 6 개 있으며, 프로젝트는 두 부분으로 나눌 수 있다.

1. 의사결정 나무 – 1~3 번의 요구 조건.
2. 연관 분석 – 4~6 번의 요구 조건.

PART I. 의사결정 나무

Query & Explanation:

(R1)

```
import mysql.connector
import os
import pandas as pd
from sklearn import tree
import graphviz
import numpy as np
from mlxtend.frequent_patterns import association_rules, apriori


# TODO: CHANGE GRAPHVIZ DIRECTORY
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

# TODO: CHANGE MYSQL INFORMATION
HOST = 'localhost'
USER = 'root'
PASSWORD = '01094321205'
SCHEMA = 'DMA_team12'
```

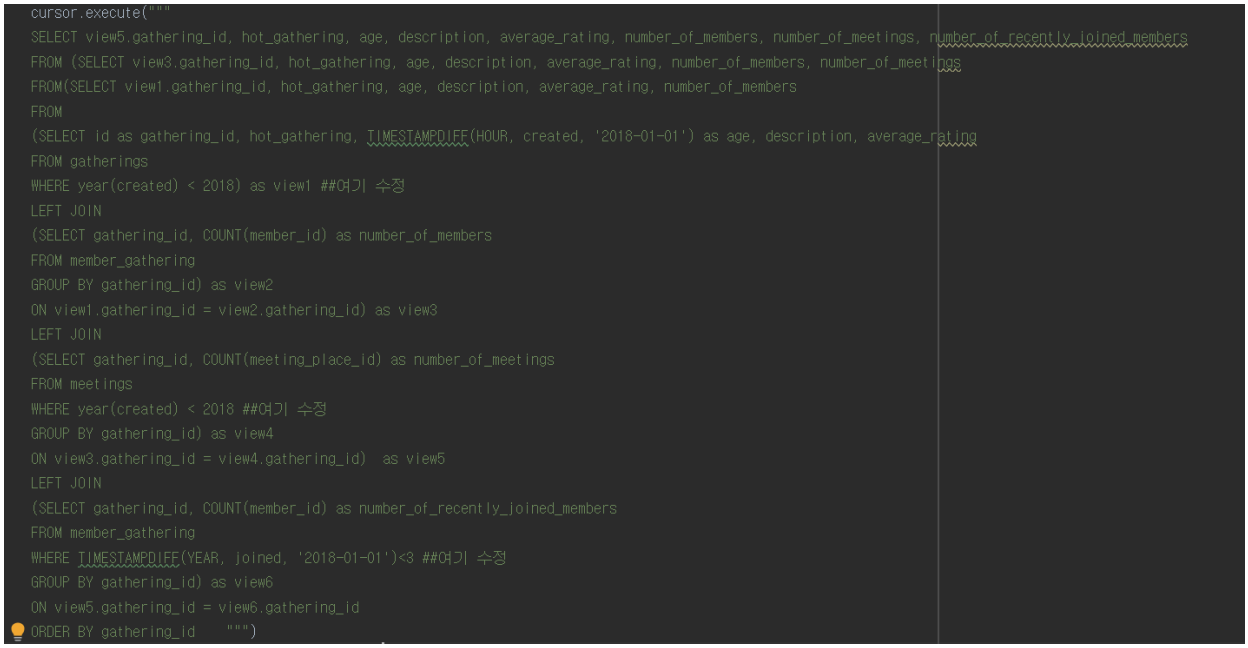
필요한 모듈들을 모두 import 하고, graphviz environment 를 설정해준다. 이 때 mysql 에 접근하는 데 필요한 host, user, password 와 project2 때 사용했던 schema 도 함께 설정한다.

```
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')
```

1. 먼저 gatherings table 에 hot_gathering 여부를 나타내기 위한 column 을 새로 추가한다. Hot_gathering 에 해당되면 1, 해당되지 않으면 0 이 이후에 할당될 것이다.
2. hot_gatherings_list txt 파일을 바로 sql 에 적용할 수 없으므로, hot_gatherings txt 파일의 내용을 담기 위한 새로운 테이블을 데이터베이스에 생성한다. 테이블 이름은 Hot_Gatherings_List 이다.

```
    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;
        """)
    #새로 추가한 hot_gathering column을 update. Hot_Gatherings_List로 위에서 생성한 TABLE 안에 포함되어 있으면 (주어진 hot_gatherings의 txt파일에 포함되어 있으면) 1, 없으면 0 할당
```

3. Hot_gatherings_list txt file 을 연 후, 파일의 레코드를 Hot_Gatherings_List table 에 저장한다.
4. 그 후, gatherings 에 추가했던 hot_gathering column 을 update 한다. txt 파일의 레코드를 추가한 Hot_Gatherings_List 테이블에 gatherings 테이블의 id 가 존재한다면 hot_gatherings 이므로 1, 없으면 hot_gatherings 가 아니므로 0 이 할당된다.



```
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    """)
```

5. R1 에서 요구하는 내용을 select 하는 쿼리문을 작성하였다. 원하는 column 은 총 8 개이다. 먼저 gatherings table 에서 가져올 수 있는 column 들인 gathering_id, hot_gathering, age, description, average_rating 을 가져온 후, 이를 view1 으로 설정하였다.
그리고 number_of_members 를 select 하기 위해, member_gathering 에서 각 gathering_id 별로 member_id 를 count 한 뒤 (그룹별 가입자를 전부 select) 이를 view2 로 설정하고, view1 과 gathering_id 가 같은 조건으로 left join 을 수행한 뒤 이를 view3 로 지정하였다.
그리고 number_of_meetings 를 select 하기 위해, meetings 테이블에서 각 gathering_id 별로 개최한 오프라인 모임의 수를 count 하였는데, 현재 시간이 2018-01-01 이므로 2018 이후에 created(미팅 날짜를 정한 날짜)가 수행되지 않은 모임들만을 count 하였다. 이를 view4 로 지정한 뒤 view3 과 gathering_id 가 같은 조건에서 left join 을 수행해 view5 로 지정하였다.
마지막으로 number_of_recently_joined_members 를 select 하기 위해 각 그룹별로 오늘과 가입 날짜 간의 년도 차이가 3 보다 작은 멤버를 member_gathering 테이블에서 count 하여 view6 으로 지정하였다. 그 후 앞의 view5 와 gathering_id 가 같은 조건에서 left join 을 수행한 후, gathering_id 로 ordering 하였다.

이렇게 left join 을 계속 수행하여 만든 table 로부터, 문제에서 요구하는 8 개의 column 을 select 하였다.

(R2)

```
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'])
```

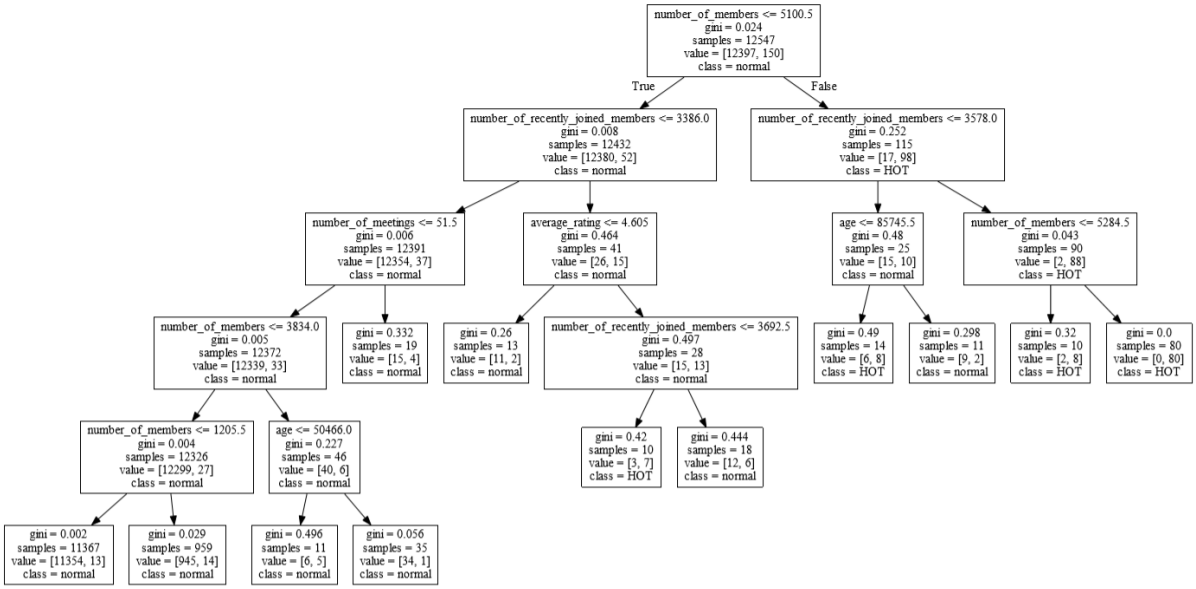
Select 한 결과를, project3_team10_data.txt 파일로 만들어 저장하였다. 그 후, 저장했던 txt 파일을 pandas 모듈을 이용해 읽은 후 바로 dataframe 형식으로 만들었다.

```
#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 문자열 'None'으로 읽히는데, 이것들을 모두 숫자 0으로 바꿈. (decisiontreeclassifier에는 숫자형만 넣어야 함)
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)

#TA07에 나온 대로 classifier 두 가지 방식으로 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('DMA_project3_team10_gini', view=True)
```

생성된 dataframe 인 data_table 에서 label 이 되는 hot_gathering 은 tree 를 생성할 때 df 형식이 아닌 array 형식으로 들어가야 하기 때문에 value 를 reshape 해 주었다.

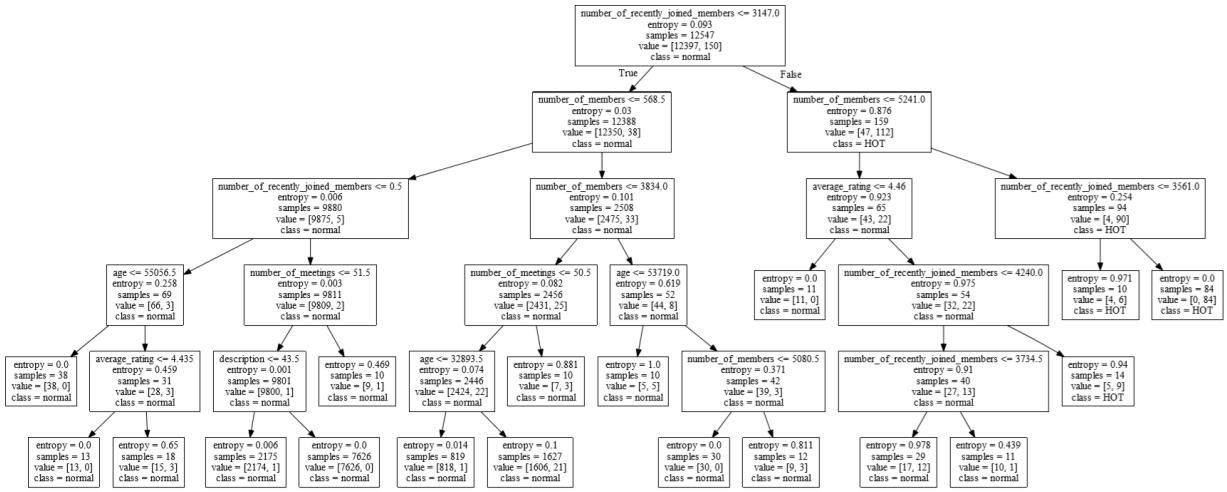
Features 의 대상이 되는 column 은 6 개로, 8 개의 column 중 class label 과 id 를 drop 하였고 DT 에는 숫자형만 넣어야 하기 때문에 null 을 'None'이라고 문자열로 읽은 것을 모두 0 으로 바꾼 후 int type 을 지정해 주었다.

그 후 node impurity 측정 criterion 을 gini 로 설정한 tree 를 생성하고, min_samples_leaf=10, max_depth=5 로 지정하였다. 위에서 설정한 classes 와 fetures 를 tree 에 fit 한 후 graphviz 로 트리를 나타내었다. 이 때 class 는 hot_gatherings 가 아니면 normal 로, hot_gatherings 에 해당되면 HOT 으로 설정하였다. 생성된 gini decision tree 는 다음과 같다.




```
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()
```

Criterion 이 entropy 인 DT 를 생성한 후 앞에서와 같이 graphviz 를 이용하여 트리 view 를 생성한다. 생성된 entropy tree 는 다음과 같다.



(R3)

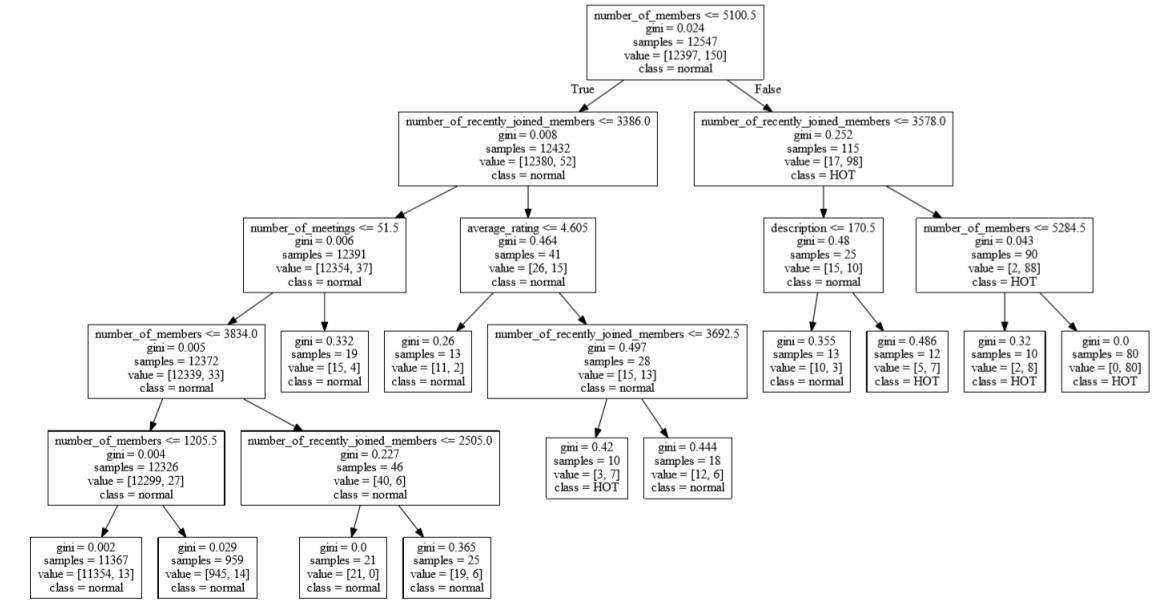
```
cursor.execute("""
SELECT view7.gathering_id, hot_gathering, age, description, average_rating, number_of_members, number_of_meetings, number_of_recently_joined_members, recently_active, promptness_active
FROM
(SELECT view5.gathering_id, hot_gathering, age, description, average_rating, number_of_members, 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
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
GROUP BY gathering_id) as view4
ON view3.gathering_id = view4.gathering_id)  as view5
```

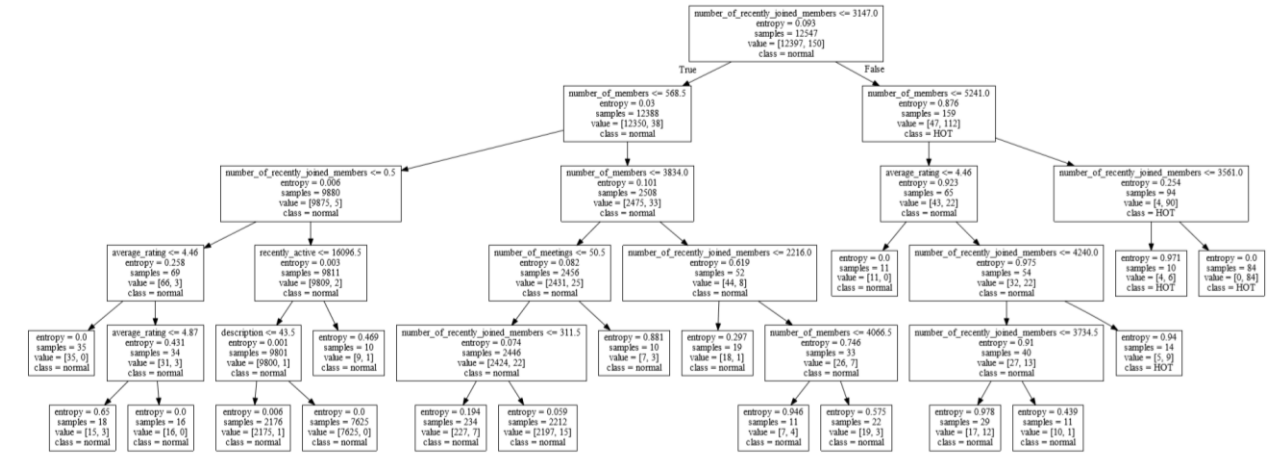
```
LEFT JOIN
(SELECT gathering_id, COUNT(member_id) as number_of_recently_joined_members
FROM member_gathering
WHERE TIMESTAMPDIFF(YEAR, joined, '2018-01-01')<3
GROUP BY gathering_id) as view6
ON view5.gathering_id = view6.gathering_id) as view7
LEFT JOIN
(SELECT gathering_id, TIMESTAMPDIFF(HOUR, updated, '2018-01-01') as recently_active, TIMESTAMPDIFF(HOUR, created, time) as promptness_active
FROM meetings) as view8
ON view7.gathering_id = view8.gathering_id
GROUP BY gathering_id
ORDER BY gathering_id

""")
```

```
features2 = data_table2.drop(['hot_gathering', 'gathering_id'], axis=1)
features2['number_of_meetings'] = features2['number_of_meetings'].replace('None', 0).astype(int)
features2['number_of_members'] = features2['number_of_members'].replace('None', 0).astype(int)
features2['number_of_recently_joined_members'] = features2['number_of_recently_joined_members'].replace('None', 0).astype(int)
features2['recently_active'] = features2['recently_active'].replace('None', 0).astype(int)
features2['promptness_active'] = features2['promptness_active'].replace('None', 0).astype(int)

#age 지움. 위에서 설명함.
features2 = features2.drop(['age'], axis=1)

DT_gini = tree.DecisionTreeClassifier(criterion='gini', min_samples_leaf=10, max_depth=5)
DT_gini.fit(X=features2, y=classes2)

graph_gini = tree.export_graphviz(DT_gini, out_file=None, feature_names=['description', 'average_rating', 'number_of_members', 'number_of_meetings',
                                                                         'number_of_recently_joined_members', 'recently_active', 'promptness_active'], class_names=['normal', 'HOT'])
graph_gini = graphviz.Source(graph_gini)
graph_gini.render('DMA_project3_team10_gini2', view=True)

DT_entropy = tree.DecisionTreeClassifier(criterion='entropy', min_samples_leaf=10, max_depth=5)
DT_entropy.fit(X=features2, y=classes2)

graph_entropy = tree.export_graphviz(DT_entropy, out_file=None, feature_names=['description', 'average_rating', 'number_of_members', 'number_of_meetings',
                                                                               'number_of_recently_joined_members', 'recently_active', 'promptness_active'], class_names=['normal', 'HOT'])
graph_entropy = graphviz.Source(graph_entropy)
graph_entropy.render('DMA_project3_team10_entropy2', view=True)
```

R2 와 다른 의사결정 나무를 생성하고자 한다. Decision tree 의 파라미터를 변경하지는 않았으며, 새로운 컬럼을 두 개 추가하고 하나의 컬럼을 삭제하였다.
오프라인 만남의 정보가 가장 최근에 업데이트된 날짜: updated 인데, 오늘(2018-01-01)로부터 오프라인 만남 정보가 최근에 업데이트된 날짜 사이의 거리가 작을수록 recently active 했다는 의미이므로 hot_gatherings 에 영향을 줄 것이라고 판단하여서 recently_active 라는 컬럼을 추가하였다.

그리고 오프라인 만남이 처음 정해진 날짜: created 이고, 오프라인 만남의 실제 모임 날짜: time 인데, 이 두 날짜 간의 차이가 작을수록 모임이 개최되는 신속성이 높고 hot_gatherings 여부에 영향을 줄 것이라고 판단하여서 promptness_active 라는 컬럼을 추가하였다.
그리고 신생 그룹이어도 최근의 활동이 활발하고 모임이 잘 이루어진다면 hot_gatherings 에 선정될 수 있다고 판단하여서 모임이 생성된지 얼마나 오래됐는지를 나타내는 컬럼인 age 를 dataframe 에서 drop 한 후 decision tree 를 생성하였다. Criterion 은 R2 와 같이 gini 와 entropy 를 모두 사용하여 decision tree 를 생성해 보았다. 그 결과는 다음과 같다.





R2 와는 확연히 다른 트리들이 생성되는 것을 확인할 수 있었다.

PART II. 연관 분석

(R4)

문제 개요:

New York 주의 New York 시에서 활동하는 사용자들이 활동하는 그룹들 사이의 연관 분석을 수행하고자 한다. 먼저, 그룹에 가입한 사용자 중 99%의 사용자가 New York 주의 New York 시에서 활동하는 그룹을 구하여라. 분석을 위해 아래의 column들이 포함되는 'maybe_NY_gatherings'라는 이름을 가지는 view를 생성하라.

- gathering_id: 그룹의 id
- member_id: 해당 그룹에 속한 사용자 id

위의 view를 만든 후 위 그룹 중 모든 사용자가 NewYork 주의 NewYork 시에서 활동하는 그룹에 대한 정보만을 남긴 'NY_gatherings'라는 이름을 가지는 view를 생성하라. NY_gatherings가 가지는 column은 maybe_NY_gatherings 의 column과 동일해야 한다.

Query & Explanation:

(R4)

```
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
    ''')
```

Query 설명 : 뉴욕에 살고 있는 member 을 선별하는 View M 생성
1. members 테이블에서 도시 ID 가 10001 (NY)인 레코드를 선별한다
2. 선별된 레코드에서 id 와 city_id 를 선별한다

```
# 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)
''')
```

Query 설명 : 구성원 중 99% 이상이 NY 출신인 Gathering 을 선별하는 View 99_NY 생성
1. member_gathering 과 members 테이블을 ID 기준으로 합친다
2. 1 의 결과값을 Gathering ID 로 Group By 한다
3. 각 Member 의 City ID 가 10001 이면 1, 아니면 0 을 부여한다
4. 부여된 값을 Gathering ID 로 Group By 한 후에, 값의 총합을 구한다. 이는 하나의 Gathering 에서 NY 에 살고 있는 구성원의 수를 나타낸다.
5. 4 의 결과값과 각 Gathering 의 Count(*), 즉 구성원 총원을 비교한다
6. 5 의 결과값 중 0.99 이상의 값만을 가지는 Gathering 만 선별한다
7. 6 의 결과에서 Gathering_ID 만 보인다

```
# 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
''')
```

Query 설명 : 구성원 중 99% 이상이 NY 출신인 Gathering 의 gathering_id, member_id
1. 99_NY 와 member_gathering 을 호출한다
2. 호출된 두 테이블의 gathering_id 에 따라 merge 한다
3. 2 의 결과에서 gathering_id 와 member_id 를 select 한다

```
# 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(*)))
''')
```

Query 설명 : 구성원 중 100%가 NY 출신인 Gathering 을 선별하는 View 100_NY 생성
1. maybe_NY_gatherings 과 members 테이블을 LEFT JOIN 한다
2. 1 의 결과값을 Gathering ID 로 Group By 한다
3. 각 Member 의 City ID 가 10001 이면 1, 아니면 0 을 부여한다
4. 부여된 값을 Gathering ID 로 Group By 한 후에, 값의 총합을 구한다. 이는 하나의 Gathering 에서 NY 에 살고 있는 구성원의 수를 나타낸다.
5. 4 의 결과값과 각 Gathering 의 Count(*), 즉 구성원 총원을 비교한다
6. 5 의 결과값 중 1 의 값만을 가지는 Gathering 만 선별한다
7. 6 의 결과에서 Gathering_ID 만 보인다

```
# 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
''')

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

Query 설명 : 구성원 중 100%가 NY 출신인 Gathering 을 선별하는 NY_gathering 생성
1. 99_NY 와 member_gathering 을 호출한다
2. 호출된 두 테이블의 gathering_id 에 따라 merge 한다
3. 2 의 결과에서 gathering_id 와 member_id 를 select 한다

(R5)

문제 개요:

NY_gatherings을 horizontal table로 만든 결과를 pandas의 DataFrame으로 저장하라. 이때, 저장된 DataFrame은 그룹의 id를 column명으로 가져야 하며, 각 row의 index는 사용자의 id로 저장되어야 한다. DataFrame의 각 사용자는 연관 분석의 transaction의 역할을, 각 그룹의 id는 연관 분석의 item의 역할을 하게 된다.

```
# 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 = ''
```

생성된 NY_gatherings 에서 Unique 한 Gathering_id 를 뽑아내기 위해 DISTINCT 명령어를 이용한다.

```
# 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
''')
```

저장된 Gathering_id 를 하나씩 호출하여 추후의 Select 문에 들어갈 수 있도록 형태를 지정한다. 이렇게 만들어진 하나의 긴 String 형태를 Query 의 Select 문에 삽입한다
해당 Query 는 Member_id 를 하나의 열로, gathering_id 하나씩을 다른 열들로 가지게 된다.

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

위의 Query 로 생성된 값을 pandas 형태로 변환하여 저장한다.

```
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]
```

R5 에 대한 결과값

(R6)

문제 개요:

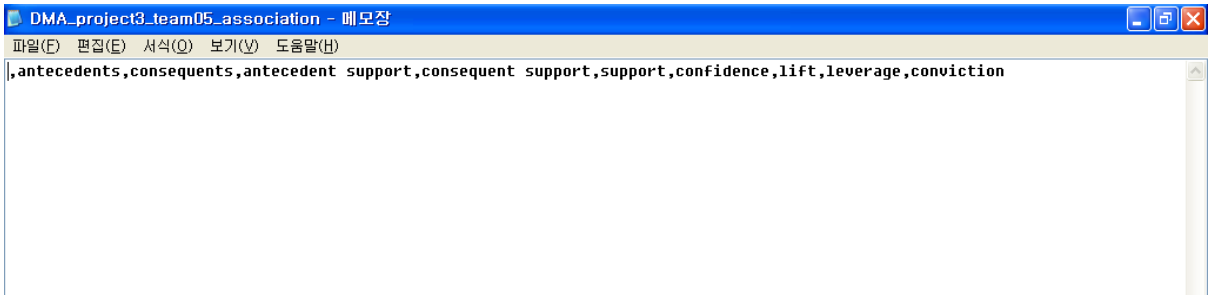
R5에서 만든 DataFrame을 사용해서 다음의 조건을 만족하는 frequent itemset을 만들고 연관 분석을 수행하라.

- frequent itemset 의 최소 support: 0.0025
- 연관 분석 metric: lift (lift>=0.5 인 것들을 출력)

결과 파일 명: DMA_project3_team##_association.txt

```
# 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')

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

Apriori 라이브러리를 호출하여 Min_support 조건을 만족하는 Frequent Itemset 을 만든다
Association Rules 라이브러리를 호출하여 Lift, threshold 0.5 를 만족하는 Rule 을 생성한다.



R6 에 대한 결과값

이를 만족하는 결과가 없는 것으로 판명되었다.