Database Mining

Making Decision Tree & Conducting Association Analysis Using data from Site A

제출일	2018. 12. 13.	**담당교수**	박종헌
작성자	산업공학과 2018-28087		최민
	자유전공학부 2017-15556		김경준
	산업공학과 2017-16495		추성민
	산업공학과 2017-17047		강명오

1. 문제 정의

본 프로젝트는 site A의 데이터를 활용하여 데이터베이스에 감춰진 관계들을 발견하는 것(Association Analysis)과 각 test record의 attribute들에 대해서 일련의 질문을 수행함으로써 Classification Problem을 해결하는 것(Decision Tree)을 목표로 한다. 앞선 프로젝트와 유사하게 본 프로젝트에서도 역시 python과 MySQL이 requirement 구현의 주 매개체가 될테지만, 선행 프로젝트들과는 달리 해당 프로젝트의 경우 sklearn, graphviz, mlxtend 라이브러리를 사용하여 결과를 시각화하도록 설계하는 데 그 목적이 있다.

본 보고서의 경우, 문제 분석 항목에서 Solution 제시, 각 query의 조건 분석, 문제 해결의 방향성, 요구 사항의 만족 여부 검토 및 결과 분석이라는 일련의 과정으로 기술될 예정이다. 한편, 요약 및 결론 단계에서는 본 프로젝트에 사용된 기법들이 현실 상황에서는 실제로 어떠한 방식으로 적용될 수 있는지에 대한 간단한 모형을 상정해보고, 그에 대한 가능성을 검토해보는 방향으로 진행하고자 한다.

2. 문제분석

1) R1 Analysis

1.1 Solution

R1을 수행하기 위한 개괄적인 코드는 다음과 같다.

```
cursor.execute("""
    SELECT user_id, if(year=2017,1,-1) as vip, ifnull(review_count,0), ifnull(positive,0), ifnull(review_avg,0),
        ifnull(review_sum,0), ifnull(tip_count,0), ifnull(Tip_likes,0) as likes,
        ifnull(tip_avg,0), ifnull(tip_sum,0)
    FROM (SELECT U1.id as user_id, review_count, positive, review_avg, review_sum
          FROM user as U1
          LEFT JOIN(    SELECT user_id as idR, COUNT(*) as review_count, SUM(R1.useful)+SUM(R1.funny)+SUM(R1.cool)
                            as positive, AVG(R1.text_length) as review_avg, SUM(R1.text_length) as review_sum
                        FROM review as R1
                        WHERE R1.date like '%2017%'
                        GROUP BY user_id)R
          ON U1.id=R.idR)UR
    INNER JOIN( SELECT id as user_id2, tip_count, Tip_likes, tip_avg, tip_sum
                FROM user as U2
                LEFT JOIN(  SELECT user_id as idT, COUNT(*) as tip_count, SUM(T1.likes) as Tip_likes,
                                AVG(T1.text_length) as tip_avg, SUM(T1.text_length) as tip_sum
                            FROM tip as T1
                            WHERE T1.date like '%2017%'
                            GROUP BY T1.user_id)T
                ON U2.id=T.idT)UT
    ON UR.user_id=UT.user_id2
    LEFT JOIN(SELECT user_id as idV, year
              FROM vip_history as V
              WHERE V.year=2017)V1
    ON UR.user_id=V1.idV
    WHERE review_count is not null or tip_count is not null
    """)
```

,Figure 1.1.1

1.2 Query 분석

User id, vip, review count, positive, review avg, review sum, tip count, likes, tip avg, tip sum이 포함된 결과를 반환하는 쿼리를 작성해야 한다. 특히, 하나 이상의 리뷰 혹은 정보를 남긴 사용자들에 한에서만 저장되어야 한다.

1.3 문제 해결 방향성

MySQL에서 full outer join이 구현되지 않아, 처음에는 조건을 만족시킬 수 있도록 review에서 각각의 조건값을 계산해 준 nested query(이하 R)와 tip에서 각각의 조건값을 계산해 준 nested query(이하 T) 의 합집합을 (R left join T) UNION (T left join R)을 이용해 구현한 후, 2017년의 vip를 골라낸 nested query를 구현한 query에 left join해서 결과를 반환하고자 하였으나, R과 T를 구하는 과정에서 group by를 쓰지 않을 수가 없었기에, group by를 필요 이상으로 사용한다고 느껴 제시된 코드와 같이 user에 R을 left join한 것과 user에 T를 left join한 것을 inner join한 후 그것에 vip를 골라낸 query를 다시 left join한 것에서 결과를 구했다. 특히 결과값을 select할 때, tip 또는 review에 관한 정보가 둘 다 null이 아닌 것 중에서 고르도록 했고, tip 또는 review 정보 중 하나는 null값들을 가질 수 있기 때문에 모든 값에 ifnull함수를 활용해 null값인 경우에 0을 도출하도록 했고, vip 선정 여부는 후에 class에 속하므로 if문을 이용해서 1과 -1로 구분하였다.

2) R2 Analysis

2.1 Solution

R2를 수행하기 위한 개괄적인 코드는 다음과 같다.

```
# R2
    rows = cursor.fetchall()

    classes = []
    features = []
    for row in rows:
        feature_row = []
        classes.append(row[1])
        for i in range(2, 10):
            feature_row.append(row[i])
        features.append(feature_row)


    DT1 = tree.DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_leaf=500)
    DT1.fit(X=features, y=classes)

    DT2 = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=500)
    DT2.fit(X=features, y=classes)

    graph1 = tree.export_graphviz(DT1, out_file=None,
                                  feature_names=['review_count', 'positive', 'review_avg', 'review_sum',
                                                 'tip_count', 'likes', 'tip_avg', 'tip_sum'],
                                  class_names=['Normal', 'VIP'])
    graph1 = graphviz.Source(graph1)
    graph1.render('team08_gini', view=True)

    graph2 = tree.export_graphviz(DT2, out_file=None,
                                  feature_names=['review_count', 'positive', 'review_avg', 'review_sum',
                                                 'tip_count', 'likes', 'tip_avg', 'tip_sum'],
                                  class_names=['Normal', 'VIP'])
    graph2 = graphviz.Source(graph2)
    graph2.render('team08_entropy', view=True)
```

,Figure 2.1.1

2.2 Query 분석

R1에서 반환 받은 결과로부터 2017년 VIP 선정 기준에 대한 의사결정 나무를 생성해야 한다. Node impurity측정 방식을 gini와 entropy두 가지로 하여 의사 결정 나무 생성해야 한다. 또한, 이 결과를 graphviz로 저장해야 한다.

2.3 문제 해결 방향성

우선 fetchall 함수를 이용해서 R1에서 구한 데이터값을 저장했고, 각 데이터마다 classes에는 vip에 해당하는 index 1번 값을, feature에는 총 8개의 값이 들어가야 하므로 매 row마다 feature_row라는 빈 list를 만들어 그 안에 8개의 값을 넣은 후, feature_row를 features에 넣는 방식으로 데이터를 입력했다. 그 후에는 주어진 조건대로 DT를 학습시키고 graphviz로 시각화한 결과 다음과 같이 의사결정 나무가 형성되었다.

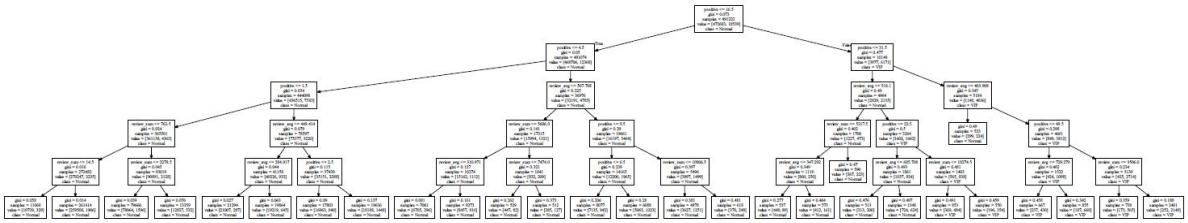


Figure 2.3.1 criterion gini

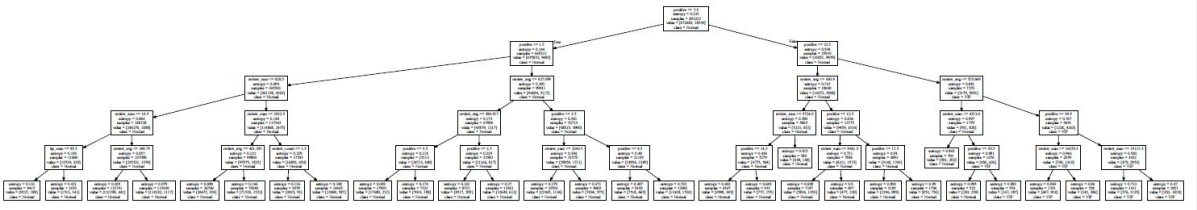


Figure 2.3.2 criterion entropy

3) R3 Analysis

3.1 Solution

R3를 수행하기 위한 개괄적인 코드는 다음과 같다.

```
# R3
    cursor.execute("""
            SELECT user_id, if(year=2017,1,-1) as vip, ifnull(review_count,0)+ifnull(tip_count,0) as quantity,
                ifnull(positive,0)+ifnull(Tip_likes,0) as quality
            FROM (SELECT U1.id as user_id, review_count, positive
                  FROM user as U1
                  LEFT JOIN(    SELECT user_id as idR, COUNT(*) as review_count,
                                    SUM(R1.useful)+SUM(R1.funny)+SUM(R1.cool) as positive
                                FROM review as R1
                                WHERE R1.date like '%2017%'
                                GROUP BY user_id)R
                  ON U1.id=R.idR)UR
            INNER JOIN( SELECT id as user_id2, tip_count, Tip_likes
                        FROM user as U2
                        LEFT JOIN(  SELECT user_id as idT, COUNT(*) as tip_count, SUM(T1.likes) as Tip_likes
                                    FROM tip as T1
                                    WHERE T1.date like '%2017%'
                                    GROUP BY T1.user_id)T
                        ON U2.id=T.idT)UT
            ON UR.user_id=UT.user_id2
            LEFT JOIN(SELECT user_id as idV, year
                      FROM vip_history as V
                      WHERE V.year=2017)V1
            ON UR.user_id=V1.idV
            WHERE review_count is not null or tip_count is not null
            """)
```

,Figure 3.1.1

```
rows2 = cursor.fetchall()

classes2=[]
features2=[]
for row in rows2:
    features_row2 = []
    classes2.append(row[1])
    for i in range(2,4):
        features_row2.append(row[i])
    features2.append(features_row2)

DT3 = tree.DecisionTreeClassifier(criterion='gini', max_depth=6, min_samples_leaf=500)
DT3.fit(X=features2, y=classes2)

graph3 = tree.export_graphviz(DT3, out_file=None,
                              feature_names=['quantity', 'quality'],
                              class_names=['Normal', 'VIP'])
graph3 = graphviz.Source(graph3)
graph3.render('team08_Req3', view=True)

cursor.close()
```

,Figure 3.1.2

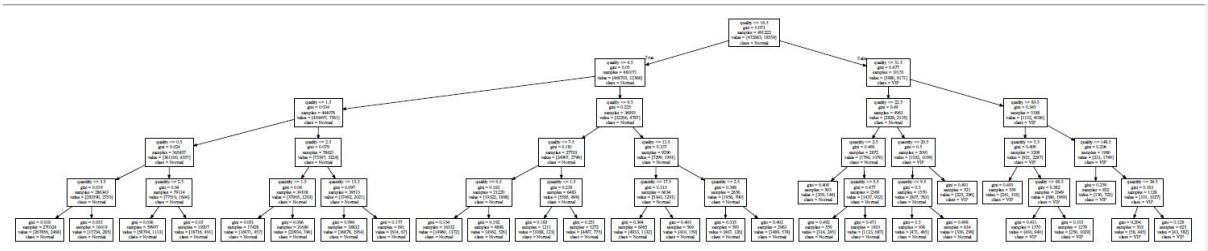
3.2 Query 분석

2017년의 VIP선정 여부에 대한 의사결정 나무를 만들고 생성된 나무를 R2에서 생성한 의사결정 나무와 비교해 본다.

3.3 문제 해결 방향성

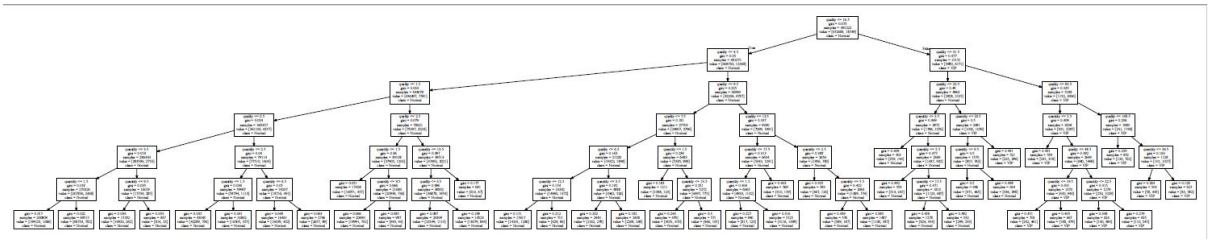
주어진 requirement의 R1은 vip 선정 기준에 대한 의사결정 나무를 만들고자 따로 뽑아낸 query이다. 그런데 대상 user들은 개인이 쓴 tip 또는 review가 하나 이상이어야 한다는 점에서, tip또는 review를 쓰는 그 '활동' 자체가 vip 선정의 선결조건 중 하나가 아닐까라는 생각을 해서, 그 선결 조건의 양과 질만을 따져본다면 의사결정 나무가 어떻게 달라질 지 궁금했다.

따라서 tip의 수와 review의 수의 합을 quantity, tip이 받은 likes와 review가 받은 positive의 합을 quality로 두고, 필요한 attribute만을 포함한 query를 다시 생성하고 R2와 같은 조건 및 과정으로 DT를 '처음' 시각화 해 본 결과는 다음과 같다.



,Figure 3.3.1

그러나 이미 전체적으로 보기에도 R2에서의 DT보다 분류 수가 적어 그다지 유의미하지 않을 것으로 보여, 조금 더 세분화하고자 max depth를 6으로 잡고 '최종적으로' 시각화한 결과는 다음과 같다.



,Figure 3.3.2

R2에서 gini index에 기반한 gini split을 계산한 결과, 0.0524가 나오는데 R3에서 gini index에 기반한 gini split을 계산하면 0.0548정도가 나온다. 따라서 성능 면에서 quality와 quantity만을 feature로 둔 R3에서의 DT와 R2의 DT가 큰 차이가 없다. 가독성 면에서는 거시적으로 봤을 때 R3가 depth가 6이기 때문에 조금 더 세부적으로 나뉘어 있어 가독성이 좋지 않다고 할 수 있으나 판정 기준의 내용들을 보면 R2는 판정 기준이 되는 feature가 불규칙적인데 반해 R3에서는 판정 기준이 초반 3단계정도까지 quality, 그 이후에는 소수의 예외를 제외하고 전부 quantity이기 때문에 조금 더 가독성이 좋다고 할 수 있을 것이다.

이러한 차이가 발생한 이유는 R3에서 R2보다 vip 선정에 조금 더 핵심적인 성질만 뽑아내는 것에 성공했기 때문이라고 생각한다. 또한, 2개의 feature로 DT를 생성했기 때문에 분류 방식에서의 가독성 측면에서 더 좋았을 것으로 보인다.

4) R4 Analysis

4.1 Solution

R4를 수행하기 위한 개괄적인 코드는 다음과 같다.

```
    # LVW_liked
    cursor.execute('''
        CREATE VIEW LVW_liked AS
        SELECT DISTINCT lower(business.name) AS name, review.user_id
        FROM business
        INNER JOIN review
        ON review.business_id=business_id
        WHERE review.stars>=3 AND business.neighborhood='Westside' AND city='Las Vegas'
        ''')
# LVW_liked_partial_users
    cursor.execute('''
    CREATE VIEW LVW_liked_partial_users AS
    SELECT DISTINCT lower(business.name) AS name, review.user_id
    FROM business
    INNER JOIN review
    ON review.business_id=business.id
    WHERE review.stars>=3 AND business.neighborhood='Westside' AND city='Las Vegas' AND review.user_id IN
    (SELECT review.user_id
     FROM review
     INNER JOIN business
     ON review.business_id=business.id
     WHERE business.neighborhood ='Westside' AND city='Las Vegas' AND review.stars >=3
     GROUP BY review.user_id
    HAVING count(review.id) >= 2)''')
```

,Figure 4.1.1

4.2 Query 분석

Requirement 4에서 요구하는 바는 다음과 같이 정리할 수 있다.

첫째, name과 user_id의 column들이 포함되는 'LVW_liked'라는 이름의 view를 생성할 것

둘째, 업소가 속한 neighborhood가 'Westside'일 것

셋째, Westside 내의 업소들에 대해 3점 이상의 리뷰를 2개 이상 남긴 사용자들에 대한 정보만을 남긴 'LVW_liked_partial_users'라는 이름의 view를 생성할 것

넷째, View 'LVW_liked_partial_users'가 가지는 column이 'LVW_liked'의 column과 동일할 것

4.3 문제 해결 방향성

첫 번째 조건을 실현하기 위해 작성한 코드는 다음과 같다.

```
CREATE VIEW LVW_liked AS
SELECT DISTINCT LOWER(business.name) AS name, review.user_id
FROM business
INNER JOIN review
ON review.business_id=business_id
```

, Figure 4.3.1

```
review.stars>=3 AND
```

, Figure 4.3.2

업소 이름의 경우, 소문자로 변환할 것이라는 추가적인 제약이 존재해 lower 함수를 이용하여 조건에 맞는 data를 추출해내었다. 이때, 'DISTINCT' 문을 사용하여 한 사용자가 하나의 업소에 대해서 여러 번 평가를 남겼을 경우, 이를 배제하는 조건을 걸어줌과 동시에 user와 business, review schema 각각에 user라는 동명의 attribute가 있기 때문에 따로 alias 작업을 거치지 않고, business.user로 구분해주도록 한다. 다음으로, requirement에서 요구한 바를 실현하기 위해 lower 함수가 내뱉는 값의 경우는 name으로 alias 작업을 수행해주었다. 마찬가지로, user_id의 경우 review 항목에 존재하는 사용자의 id를 추출해야 하는데, 이때 다른 schema에 동일한 이름의 attribute가 존재하기 때문에, business.user와 유사하게 review.user_id로 구분하도록 한다. 한편, 상단에 작성된 코드에서 확인할 수 있듯이 업소의 name과 review를 작성한 사용자의 아이디(review.user_id)의 column들이 포함되는 view를 생성하기 위해 'CREATE VIEW + 이름'의 구조를 사용하였다. 한편, 상단의 Figure 4.3.2는 WHERE문의 일부로, 추출하는 사용자의 id가 해당 업소에 별점 3점 이상의 리뷰를 남긴 사람들의 것으로 정의되어 있기에 business schema에 존재하는 업소의 id가 review 스키마에 존재하는 별점 3점 이상을 받은 업소의 id와 일치하는 조건을 만족시키는 것으로 구현할 수 있다.

두 번째 조건을 실현하기 위하여 작성된 코드는 다음과 같다.

```
WHERE business.neighborhood ='Westside' AND city='Las Vegas'
```

, Figure 4.3.3

업소가 속한 neighborhood가 'Westside'를 만족해야 하므로, business table의 neighborhood attribute의 값이 Westside와 동일하다는 조건을 활용하면 된다.

세 번째 조건을 실현하기 위하여 작성된 코드는 다음과 같다.

```
CREATE VIEW LV_liked_partial_users AS
SELECT DISTINCT lower(business.name) AS name, review.user_id
FROM business
INNER JOIN review
ON review.business_id=business.id
WHERE review.stars>=3 AND business.neighborhood='Westside' AND city='Las Vegas' AND review.user_id IN
(SELECT review.user_id
 FROM review
 INNER JOIN business
 ON review.business_id=business.id
 WHERE business.neighborhood ='Westside' AND city='Las Vegas' AND review.stars >=3
 GROUP BY review.user_id
HAVING count(review.id) >= 2)''')
```

, Figure 4.3.4

개괄적인 코드는 첫 번째 조건과 크게 다를 바가 없으나, 위의 경우에는 SELECT문을 복문으로 사용하여 나타내었다. Where문까지의 의미는 동일하나 세 번째 조건이 요구하는 리뷰를 두 개 이상 남긴 사용자들에 대한 정보만을 남기기 위해 'GROUP BY' 문과 'HAVING count(*) >=2' 문을 사용하였다. 이때, 첫 번째 조건과 두 번째 조건은 세 번째 조건을 실현하기 위한 전제조건에 해당하며, 이에 대한 자세한 설명은 이미 위에서 언급했으므로 생략하도록 한다.

네 번째 조건을 실현하기 위하여 작성된 코드는 Figure 4.3.1과 Figure 4.3.4를 활용하도록 한다. 두 그림의 SELECT문을 확인하면, 도출하고자 하는 변수가 모두 name과 user_id임을 확인할 수 있다. 이를 MySQL Workbench를 이용하여 확인하면, 아래 그림과 같이 column들이 형성됨을 확인할 수 있다.

name	user_id
little caesars pizza	OVdToPHAp_AhNOAbYiijww
little caesars pizza	Rfjwn3Rx5sjvZKA6jbLZjA
little caesars pizza	Unhr6Ut9xQowzzohevNdxg
little caesars pizza	Z4a9QmzKt7dB4Ju59yQOMA
little caesars pizza	4PFwh5zbgyPhTGRtxkSZZA
mcdonalds	duFV7fa27xPp33iNlwZRhA
mcdonalds	mvS2UIMT_UhnzrfcDDo6uA
mcdonalds	yh1cYqWWUfwMHp_4a8t51A
mcdonalds	604vXbJUsR5glg15zbFe1Q
mcdonalds	yH1rIEvxXPb_YCmRQVRdBw
mcdonalds	THQIu1BODr6Cjexz9GT0KQ
mcdonalds	PKEzKWv_FktMm2mGPjwd0Q
mcdonalds	1DVCkL1W6TGzXcu8ZbBptA
mcdonalds	Sr2Vvrnymu6nIQNDkXPcrg
mcdonalds	4PFwh5zbgyPhTGRtxkSZZA
mcdonalds	CZs1Eq7t1fqpXV1yox-CVg

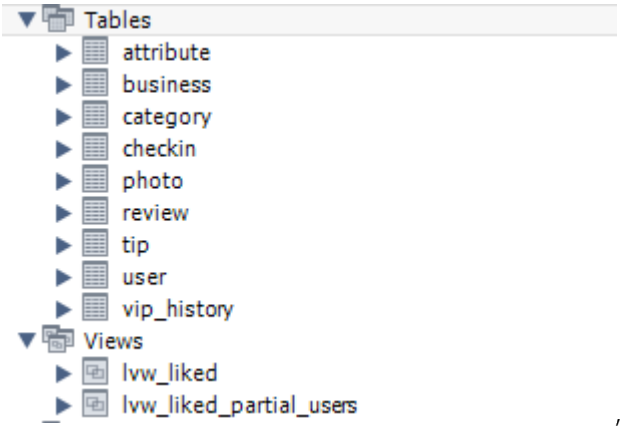
,Figure 4.3.5

name	user_id
little caesars pizza	OVdToPHAp_AhNOAbYiijww
little caesars pizza	Rfjwn3Rx5sjvZKA6jbLZjA
little caesars pizza	Unhr6Ut9xQowzzohevNdxg
little caesars pizza	Z4a9QmzKt7dB4Ju59yQOMA
little caesars pizza	4PFwh5zbgyPhTGRtxkSZZA
mcdonalds	duFV7fa27xPp33iNlwZRhA
mcdonalds	mvS2UIMT_UhnzrfcDDo6uA
mcdonalds	yh1cYqWWUfwMHp_4a8t51A
mcdonalds	604vXbJUsR5glg15zbFe1Q
mcdonalds	yH1rIEvxXPb_YCmRQVRdBw
mcdonalds	THQIu1BODr6Cjexz9GT0KQ
mcdonalds	PKEzKWv_FktMm2mGPjwd0Q
mcdonalds	1DVCkL1W6TGzXcu8ZbBptA
mcdonalds	Sr2Vvrnymu6nIQNDkXPcrg

,Figure 4.3.6

4.4 조건 만족 여부 검토 및 결과 분석

위의 코드들을 실행하면, MySQL Workbench에는 다음과 같은 그림이 추가된다.



,Figure 4.4.1

첫 번째 조건이 만족되었는가의 여부는 Figure 4.3.5를 통해서 확인해보도록 하겠다. 위 그림은 MySQL Workbench 상에 존재했던 기존 schema의 view 영역에 새로 추가된 부분을 table로 도출해낸 것의 일부에 해당하는데, name과 user_id가 포함되는 column을 결과 값으로 산출해냄을 확인할 수 있었다. 이때, Figure 4.3.5는 Figure 4.4.1의 view 중 'LVW_liked' 실행했을 때 얻을 수 있는 data set이다. 마찬가지로 Figure 4.3.6은 Figure 4.4.1을 통해 view 'LVW_liked_partial_users'를 실행했을 때 얻을 수 있는 data set으로, 첫 번째부터 세 번째 조건을 모두 충족시켰음을 확인할 수 있다. 마지막으로, 네 번째 조건의 경우 두 view가 같은 column의 형태를 띠고 있다는 점에서 R4의 모든 조건이 만족되었음을 입증하는 바이다.

5) R5 Analysis

5.1 Solution

R5의 개괄적인 코드는 다음과 같다.

```
cursor.execute( `select * from LVW_liked_partial_users` )
store_name = []
user_id = []
for row in cursor:
    store_name.append(str(list(row)[0]))
    user_id.append(str(list(row)[1]))

df = pd.DataFrame({'store_name': store_name, 'user_id': user_id})
df = df.drop_duplicates()
df_final = df.groupby(['user_id', 'store_name']).size().unstack(fill_value=0)
```

,Figure 5.1.1

5.2 Query 분석

LVW_liked_partial_users를 horizontal table로 만든 결과를 pandas DataFrame으로 저장하라

5.3 문제해결 방향성

TA 세션에서 사용하신 sql의 select, max 문을 사용해야한다는 강제 조건이 없어 python의 pandas 패키지를 사용해 데이터를 horizontal화 했다.

horizontal dataframe을 만들기 위해 LVW_liked_partial_users view의 데이터를 모두 불러와 가게명은 store_name list, user id는 user_id list에 저장했다. for문을 활용해 cursor의 row를 각 list에 append하는 방식을 사용했다.

* process

- store_name, user_id list를 pandas의 DataFrame 'df'로 구성
- R4의 distinct 문으로 중복값이 제거 되었지만, 확인차 drop_duplicate를 통해 df의 중복값 제거
- df를 분석에 사용할 horizontal dataframe 'df_final' 으로 변환: groupby와 unstack 사용

> 위의 df를 'user_id', 'store_name'으로 groupby.size() 한 결과는 각 (user_id, store_name) 조합의 갯수에 대해 (user_id, store_name, 갯수)로 표현된다. 중복값을 제거했기 때문에 갯수는 0 또는 1의 값을 가진다.

```
In [66]: s3
Out[66]: one  a    0
             b    1
             c    2
             d    3
         two  c    4
             d    5
             e    6
         dtype: int64
```

, Figure 5.3.1

```
In [67]: s3.unstack()
```

Out[67]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

, Figure 5.3.2

> unstack은 계층적 인덱스에서 최하위 index를 column으로 올려주는 함수이다.(위 figure참고) fill_value = 0 조건을 추가해 값이 NaN인 경우 0으로 변환해 주었다.

5.4 조건 만족 여부 검토 및 결과 분석

동 과정을 통해 store_name, user_id의 vertical data를 horizontal data 양식으로 변환했다.

store_name	user_id
paymons mediterranean cafe & hookah lounge	vvxlVr_OR6sHBzGPG9dRXg
superb maids	2xoQviFaWZyVtvcxUsD1Xw
dominos pizza	cFGAElozHTdQDfHimOiSBQ
cold stone creamery	z92mH_mzx27KpxWdBK9jog
capos italian cuisine	tp4Pf8EPruRnkBZ1RLtjpw
kona grill	5DuZbOCtUECYoWX7HPfMoA
capos italian cuisine	Mmrj5j6NYxTHHx_QlsyouA
sofias cafe	OfUaF8Amqzb_GaI01Mumlg

, Figure 5.4.1

store_name	1 reef	1 world medicine	100.5 jack fm	21 restaurant & lounge	221 wine bar	24 hour cleaners	24 hour fitness	24 hour fitness - las vegas rainbow	s
user_id									
---1lKK3aKOuomHnwAkAow	0	0	0	0	0	0	0	0	
--0kuuLmuYBe3Rmu0lycww	0	0	0	0	0	0	0	0	
--HCoE1ghaAlcaAfshlCgw	0	0	0	0	0	0	0	0	
--LUapetRSkZpFZ2d-MXLQ	0	0	0	0	0	0	0	0	
--RlSfc-QmcHFGHyX6aVjA	0	0	0	0	0	0	0	0	
-05XqtNjcBq19vh2CVJN8g	0	0	0	0	0	0	0	0	
-0Hf2jiBo7hJdxYW0Y6PKQ	0	0	0	0	0	0	0	0	

, Figure 5.4.2

6) R6 Analysis

6.1 Solution

R6의 개괄적인 코드는 다음과 같다.

```
frequent_itemsets = apriori(df_final, min_support=0.005, use_colnames=True)
print(frequent_itemsets)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
print(rules.to_string())

rules.to_csv('team08_association.txt', header=True, index=True, sep='₩t')

cursor.close()
```

, Figure 6.1.1

6.2 Query 분석

Requirement 6에서 요구하는 바는 다음과 같이 정리할 수 있다.

첫째, frequent itemset의 최소 support는 0.005일 것

둘째, 연관 분석 metric은 lift이되, lift가 1보다 같거나 큰 것들을 출력할 것

셋째, 결과 파일 명은 team##_association.txt 형태로 저장할 것

6.3 문제해결 방향성

첫 번째 조건을 실현하기 위해 작성한 코드는 다음과 같다.

```
frequent_itemsets = apriori(df_final, min_support=0.005, use_colnames=True)
print(frequent_itemsets)
```

, Figure 6.3.1

support가 min_support를 넘는 itemset을 생성하는 코드이다. 이때 use_colnames는 DataFrame의 column 명을 item의 이름으로 활용할 것인지에 대한 여부를 나타내는 것이며 위 코드에서는 True 값으로 지정해주었다. 또한 frequent itemset의 최소 support는 0.005이어야 하므로 'min_support=0.005'라는 제약을 추가하였다.

두 번째 조건을 실현하기 위해 작성한 코드는 다음과 같다.

```
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
print(rules.to_string())
```

, Figure 6.3.2

metric은 lift로 지정이 되어 있으며, metric이 min_threshold를 넘는 rule들을 생성해야 한다. 이때 to_string() 함수는 Dataframe을 출력 가능한 형태로 변환하는 역할을 담당하며, 특정 조건을 만족하는

규칙을 추출하는 데 활용된다. 조건 상, lift 값이 1보다 같거나 커야 하므로 'min_threshold=1'이라는 제약조건을 추가하였다.

세 번째 조건을 실현하기 위해 작성한 코드는 다음과 같다.

```
rules.to_csv('team08_association.txt', header=True, index=True, sep='₩t')
```

Figure 6.3.3

결과 파일 명은 'team##_association.txt'의 형태를 띠고 있어야 하므로, header와 index, sep을 지정해주는 코드를 작성함으로써 Figure 6.3.4와 같은 txt를얻을 수 있었다.

team08_association - 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	frozenset({'echo & rig'})	frozenset({'bachi burger'})	0.060996082820369335	0.034321954859168065	0.005098551265311198				0.08358817
1	frozenset({'bachi burger'})	frozenset({'echo & rig'})	0.034321954859168065	0.060996082820369335	0.005098551265311198				0.14855072
2	frozenset({'bachi burger'})	frozenset({'honey salt'})	0.034321954859168065	0.04228066902940993	0.005222906174221227				0.15217391
3	frozenset({'honey salt'})	frozenset({'bachi burger'})	0.04228066902940993	0.034321954859168065	0.005222906174221227				0.12352941
4	frozenset({'herbs & rye'})	frozenset({'echo & rig'})	0.032456631225517625	0.060996082820369335	0.005906858173226388				0.18199233
5	frozenset({'echo & rig'})	frozenset({'herbs & rye'})	0.060996082820369335	0.032456631225517625	0.005906858173226388				0.09683995
6	frozenset({'echo & rig'})	frozenset({'honey salt'})	0.060996082820369335	0.04228066902940993	0.008953553441522105				0.14678899
7	frozenset({'honey salt'})	frozenset({'echo & rig'})	0.04228066902940993	0.060996082820369335	0.008953553441522105				0.21176470
8	frozenset({'sambalatte torrefazione'})	frozenset({'echo & rig'})	0.03307840577006777	0.060996082820369335	0.005471615992041286				0
9	frozenset({'echo & rig'})	frozenset({'sambalatte torrefazione'})	0.060996082820369335	0.03307840577006777	0.005471615992041286				0
10	frozenset({'hash house a go go'})	frozenset({'egg & i'})	0.05931729155008394	0.03469501958589815	0.006777342535596593				0
11	frozenset({'egg & i'})	frozenset({'hash house a go go'})	0.03469501958589815	0.05931729155008394	0.006777342535596593				0

,Figure 6.3.4 team08_association.txt 일부

6.4 조건 만족 여부 검토 및 결과 분석

결과로 도출된 텍스트 파일을 살펴보면, support는 모두 0.005 이상이며, lift의 값 역시 모두 1을 넘으므로 조건 1과 2는 충족되었다. 또한, 결과 파일의 경우, Figure 6.3.4를 참조하면, 'team##_association.txt'의 형태를 띠고 있음을 확인할 수 있다는 점에서 requirement 6는 성공적으로 수행되었다.

한편, Lift의 경우 변수들 간의 연관성을 파악하는 지표로 사용되고는 하는데, 데이터가 어떠한 방식으로 구성되어 있는지에 대한 정확한 정보는 없으나, lift가 대다수 2를 넘는 값을 산출해내는 것으로 미루어 볼 때, 변수들 간에는 강한 양의 상관관계가 있음을 추론할 수 있다. Support의 경우, LHS/(LHS+ RHS)로 정의되는데, 어떠한 의미를 지닌다기 보다는 사전에 설정한 minsup을 넘는 데이터의 집합만을 확인함으로써, 방대한 양의 데이터를 처리하는 데 드는 시간과 비용을 감축할 수 있다.

3. 요약 및 결론

의사 결정 나무(DT)의 경우 예측력은 비록 다른 지도 학습 기법들에 비해 떨어지며, record의 수가 일정 수준 이상 확보되지 않는 경우 통계적으로 유의하지 않을 수 있다는 단점이 있지만 해석이 직관적이고, 분류 대상에 대한 사전 지식이 요구되지 않기 때문에 보다 빠르고 간단한 구성이 가능하다는 점에서 실생활에서도 많이 활용된다. 예컨대, 마케팅 분야에서 고객의 이동 경로를 의사

결정 나무를 통해 분석한 후 고객이 자주 구매하는 물품을 추천하는 시스템을 구성하는 알고리즘 역시 의사 결정 나무에 기반하는 경우가 더러 존재한다. 한편, record가 풍부하여 leaf와 node가 수준 이상으로 확보된다면 보다 고차원적인 분석이 가능해지는데 은행권에서 고객을 관리하는 방식이 바로 그러한 예에 해당한다. 신용상태, 월소득, 직업, 연령 등 다양한 데이터는 고객이 대출을 받을만한 신용이 있는 사람인지, 위험성이 어느 정도 존재하는지 타당성을 평가하고, 예측하는 데 활용될 수 있다.

연관 분석 역시 데이터 분석에 있어서 매우 유의미한 기법 중 하나이다. 이는 transaction set이 주어졌을 때 특정 item의 존재로부터 다른 item의 발생 정보를 예측하는 법칙으로, 각 가치 있는 데이터 간의 연관성을 확인함으로써 미처 예상치 못한 결론을 미리 내다볼 수 있으며, 나아가 시각화된 정보를 통해 사람들을 설득하고 이해시키는데 활용될 수 있다. 연관 분석은 크게 지지도(support), 신뢰도(confidence), 향상도(lift)를 통해서 이해할 수 있는데, 데이터의 환경에 따라 각 평가 척도는 한계가 존재할 수 있으므로, 데이터의 구성 형태에 주의해야 한다. 한편, 연관분석은 계산이 용이하고, 사용이 편리한 분석 데이터의 형태를 지니고 있으며 목적변수가 없기에 그 유용성은 매우 크다. 하지만, 품목 수의 증가는 계산의 기하급수적 증가를 의미하며, 세분화된 품목은 의미 없는 분석 결과를 도출할 가능성이 커진다. 또한 품목 간 비율 차이가 크다면 규칙발견이 쉽지 않기에 이 또한 염두에 두고 데이터의 형태에 따라 적절한 평가 방법을 사용해야 할 것이다.