Assignment 1: Apriori Algorithm

2019094511 김준표

1. Summary of algorithm

Apriori 알고리즘은 DB를 스캔한 후 item set의 길이를 1씩 늘려가며 후보 item set을 정하여 frequent item set을 찾는 알고리즘이다. 후보 item set을 정하는 과정에서 self-join과 pruning을 실시하며 입력 받은 minimum support를 기준으로 후보 선정 또는 탈락을 실시한다.

해당 알고리즘은 후보 아이템이 더 이상 없을 때까지 후보 생성 및 frequent item set을 찾는 일을 반복한다. K 단계에서의 self-join에서는 해당 단계에서 존재하는 item set을 조합하여 (K+1) 길이의 후보 아이템을 선정한다.

이후 pruning을 실시하여 새로 생성된 후보 item의 조합이 K 단계에서의 item set에 없는 경우를 제거한다. 예를 들면 (K=2) 단계에서 {A,B} {B,C} {B,E} {C,E}가 있을 때 (K=3)에서 {B,C,E}는 가능하지만 {A,B,C}는 {A,B}가 없으므로 생성될 수 없다. 이와 같은 경우를 제거해주는 작업이 pruning이다.

위의 작업을 모두 시행하여 새로운 item set을 생성했다면 해당 item set의 빈도수를 카운트하여 다음 단계로 넘겨줄 DB를 만들어주면 한 사이클이 마무리된다.

2. Code description

```python
def main():
    minimum_support = int(sys.argv[1])
    input_file = sys.argv[2]
    output_file = sys.argv[3]

    transactions = []
    input_items = {}
    frequent_items = []

    # DB의 transaction 정보들을 transactions table에 담는다
    with open(input_file, "r") as f:
        while True:
            line = f.readline()
            if not line:
                break
            str = line.split("\n")
            # 각 줄에 있는 아이템을 입력받아 transaction에 넣어준다
            items = list(map(int, str[0].split('\t')))
            transactions.append(items)
            # transaction의 아이템 입력받기
            for item in items:
                item = tuple([item])
                if item in input_items:
                    input_items[item] += 1
                else:
                    input_items[item] = 1
    f.close()
    # 퍼센트로 입력받은 minimum support 값을 count로 변환
    minimum_support_cnt = float(minimum_support / 100) * len(transactions)
    # 빈번 아이템 필터링
    frequent_items.append(filtering(input_items, minimum_support_cnt))

    apriori(transactions, frequent_items, minimum_support_cnt)
    association_rule(output_file, frequent_items, transactions)

```

main 함수에서는 input file로부터 받은 item 정보들을 바탕으로 초기 DB를 생성하고 Apriori 함수

를 호출한다.

```python
# Apriori 알고리즘
def apriori(transactions, frequent_items, minimum_support_cnt):
    cnt = 1
    k = 1
    while True:
        # 후보 아이템 생성
        # Step 1: self-join
        # Step 2: pruning
        candidate_items = self_join(frequent_items[k-1], cnt)
        # 다음 후보 아이템이 없는 경우 종료
        if len(candidate_items) == 0:
            break
        # 아이템과 빈도수를 저장하기 위한 딕셔너리
        item_cnt_dict = {}
        for item_set in transactions:
            for item in candidate_items:
                tmp_item = tuple(item)
                if set(tmp_item) <= set(item_set):
                    if tmp_item in item_cnt_dict:
                        item_cnt_dict[tmp_item] += 1
                    else:
                        item_cnt_dict[tmp_item] = 1
        frequent_items.append(filtering(item_cnt_dict, minimum_support_cnt))
        cnt += 1
        k += 1

```

Apriori 함수에서는 Apriori 알고리즘의 전체 틀을 바탕으로 동작하며 [frequent_items]를 바탕으로 후보 item set인 [candidate_items]를 추출하고, [candidate_items]의 길이가 0이 될 때 (후보 아이템이 존재하지 않을 때)까지 일련의 작업을 수행한다. 후보 item set을 구한 뒤 전체 DB를 돌면서 유효한 item set과 빈도수를 저장하고 다음 단계로 넘겨줄 [frequent_items]를 생성한다.

frequent_items는 각 index마다 (index+1)의 길이를 가지는 item set과 해당 item set의 빈도수를 저장하는 dictionary를 모은 list로 구성되어 있다. 그래서 frequent_items[0]에는 길이 1의 item과 빈도수를 저장하고, frequent_items[1]에는 길이 2의 item과 빈도수를 저장하는 식으로 구성된다.

```python
# k+1 빈번 아이템의 후보 생성
def self_join(items, cnt):
    tmp_items = copy.deepcopy(items)
    candidate = []
    for set1 in tmp_items:
        for set2 in tmp_items:
            # 후보 선정을 원활히 하기 위해 union한 결과를 sort하여 후보 군에 추가
            tmp = sorted(list(set(set1).union(set(set2))))
            if len(tmp) == (cnt + 1) and tmp not in candidate:
                candidate.append(tmp)
    return pruning(candidate, tmp_items, cnt)
```

Self_join 함수에서는 K 단계의 item set을 입력 받아 해당 item set의 item들끼리 self-join을 수행한 후 (K+1)의 길이를 가지는 경우 후보군에 추가해준다. Self-join을 구현하기 위해 동일한 item set을 이중 for문으로 돌면서 각 set의 합집합인 union을 실시하고 검사하는 방식으로 후보를 선정했다.

```python
# k 빈번 아이템에 없는 요소를 갖는 아이템 삭제
def pruning(candidate, prev_list, prev_cnt):
    for item in candidate:
        sub_item = list(itertools.combinations(item, prev_cnt))
        flag = True
        for sub in sub_item:
            if sub not in prev_list:
                flag = False
                break
        if flag is False:
            del candidate[candidate.index(item)]
    return candidate
```

Pruning 함수는 self-join을 거쳐 생성된 후보 중에서 K 단계에서 없는 조합 요소를 가지는 item을 삭제해준다. 이를 구현하기 위해 itertools.combination을 이용했으며 해당 모듈은 리스트의 요소를 받아서 parameter로 입력 받은 길이의 조합을 뽑아내는 함수이다.

예를 들어 test = ['A', 'B', 'C']의 리스트가 있고, c = itertools.combinations(test, 2)를 실시한다면, c는 다음의 결과를 갖는다. [('A', 'B'), ('A','C'), ('B','C')]

```
# 입력받은 아이템 준에서 minimum support 값을 넘는 요소만 반환
def filtering(item_db, min_sup):
    frequent = dict(filter(lambda val: val[1] >= min_sup, item_db.items()))
    return frequent

```

Filtering 함수는 minimum support 값을 넘는 요소만 반환해준다. 이를 구현하기 위해 lambda 함수를 이용해서 간단하게 {아이템: 빈도수}로 구성된 dictionary의 value를 기준으로 필터링을 실시했다.

```
# Association rule을 구하고 출력하는 함수
def association_rule(output_file, frequent_items, transactions):
    t_num = len(transactions)
    fo = open(output_file, "w")
    output = ""
    for i in range(len(frequent_items)): # 모든 행들을 순회하며
        for item in frequent_items[i]:      # 각 행에 대해서
            for size in range(1, i+1):      # 길이가 size인 조합 반환
                tmp_item = list(itertools.combinations(item, size))
                for tmp in tmp_item:        # 생성된 조합들로 이루어진 집합의 모든 부분집합들을 비교
                    result_item = list(item)
                    result_item = tuple([x for x in result_item if x not in tmp])
                    # X, Y 두 아이템이 같이 나오는 비율
                    support = (frequent_items[i][item]/t_num) * 100
                    # X가 나온 경우 X와 Y가 같이 나오는 비율
                    item_set = format(tmp)
                    associative_item_set = format(result_item)
                    confidence = (frequent_items[i][item]/frequent_items[len(set(tmp))-1][tmp]) * 100
                    output += f"{item_set}\t{associative_item_set}\t"+str("%.2f"%round(support,2))+"\t"+str("%.2f"%round(confidence,2))+"\n"
    fo.write(output)
    fo.close()
```

Association rule 함수에서는 입력 받은 frequent_items 리스트에 저장된 모든 item 조합 요소와 associative item set을 출력하고, 이에 대한 support와 confidence를 구하여 출력한다.

```
def format(item_set):
    return '{'+','.join(map(str, item_set)) + '}'
```

Association rule 함수 내에서 출력을 할 때, 출력 format을 맞추기 위해 format함수를 만들어줬다. 해당 함수를 통해 {[item],[item],[item]}과 같은 format을 지켜 출력하도록 했다.

3. Compiling method

실행환경:

OS: Windows 10

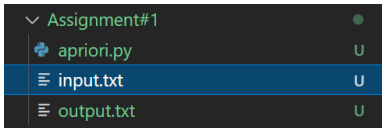
Python version: Python 3.8.3

실행방법:

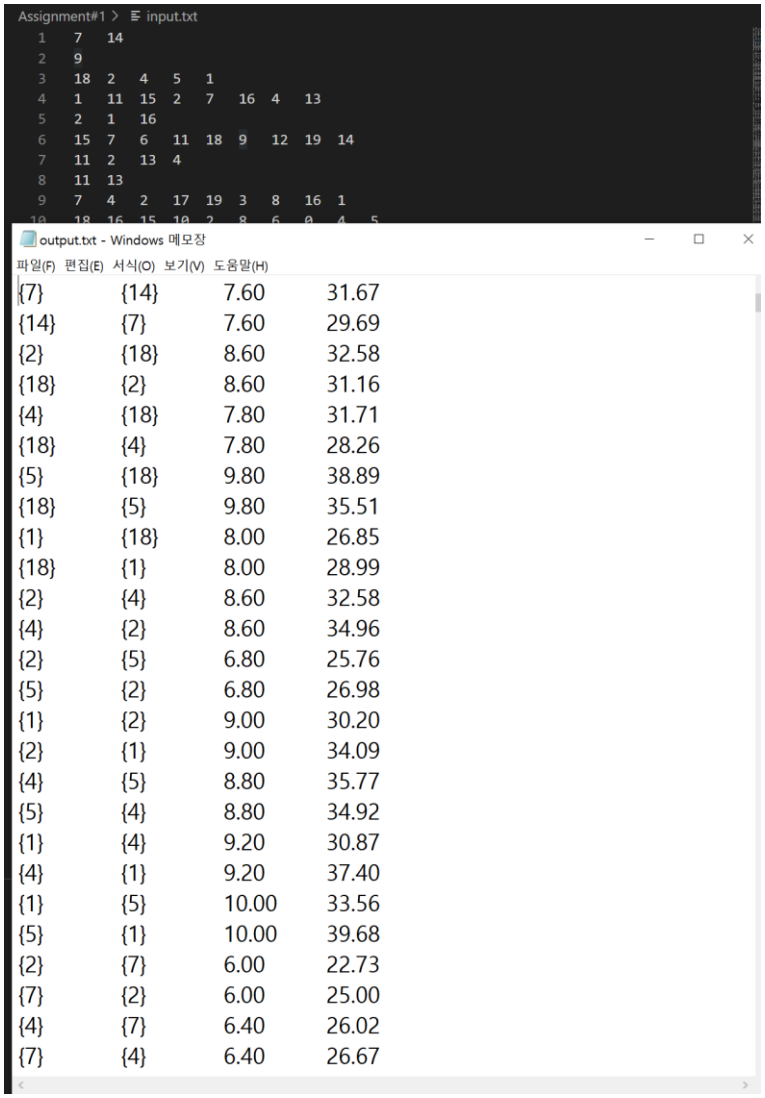
```
PS C:\Users\pyo99\CSE\DS\2022_ite4005_2019094511\Assignment#1>
PS C:\Users\pyo99\CSE\DS\2022_ite4005_2019094511\Assignment#1> python .\apriori.py 5 input.txt output.txt
PS C:\Users\pyo99\CSE\DS\2022_ite4005_2019094511\Assignment#1>
```

python apriori.py 5 input.txt output.txt

위의 실행 명령어를 입력하면 아래와 같이 output.txt 파일이 생성되는 것을 확인할 수 있다.



4. Other specification



위와 같이 input.txt에 대해 output.txt가 출력되는 것을 확인할 수 있다.