Assignment#1

2017029970 우원진

1. Algorithm Summary

거의 모든 함수에서 원소의 길이가 1일때와 나머지로 나누어서 진행 했습니다.
대략적인 알고리즘은 이론에서 배운것과 마찬가지로, 원소의 길이를 늘려가면서 예를들어 길이가 3이면 길이 2짜리 원소들에서 중복되지않는 길이 1짜리 원소들을 모두 뽑아서 그 길이 1짜리 원소들로 만들수있는 모든 조합을 만든후, 그 조합들에서 각각 조합마다(예시에서는 길이 3짜리 원소 겠죠?) 그 조합으로 만들수있는 길이-1 짜리 모든조합(예시에서는 길이 2짜리 조합)을 만든 후 만약 그 조합중 하나라도 이전에 구했던 조합에 들어있지 않다면 삭제했습니다. 그리고 이 과정을 통해 나온 원소들을 가지고 또 DB를 돌면서 minimum support를 만족하는지 확인하고 만족하지 못한다면 삭제한 후 그 결과물을 저장해서 반복하는 식으로 진행했습니다.

2. Description of Code

```
"""
각 elements값들이 DB에 들어있는 갯수를 {원소: 갯수} 형태로 가지고 있는 Dictionary를 인자로 받아 minimum_support
를 넘지 못하는 elements들은 거르고, 만족하는 원소들의 list를 반환하는 함수이다
"""
def remove_item_about_support(item_dict):
    result = []
    for item in item_dict:
        if item_dict[item] >= minimum_support:
            result.append(item)
    if len(result) == 0:
        is_stop = True
    else:
        return result
```

“Remove_item_about_support” 함수는 {원소:갯수}식의 dictionary를 argument로 받아 원소들중 주어진 minimum_support를 만족하지 못하는 원소들을 제거하는 함수이다.

```
"""
예를들어 [1, 2]가 [1, 2, 3]에 들어있는지와 같은 확인은 순서에 상관없어야하고, 편리성을 위해 set으로 바꾸는게 필요해서
list를 set으로 바꾸는 함수를 만들었다
"""
def list_to_set(item_list):
    result = []
    for item in item_list:
        result.append(set(item))
    return result
```

List_to_set함수는 계산의 편리성을 위해 Set을 사용하기로 했는데 만약 아이템들이 [[1,2,3], [3,4,5]] 식으로 있을때 이것을 [{1,2,3}, {3,4,5}] 식으로 바꿔주는 함수이다.

```
"""
조합에 해당하는 함수로, 현재 가진 원소들에서 나올수 있는 모든 조합을 만들고 그 조합들 중에서, 각각 마다 만약
subset이 해당 갯수의 frequent set에 하나라도 포함되지않으면 지우는 함수이다.
"""
def join(length):
    if length == 2:
        result = list(itertools.combinations(
            frequent_items[current_item_len-2], current_item_len))
        result_list = list(map(list, result))
        result_set = list_to_set(result_list)
        if(len(result_set) == 0):
            is_stop = True
        return result_set
    else:  # 원소가 3 개이상
        elements = []
        k_items = frequent_items[current_item_len-2]
        for k_item in k_items:
            k_item_list = list(k_item)
            for item in k_item_list:
                if item not in elements:
                    elements.append(item)
        combinations_result = list(itertools.combinations(elements, length))
        remove_result = copy.deepcopy(combinations_result)
        for l in combinations_result:  # 원소 3개 짜리 items라고 생각
            l_combinations = list(itertools.combinations(l, length-1))
            for i in l_combinations:  # 원소 2개 짜리 items라고 생각
                i_set = set(i)
                if i_set not in frequent_items[length-2]:
                    remove_result.remove(l)
                    break
        remove_result_set = list_to_set(remove_result)
        if len(remove_result_set) == 0:
            is_stop = True
        return remove_result_set
```

Join 함수는 현재의 길이보다 1작은 원소들을 가지는 list에서 길이 1짜리 원소들을 다 뽑아내어 이 원소들로 만들수있는 현재길이의 조합을 모두 만들고, 그 조합 마다의 길이-1 짜리 조합을 모두 만들어 만약 이 조합들에서 하나라도 frequent list에 들어있지 않으면 삭제하는 식으로 작동하는 함수이다.

```
"""
원소가 1개부터 늘어나는 형식이므로 frequent한 원소들은 추가하고, 또 원소의 갯수도 증가시켜준다. 메인함수라고 볼수있다.
frequent_items list에는 0번 index에는 길이 1짜리 frequent item set list, 1번 index에는 길이 2짜리 frequent item set list, 이런식으로 저장된다.
"""
def execute_apriori():
    global current_item_len
    global DB_transactions
    while True:
        dict_for_item_and_count = {}
        if current_item_len == 1:
            for transac in DB_transactions:
                for item in transac:
                    if item not in dict_for_item_and_count:
                        dict_for_item_and_count[item] = 1
                    else:
                        dict_for_item_and_count[item] += 1
            result = remove_item_about_support(dict_for_item_and_count)  # list
            frequent_items.append(result)
            current_item_len += 1
        else:  # 원소 갯수가 2개 이상
            items = join(current_item_len)  # sets
            if is_stop:
                break
            dict_for_item_and_count = {}
            for item in items:
                item_cnt = 0
                for transac in DB_transactions:
                    transac_combinations = list(
                        itertools.combinations(transac, current_item_len))
                    transac_sets = list_to_set(transac_combinations)
                    if item in transac_sets:
                        item_cnt += 1
                dict_for_item_and_count[tuple(item)] = item_cnt
            # print(dict_for_item_and_count)
            result = remove_item_about_support(dict_for_item_and_count)
            if result == None:
                break
            result_set = list_to_set(result)
            frequent_items.append(result_set)
            current_item_len += 1
```

Execute_apriori 함수는 메인함수에 해당하며, 위에 설명한 함수들을 모두 실행시키는 함수이다.
이 함수는 조건에 만족하는 원소 set의 list가 나오면 이를 frequent_items list변수에 저장하고, 현재 찾고있는 원소의 길이를 나타내는 current_item_len 변수를 1만큼 증가시킨다.

```
"""
association rule을 만드는 함수로, 만약 {1, 2, 3}이라는 원소가 있다면, 이와 관련된 길이 1, 2짜리 모든 subset을 만들었고,
그것과의 차집합을 구해 차집합 -> subset 식의 모든 association rule을 만들어 support와 confidence 를 구했다.
"""
def association_rule():
    max_frequent_len = len(frequent_items)
    DB_set = list_to_set(DB_transactions)
    DB_len = len(DB_transactions)
    file_output = open(output_file, "w")
    for i in range(1, max_frequent_len):  # 빈번하게 발생하는 모든 elements
        for element in frequent_items[i]:
            element_len = i+1
            for_iter = 1
            support_cnt = 0
            for transac in DB_set:
                if transac >= element:
                    support_cnt += 1

            while element_len > for_iter:
                element_combinations = list(
                    itertools.combinations(element, element_len-for_iter))
                element_comb_set = list_to_set(element_combinations)
                for comb in element_comb_set:
                    left = element - comb  # left side, comb is right side
                    left_cnt = 0
                    right_cnt = 0
                    for transac in DB_set:
                        if transac >= left:
                            left_cnt += 1
                            if transac >= comb:
                                right_cnt += 1
                    support = format(support_cnt/DB_len * 100, ".2f")
                    confidence = format(right_cnt/left_cnt*100, ".2f")
                    output = str(left) + "\t" + str(comb) + "\t" + \
                        str(support) + "\t" + str(confidence) + "\n"
                    file_output.write(output)
                for_iter += 1


execute_apriori()
association_rule()
```

Association_rule 함수는 frequent_items 들에 대해 각각 마다 만들수있는 모든 조합을 만들고 그거을 가지고 조합마다 원래꺼와의 차집합을 구하고 그 후 차집합 -> 조합 이라는 Association rule을 만들어 support와 confidence를 구하고, 이를 output.txt에 저장하는 함수이다.

3. 실행방법

```
uwonjin-ui-MacBookPro:Assignment#1 woowonjin$ python3 apriori.py 5 input.txt output.txt
```

이런식으로 실행하면된다. 단 저의 실행환경은 python 3.8.2 버전입니다.