Data Science Assignment #1

2018062733 윤동빈

1. Summary of your algorithm

input data로부터 frequent patterns, association rules를 찾기 위해 Apriori algorithm을 구현했다. 알고리즘의 전체적인 흐름은 다음과 같다.

1) input data의 모든 transaction에서 size가 1인 itemsets 추출 (1-itemset = C_1)
2) C_1에서 minimum support를 넘는 itemsets를 모아 L_1에 저장
3) $L_k (k \geq 1)$에서 각 itemset의 subsets의 조합을 통해 길이가 $k+1$인 C_{k+1} 생성
4) C_{k+1}의 각 itemset의 support를 database의 transactions를 참조해 얻은 후, support가 minimum support를 넘는 itemsets 중 모든 subsets가 L_{len}에 속하는 itemsets만 L_{k+1}에 저장 (len : 해당 itemset의 size)
5) C_{k+1}이나 L_{k+1}의 size가 0이면 종료하고 그렇지 않으면 3)으로 돌아감

2. Detailed description of your codes (for each function)

```
# 부분 집합 frequent 여부 확인
def checkSubsetIsFrequent(L, arr):

    for i in range(len(arr)):

        if i == 0:
            continue

        subsets = list(combinations(arr, i))
        for subset in subsets:

            s = list(subset)
            flag = False
            for l in L[len(s)-1]:
                if set(s).issubset(set(l)):
                    flag = True

            if flag is False:
                return False

    return True
```

checkSubsetIsFrequent(L, arr)

String *arr*의 모든 부분집합(*subsets*)가 L에 속하는지 즉, frequent pattern에 속하는지 확인하는 함수로 True or False를 return 한다.

```python
# 리스트 내 조합 중 길이가 1만큼 더 긴 조합들을 찾아 반환
def combination(C, L, k):

    ret = []

    cnt = 0
    for fir in range(len(C)-1):
        cnt = cnt+1

        sec = fir + 1
        while True:

            if sec >= len(C):
                break

            uni = set(C[fir]) | set(C[sec])

            if len(uni) == k+1 and checkSubsetIsFrequent(L, list(uni)):
                ret.append(list(uni))

            sec = sec + 1

    return ret
```

combination(C, L, k)

인자로 들어오는 C는 C_k와 같음
(C_k : size가 k인 itemsets 모음)

C의 모든 조합 중,
길이가 $k+1$이고 frequent pattern인 itemset만 모아서 return

frequent pattern 여부는
checkSubsetIsFrequent()를 통해 확인

```python
input_min_sup, input_name, output_name = input().split()
min_sup = float(input_min_sup)

f = open(input_name, 'r')
DB = []

while True:

    line = f.readline()
    if not line:
        break

    numbers = re.findall(r'\d+', line)

    transaction = []
    for num in numbers:
        transaction.append(int(num))

    DB.append(transaction)

f.close()
```

minimum support, input file name,
output file name 입력받은 후,

input file에 있는 transactions를
database(*DB*)에 저장

```python
C = []
L = []
cnt = []    # C[k] itemset count

# DB에서 1-itemset 추출
one_cnt = {}
one_itemsets = set()
for ts in DB:
    for num in ts:

        one_itemsets.add(num)

        if num in one_cnt:
            one_cnt[num] = one_cnt[num] + 1
        else:
            one_cnt[num] = 1
```

```python
one_list_items = []
tmp = list(one_itemsets)
index = 0
for t in tmp:
    if (one_cnt[index] * 100 / len(DB)) >= min_sup:
        new_list = []
        new_list.append(t)
        one_list_items.append(new_list)
    index = index + 1

C.append(one_list_items)
L.append(one_list_items)

list_one_cnt = []
for num in one_list_items:
    list_one_cnt.append(one_cnt[num[0]])
cnt.append(list_one_cnt)
```

database에 저장된
transactions에 속한
itemsets 중 size가
1인 itemset 추출

```python
# L[k]에서 C[k+1] 생성
k = 0
while True:

    # 종료 조건
    if len(C[k]) == 0 or len(L[k]) == 0:
        break

    # C[k+1] 생성
    com = combination(C[k], L, k+1)
    C.append(com)

    # C[k+1]의 각 itemset의 support value 설정
    tmp_cnt = [0 for i in range(len(com))]
    index = 0
    for itemset in com:
        for tr in DB:
            if set(itemset).issubset(set(tr)):
                tmp_cnt[index] = tmp_cnt[index] + 1

        index = index + 1
    cnt.append(tmp_cnt)

    # frequent 여부 확인 후 L[k+1] 갱신
    tmp = []
    index = 0
    for sup in tmp_cnt:
        if (sup * 100 / len(DB)) >= min_sup:
            tmp.append(com[index])
        index = index + 1
    L.append(tmp)

    k = k + 1
```

L_k에서 C_{k+1}, L_{k+1} 생성 과정

C_k나 L_k가 0이면 더 이상 진행하지 않도록 설정

C_k에서 C_{k+1}을 *combination()*을 통해 생성

생성된 C_{k+1}에 속하는 각 itemset의 support 값을 database의 transactions를 참조해서 설정
(*tmp_cnt* : C_{k+1}의 itemset support value List)

C_{k+1}에서 support가 minimum_support를 넘는 itemset을 추출해 L_{k+1}에 저장

```python
# FP 중복 제거
arr = []
for i in range(len(L)):
    chk = {}
    tmp = []
    for j in range(len(L[i])):

        if j in chk:
            continue

        L[i][j].sort()
        tmp.append(L[i][j])

        for k in range(j+1, len(L[i])):
            if set(L[i][j]) == set(L[i][k]):
                chk[k] = True

    arr.append(tmp)
```

길이가 i인 frequent patterns 집합을 L_i이라 하면,
각 L_i에 속한 frequent patterns의 중복 제거

중복 제거된 내용은 *arr*에 저장

```
f = open(output_name, 'w')

for r in range(len(arr)):
    for c in range(len(arr[r])):
        for r2 in range(len(arr)):
            for c2 in range(len(arr[r2])):

                # 서로의 subset인 경우 제외
                if set(arr[r][c]).issubset(set(arr[r2][c2])) or set(arr[r2][c2]).issubset(set(arr[r][c])):
                    continue

                # support, confidence 값 얻기
                sup_cnt = 0
                conf_cnt = 0
                conf_all = 0

                union_set = set(arr[r][c]).union(set(arr[r2][c2]))

                for tr in DB:

                    tr_set = set(tr)
                    if union_set.issubset(tr_set):
                        sup_cnt = sup_cnt + 1
                        conf_cnt = conf_cnt + 1

                    if set(arr[r][c]).issubset(tr_set):
                        conf_all = conf_all + 1

                # association rule이 의미없는 경우 제외
                if conf_cnt == 0 or conf_all == 0:
                    continue

                # association rule : A->B
                A = '{'
                B = '{'

                index = 0
                for item in arr[r][c]:
                    if index < len(arr[r][c])-1:
                        A += str(item) + ','
                    else:
                        A += str(item)
                    index = index + 1
                A += '}'

                index = 0
                for item in arr[r2][c2]:
                    if index < len(arr[r2][c2])-1:
                        B += str(item) + ','
                    else:
                        B += str(item)
                    index = index + 1
                B += '}'

                sup = sup_cnt / len(DB) * 100
                conf = conf_cnt / conf_all * 100

                if sup < min_sup:
                    continue

                f.write(A + '\t' + B + '\t' + str(f"{sup:.2f}") + '\t' + str(f"{conf:.2f}") + '\n')
f.close()
```

*output.txt*를 작성하는 과정으로,

*arr*에 있는 모든 freqeunt patterns 간에 존재하는 association rules를 얻기 위해 모든 조합을 만들어 support, confidence 계산 후 minimum support를 넘는 association rules만 *output.txt*에 작성했다.

3. Instructions for compiling your source codes with testing

```
C:₩Users₩동비니₩Desktop₩Python₩Workspace>python apriori.py 5 input.txt output.txt

C:₩Users₩동비니₩Desktop₩Python₩Workspace>
```

▶ python apriori.py 5 input.txt output.txt

output.txt

output.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

{3,8,16}	{3,7}	6.20	25.83
{3,8,16}	{3,9}	7.20	30.00
{3,8,16}	{3,10}	7.00	29.17
{3,8,16}	{3,11}	6.40	26.67
{3,8,16}	{3,12}	6.40	26.67
{3,8,16}	{3,13}	7.40	30.83
{3,8,16}	{3,14}	5.40	22.50
{3,8,16}	{3,15}	5.80	24.17
{3,8,16}	{3,17}	5.80	24.17