

Assignment1

Data Science
컴퓨터소프트웨어학부 2017029589 류지범

#0 Environment

- Python 3.7.11 에서 진행했다.
- 외부 모듈은 사용하지 않았다.
- 실행 환경은 macOS Monterey 12.3 이다.

#1 Run

- 소스 코드와 input 파일이 있는 폴더에서 `python apriori.py min_sup input.txt output.txt` 의 형태로 실행하면 된다.
- Min_sup의 값을 5로 할 경우 `python apriori.py 5 input.txt output.txt` 로 실행하면 된다.



```
Apple > ~/PycharmProjects/DataScience > DataScience < at 00:03:32  
> python apriori.py 5 input.txt output.txt
```

#2 Code & Function Description

- `read_input()` : input 파일로부터 입력을 받아온다. 각 transaction을 set의 형태로 list에 저장한다. Min_sup를 계산하기 편하게 하기 위해 count로 변환해서 저장해둔다.

```
def read_input():  
    global db_size, transactions, min_sup_cnt  
    with open(sys.argv[2], "r") as f:  
        while True:  
            line = f.readline()  
            if not line:  
                break  
            transactions.append(line.split())  
    db_size = len(transactions)  
    min_sup_cnt = db_size * (min_sup / 100)  
    transactions = list_to_set(transactions)
```

- `list_to_set()` : transaction의 list들을 set으로 바꾸기 위한 함수이다.

```
def list_to_set(l):  
    result = list()  
    for s in l:  
        result.append(set(s))  
    return result
```

- `get_cnt()` : 해당 item set이 전체 transaction에서 몇 번 등장했는지 개수를 반환해주는 함수이다. 집합 연산을 이용했다.

```
def get_cnt(item_set):
    cnt = 0
    for tra in transactions:
        if item_set == item_set & tra:
            cnt += 1
    return cnt
```

- `get_sup()` : support를 구하기 위해 해당 item set이 전체 transaction에서 몇 번 등장했는지 확률을 구해주는 함수이다. Min_sup 기준을 만족하지 못할 경우 0을 리턴하도록 했다.

```
def get_sup(item_set):
    cnt = 0
    for tra in transactions:
        if item_set == item_set & tra:
            cnt += 1
    if cnt >= min_sup_cnt:
        return cnt / db_size
    else:
        return 0
```

- `get_conf()` : item set -> associative item set의 confidence를 구하기 위한 함수이다. confidence를 구하기 위한 조건부 확률 식을 이용해서 계산하도록 했다.

```
def get_conf(item_set, associative_item_set):
    return get_cnt(item_set | associative_item_set) / get_cnt(item_set)
```

- `self_join()` : L_i 가 주어졌을 때 C_{i+1} 를 구하기 위한 self join 함수이다. L_i 의 각 item set의 전체 합집합을 구하면 C_{i+1} 를 구하기 위한 원소들의 집합을 구할 수 있고, `itertools`의 `combination()`을 이용해서 길이 $i + 1$ 을 가지는 모든 조합을 구하도록 했다.

```
def self_join(frequent_item_set, length):
    temp = set()
    for frequent_item in frequent_item_set:
        temp |= frequent_item
    return list_to_set(list(itertools.combinations(temp, length)))
```

- `pruning()` : `self_join()` 으로부터 C_i 를 구한 후 min_sup를 만족하지 못하는 것들은 가지치기하는 함수이다.

```
def pruning(candidate_item_set):
    result = list()

    for candidate in candidate_item_set:
        cnt = 0
        for tra in transactions:
            if candidate == candidate & tra:
                cnt += 1
        if cnt >= min_sup_cnt:
            result.append(candidate)

    return result
```

- `apriori()`: k의 크기를 1씩 늘려가면서 더 이상 candidate를 만들어내지 못할 때까지 L_i 와 C_{i+1} 를 만들어 내는 과정을 반복한다. pruning 까지 완료한 candidate는 `frequent_pattern` 에 저장해둔다.

```
def apriori():
    global frequent_pattern
    k = 1
    candidate = self_join(transactions, k)
    while True:
        l = pruning(candidate)
        frequent_pattern.extend(l)
        if len(l) == 0:
            break
        k += 1
        candidate = self_join(l, k)
```

- `get_associative()` : apriori 알고리즘을 돌려서 frequent pattern을 모두 찾으면, 각 frequent pattern마다 association rule을 구해주는 함수이다. 각각의 frequent pattern은 집합으로 이루어져있는데, 이 집합을 공집합이 아닌 서로 다른 집합(교집합이 x)으로 분할해서 min_sup를 만족하는 경우 파일에 item set -> associative item set sup conf 형태로 쓰도록 했다. 파일의 출력 양식을 맞추기 위해 소숫점 3자리에서 반올림했고, 공백 구분은 tab으로 했다.

```
def get_associative(item_set):
    global out
    if len(item_set) == 1:
        return

    a = list()
    b = list()

    for i in range(1, len(item_set)):
        a.extend(list_to_set(list((itertools.combinations(item_set, i)))))
        b.extend(list_to_set(list((itertools.combinations(item_set, i)))))
    for prev in a:
        for nxt in b:
            if len(prev & nxt) > 0 or (prev | nxt) != item_set:
                continue
```

```

        union = prev | nxt
        test = get_sup(union)
        if test == 0:
            continue
        else:
            line = print_set(prev) + "\t" + print_set(nxt) + "\t" + "
{:.2f}".format(round(decimal.Decimal(str(test * 100)), 2)) + "\t" + "
{:.2f}".format(round(decimal.Decimal(str(get_conf(prev, nxt) * 100)), 2)) + '\n'
            out.write(line)

```

- python의 `round()` 함수는 ROUND_HALF_EVEN을 따르기 때문에 원하는 결과를 얻기 위해 decimal module을 사용했다.

```

context = decimal.getcontext()
context.rounding = decimal.ROUND_HALF_UP

```

- `print_set()` : 집합을 출력 양식에 맞도록 변환해주는 함수이다.

```

def print_set(s):
    temp = sorted(map(int, list(s)))
    result = "{"
    for item in temp:
        result += (str(item) + ",")
    return result[:-1] + "}"

```

- `main()` : 프로그램에 필요한 전역 변수들을 선언했고, apriori 알고리즘을 돌린 후 구한 frequent pattern에 대해 association rule을 구해서 파일에 저장해준다.

```

if __name__ == '__main__':
    min_sup = int(sys.argv[1])
    out = open(sys.argv[3], 'w')
    transactions = list()
    db_size = 0
    min_sup_cnt = 0
    frequent_pattern = list()
    read_input()
    apriori()

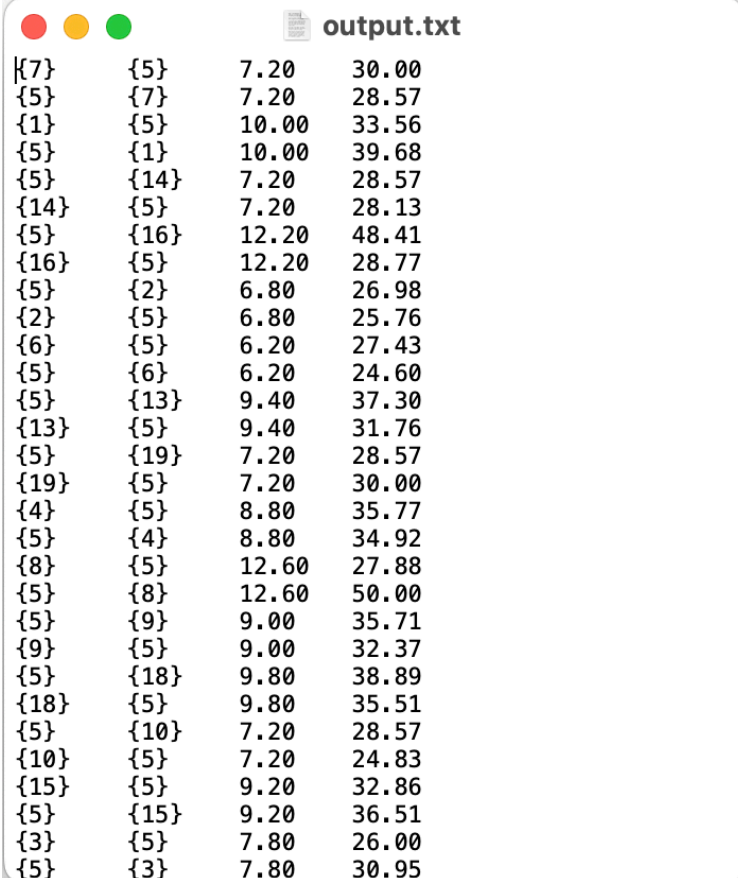
    for item_set in frequent_pattern:
        get_associative(item_set)

    out.close()

```

#3 Result

- output.txt는 다음과 같은 형태로 출력됐다.



{7}	{5}	7.20	30.00
{5}	{7}	7.20	28.57
{1}	{5}	10.00	33.56
{5}	{1}	10.00	39.68
{5}	{14}	7.20	28.57
{14}	{5}	7.20	28.13
{5}	{16}	12.20	48.41
{16}	{5}	12.20	28.77
{5}	{2}	6.80	26.98
{2}	{5}	6.80	25.76
{6}	{5}	6.20	27.43
{5}	{6}	6.20	24.60
{5}	{13}	9.40	37.30
{13}	{5}	9.40	31.76
{5}	{19}	7.20	28.57
{19}	{5}	7.20	30.00
{4}	{5}	8.80	35.77
{5}	{4}	8.80	34.92
{8}	{5}	12.60	27.88
{5}	{8}	12.60	50.00
{5}	{9}	9.00	35.71
{9}	{5}	9.00	32.37
{5}	{18}	9.80	38.89
{18}	{5}	9.80	35.51
{5}	{10}	7.20	28.57
{10}	{5}	7.20	24.83
{15}	{5}	9.20	32.86
{5}	{15}	9.20	36.51
{3}	{5}	7.80	26.00
{5}	{3}	7.80	30.95

- 총 1066 line이 출력됐고, min_sup를 만족하는 모든 association rule이 잘 출력되었다.