Apriori Algorithm

(Assignment #1)

2016025678 이강민

1. Summary of your algorithm

Apriori 알고리즘은 frequent itemset을 찾고 이를 이용하여 연관 규칙 탐색을 하는 알고리즘입니다. 제가 구현한 Apriori 알고리즘은 먼저 입력받은 input file로부터 minimun support를 만족하는 frequent itemset을 구합니다. 그 후에 만들어진 frequent itemset을 인자로 받는 association rule 함수를 실행합니다. association rule 함수는 frequent itemset에서 나올 수 있는 모든 pattern의 support 값과 confidence 값을 구하고 반환해줍니다. 최종적으로 만들어진 내용을 output file에 작성해줍니다.

2. Detailed description of your codes

1) read_input 함수

```
def read_input(input_file):
    f = open('./'+input_file)
    lines = f.readlines()
    a = []
    for i in range(0,len(lines)):
        a.append(list(map(int,lines[i].strip().split('₩t'))))
    return a
```

- read_input 함수는 입력받은 input file을 열고 그 안에 있는 모든 값을 list에 추가하여 반환합니다.

2) scan 함수

```
def scan(tdb,c,min_sup):
    cand = defaultdict(int)
    for i in tdb:
        for j in c:
            if type(j) == int:
                if {j}.issubset(i):
                    cand[j]+=1
            elif set(j).issubset(i):
                if len(j)==2:
                    cand[tuple(sorted(j))] += 1
                else:
                    cand[tuple(j)] += 1
    ret = dict()
    for i,j in cand.items():
        if j/len(tdb) >= min_sup:
            ret[i] = j/len(tdb)
    return ret
```

- scan 함수는 frequent itemset을 생성하는 함수입니다. item이 하나인 경우 set 함수로 set을 만들 수 없어 item의 길이가 1인 경우와 길이가 2이상인 경우로 나누어 실행합니다. 마지막으로 subset을 확인하고 나서 minimum support 조건에 만족하면 frequent itemset에 추가시킵니다.

3) join_cand 함수

```
def join_cand(c,k):
    new = []
    ret = []
    if k == 2:
        new = list(combinations(list(c.keys()),k))
    else:
        for i in c.keys():
            for j in c.keys():
                if len(set(i).union(j)) == k:
                    new.append(tuple(sorted(set(i).union(j))))
    new = list(set(new))
    return  new
```

- join_cand 함수는 item 길이가 k-1인 frequent itemset으로부터 길이가 k인 frequent itemset을 생성하는 함수입니다. 이 함수도 마찬가지로 item 길이가 2인 경우와 길이가 3 이상인 경우로 나누어 실행됩니다. 길이가 2인 경우에는 combination 함수를 이용하여 조합을 만들어내고 길이가 3 이상인 경우에는 union 함수를 이용하여 조합을 생성합니다.

4) associate_rule 함수

```
def associate_rule(total):
    result = ''
    for length,item in total.items():
        if length >= 2:
            for i in item.keys():
                item_set = [combinations(i,j) for j in range(1,length)]
                for element in map(list,chain(*item_set)):
                    comple = set(i) - set(element)
                    support = sup(total,i)
                    confidence = sup(total,i) / sup(total,element)
                    result += '%s\t%s\t%.2f\t%.2f\n' % ('{{{}}}'.format(','.join(map(str,element))),'{{{}}}'.format(','.join(map(str,comple))),round(support*100,2),round(confidence*100,2))
    return result
```

- associate_rule 함수는 frequent itemset의 모든 pattern들의 support 값과 cofidence 값을 구하여 반환하는 함수입니다. 각 pattern 들의 support를 sup 함수를 통하여 구하고 차집합을 이용하여 반대 집합을 구합니다. 두 집합의 조건부 확률을 구하여 형식에 맞게 string에 저장합니다.

5) sup 함수

```
def sup(total,element):
    ori = 0
    if len(element)==1:
        temp = element[0]
        ori = total[len(element)][temp]
    else:
        temp = tuple(sorted(element))
        ori = total[len(element)][temp]
    return ori
```

- sup 함수는 해당 element의 support 값을 반환해주는 함수입니다. 제가 구현한 frequent itemset은 key가 tuple인 dictionary이기 때문에 element의 길이에 따라 나누어 구해주게 됩니다.

6) write_output 함수

```
def write_output(output_file,result):
    f = open(output_file,'w')
    f.write(result)
    f.close()
```

- write_ouput 함수는 associatea_rule 함수에서 만들어진 결과를 output file에 write하는 함수입니다.

3. Instructions for compiling your sources coeds at TA's computer

- python으로 구현하였습니다.