# Find association rules using the Apriori

### summary

apriori 알고리즘을 구현하고, 그것을 이용하여 minimum support를 만족하는 association rule과 각각에 대한 support, confidence를 구한다.

### Compilation method and environment

Windows: Python 3.7

```
$ python apriori.py 2 input.txt output.txt
```

argv[1] = minimum support

argv[2] = input file

argv[3] = output file

# apriori.py

## `cal_freq`

```
def cal_freq(data, set_):
    cnt = 0
    for i in range(len(data)):
        if set(set_).issubset(set(data[i])):
            cnt = cnt + 1
    return cnt
```

cal_freq 함수는 인자로 들어온 itemset에 대해 전체 transaction에서의 빈도수를 구해서 return한다. 각 transaction이 data[i]로 표현되며, 빈도수를 구하고자 하는 itemset이 해당 transaction의 subset이라면 `cnt`를 증가시킨다.

## `support`

```
def support(data, n):
    return n/len(data) * 100
```

support 함수는 인자로 들어온 정수 n을 support 형태로 변환하여 return한다. 즉, 값을 %단위로 변환한다.

## `nCr`

```
def nCr(n, r):
    f = math.factorial
    if n < r:
        return 0
    if n == r:
        return 1
    return f(n) // f(r) // f(n-r)
```

nCr 함수는 n개 중에서 r개를 뽑는 조합의 수를 return한다.

## `write_file`

```
def write_file(data, freq_set):
    for i in range(1, len(freq_set)):
        for j in range(len(freq_set[i])):
            for k in range(1, len(freq_set[i][j][0])):
                for ele in combinations(list(freq_set[i][j][0]), k):
                    output_file.write(str(set(ele)) + '\t'
                                      + str(set(freq_set[i][j][0] - set(ele))) + '\t'
                                      + format(freq_set[i][j][1], '.2f') + '\t'
                                      + format(freq_set[i][j][1]/support(data,
cal_freq(data, set(ele))) * 100, '.2f') + '\n')
```

write_file 함수는 association rules와 각각의 rule에 대한 support, confidence 값을 output file에 출력한다. `freq_set`에 저장된 frequent items에 대해, 2-items set부터 시작하여 가장 큰 items set까지 combinations 함수를 사용하여 각 item set에 대한 가능한 association rule을 모두 생성하고, 각 rule에 대해 support와 confidence를 구하여 출력한다.

## `candi_set_gen`

```
def candi_set_gen(data, items, k):
    global candi_set
    if k == 0:
        candi_set.append([])
        for i in range(len(items)):
            cnt = cal_freq(data, [items[i]])
            candi_set[k].insert(len(candi_set[k]),(set([items[i]]), support(data, cnt)))

    else:
        candi_set.append([])
        comb_all = combinations(items, k + 1)
        for comb in comb_all:
            chk = 0
            for pset in freq_set[k-1]:
                if set(pset[0]).issubset(set(comb)):
                    chk = chk + 1
            if chk >= (nCr(k + 1, k)):
                cnt = cal_freq(data, comb)
```

```
                candi_set[k].insert(len(candi_set[k]), (set(comb), support(data, cnt)))
```

candi_set_gen 함수는 k번째 candidate set (pattern의 길이가 k+1)을 tuple list형태로 generate하는 함수다.

k가 0일 경우에는, transaction에 있는 모든 item set이 set에 포함되므로 따로 처리해준다. `items`는 transaction에 존재하는 모든 unique한 item들의 list인데, `items`에 포함된 모든 원소들에 대한 frequency를 계산하여 0번째 candi_set에 insert한다.

k가 0이 아닐 경우에는, `item`으로 만들 수 있는 원소가 k+1인 조합들을 생성하여 `comb_all`에 저장하고, 각 조합들에 대해 k-1번째 frequent set을 확인하며, 조합의 원소보다 1개 적은 길이의 subset이 모두 frequent set에 존재하는지 체크한다 (Pruning). 조건을 만족한 조합들에 대해서는 support를 계산하여 k번째 candidate set에 insert한다.

## `freq_set_gen`

```
def freq_set_gen(candi_set, min_supp, k):
    global freq_set
    freq_set.append([])
    freq_set[k] = [x for x in candi_set[k] if x[1] >= min_supp]
```

freq_set_gen 함수는 k번째 frequent set을 tuple list형태로 generate하는 함수다.

k번째 candidate set에 대해, support 값이 minimum support 이상이면 frequent set에 포함한다.

## `apriori`

```
def apriori(input_file, output_file, min_supp):
    global data
    global candi_set
    global items

    data = input_file.readlines()
    for i in range(len(data)):
        data[i] = data[i].strip().split('\t')
        data[i] = list(map(int, data[i]))

    t = pd.DataFrame(data).fillna(-1).astype('int')

    for i in range(len(t.columns)):
        items = np.concatenate((items, t[i].unique()), axis = 0)
    items = np.unique(items)
    items = np.delete(items, 0).astype('int')

    k = 0
    while True:
        candi_set_gen(data, items, k)
        freq_set_gen(candi_set, min_supp, k)
        if len(freq_set[k]) == 0:
            del freq_set[k]
            break
```

```
        k = k + 1
```

apriori 함수는 input file에서 transaction을 받아 list형태의 `data`, dataframe 형태의 `t`, numpy array 형태의 `items`에 저장한다. 그리고 더이상 `freq_set`에 새로운 frequent pattern이 추가되기 전까지 candidate set과 frequent set을 업데이트하고, 마지막으로 생성된 빈 freq_set의 list를 삭제하여 종료한다.

## result (ex)

```
1   {0} {1} 6.60    24.63
2   {1} {0} 6.60    22.15
3   {0} {2} 8.60    32.09
4   {2} {0} 8.60    32.58
5   {0} {3} 5.60    20.90
6   {3} {0} 5.60    18.67
7   {0} {4} 5.60    20.90
8   {4} {0} 5.60    22.76
9   {0} {5} 7.40    27.61
10  {5} {0} 7.40    29.37
11  {0} {6} 6.80    25.37
12  {6} {0} 6.80    30.09
13  {0} {7} 7.40    27.61
14  {7} {0} 7.40    30.83
15  {0} {8} 11.80   44.03
16  {8} {0} 11.80   26.11
17  {0} {9} 7.60    28.36
18  {9} {0} 7.60    27.34
19  {0} {10}    10.00   37.31
20  {10}    {0} 10.00   34.48
```