Assignment-1 (Apriori)

2016025105
컴퓨터공학과
강재훈

(i) Summary of your algorithm

Apriori 알고리즘은 frequent pattern 및 association rule을 찾기 위한 알고리즘입니다. database 내의 trx들을 읽으며 candidate들을 뽑아내고 이러한 candidate들이 실제로 frequent한지 판별하는 작업을 거칩니다.

구체적으로 self_-joining과 pruning을 통해 candidate를 선정합니다.
self-joining이란 길이가 K인 pattern의 candidate는 길이가 K-1인 pattern들 중 2개의 pattern을 골라 길이가 K인 pattern을 만드는 방식입니다. (K-2개의 원소가 같고 1개의 원소가 다를 때 K-2 +1 +1 => K 길이의 패턴)
pruning이란 self-joining을 통해 만들어진 candidate들 중 실제 frequent한 pattern이 될 수 있는 것은 어떤 것이 있는지를 선별합니다. 이는 "어떠한 frequent pattern이 있다면 해당 pattern의 sub-pattern은 모두 frequent하다." 라는 참인 명제의 대우인 "어떤 itemset이 frequent하지 않다면 이것의 super-set들은 frequent 할 수 없다." 를 이용합니다.
이후 선정된 candidate를 가지고 해당 pattern이 database의 trx들에서 실제로 몇번 나오는지 카운트하여 특정 길이에서의 frequent pattern이 될 수 있는 후보자가 없을 때까지 frequent pattern인지 아닌지 결정 짓습니다.

Apriori 알고리즘은 trx database를 여러번 scan해야 하는 점과 여전히 cadidate의 수가 많다는 단점이 있고 이를 해결하기 위해 다양한 기법들이 존재합니다.

우리의 과제에서는 trx database의 크기가 main-memory에 모두 올라갈 수 있는 크기로 제한되어 있기 때문에 프로그램이 처음 실행 될 때 trx database의 내용 전부를 memory에 올리는 방식을 채택하였고 pattern들의 포함 유무를 쉽게 측정하기 위해 python의 set() 자료 구조를 사용하였습니다. 또한 pattern들의 count를 쉽게 저장하고 접근하기 위해 python의 dict() 자료구조를 사용하였습니다. 코드의 자세한 구현은 (ii)에서 설명하도록 하겠습니다.

(ii) Summary of your algorithm

- Apriori

본 과제의 목표인 Apriori 알고리즘을 구현한 함수입니다.

해당 함수를 몇개의 부분으로 나눠 설명하겠습니다.

```
def apriori(trxs):
    global min_support
    trxs_len = len(trxs)

    idxs = set([])
    for trx in trxs :
        idxs = idxs.union(trx)

    # 일반 set 은 unhashable type 이므로 frozenset 사용
    candidates = { frozenset({i}) for i in idxs }
    # candidates = { set({i}) for i in idxs }

    item_set = dict()
    K = 1
```

trxs_len == 전체 trx의 갯수

(이 trxs_len은 이후 pattern의 support를 구하기 위해 사용합니다.)

idxs == 전체 trx들 안에 들어있는 index

(set() 자료구조를 사용하여 각각의 index가 1개씩만 존재할 수 있도록 하였습니다.)

candidates == frequent pattern이 될 수 있는 후보자들

(길이가 1인 pattern은 모두 candidate가 될 수 있으므로 모두 저장)

(python의 set은 unhashable type이기에 frozenset()을 이용해 저장)

item_set == 최종 frequent pattern들을 저장

ex) { 1 : {길이가 1인 FP : FP의 support, : , ...}, 2 : {길이가 2인 FP : support ..... } ..... }

K == 현재 조사하고 있는 pattern의 길이

```
while candidates :
    count = dict()

    for trx in trxs :
        for candidate in candidates:
            if candidate.issubset(trx):
                try :
                    count[candidate] += 1
                except KeyError :
                    count[candidate] = 1
```

count == { candidate1 : trx db에서 몇번 나왔는지, candidate2 : 몇번 나왔는지 … }

main-memory에 올라와 있는 trx database (trxs)들을 1개씩 보며 각각의 candidate가 몇번 나오는지 기록합니다.

```
# pruning
after_pruning = { key : (float(value) / trxs_len)
for (key, value) in count.items() if (float(value) / trxs_len) >= min_support }
item_set[K] = after_pruning
```

after_pruning == 실제 길이가 K인 frequent한 pattern을 저장하고 있음

상단의 for trx in trxs : 반복문이 끝나고 나서 candidate들 중 실제 frequent한 pattern들만 after_pruning에 담습니다. 이후 item_set[K] 에 해당 pattern들을 저장해줍니다.

```
#self_joining
K += 1
candidates = { i.union(j) for i in after_pruning for j in after_pruning if len(i.union(j)) == K }
```

candidates == 길이가 K-1인 FP들로 self-joining해서 만들어진 길이가 K인 candidate들

이후 K를 1 증가시켜 주고 해당 길이의 candidate를 구해줌으로써 다음 반복을 준비합니다.

이후 FP들이 모두 저장되어 있는 item_set을 return 하며 apriori 알고리즘을 종료됩니다.

- print_output

apriori 알고리즘을 통해 구한 FP들을 이용해 association rule을 구하고 association의 X 부분, Y 부분, support, confidence를 출력합니다.

(min_confidence가 제시되지 않았으므로 0이라 생각하고 모든 경우 출력)

```
def print_output(trxs, fps):
    for patt_len, patt_len_fps in fps.items():
        if patt_len == 1 :
            continue
```

trxs == main-memory에 올라가 있는 trx database

fps == apriori 알고리즘을 통해 구한 FP들

patt_len == 현재 조사하고 있는 fp들의 길이

patt_len_fps == 현재 조사하고 있는 길이의 fp들

patt_len 이 1인 경우 association rule을 만들수 없으므로 다음 반복으로 넘어갑니다.

```
for fp in patt_len_fps :
    com_len_cases = [ combinations(fp, length) for length in range(1, len(fp)+1, 1) ]
    all_cases = []
    for  cases in com_len_cases :
        for case in cases :
            all_cases.append(frozenset(case))
```

com_len_cases == FP 1개로 만들 수 있는 모든 조합

all_cases == 조합을 frozenset으로 바꾸어 저장

(이후 계속해서 집합 연산을 사용하기 때문에)

```
for case in all_cases:
    remainder = fp.difference(case)

    if remainder :
        confidence = fps[len(fp)][fp] / fps[len(case)][case]
        prt_case, prt_remainder = str(set(map(int,case))).replace(" ", ""), str(set(map(int,rema
        prt_supp, prt_confi =  str('%.2f' % round(fps[len(fp)][fp] * 100, 2)), str('%.2f' % roun
        string = prt_case + '\t' + prt_remainder + '\t' + prt_supp + '\t' + prt_confi + '\n'

        with open(output_file, 'a') as f :
            f.write(string)
```

remainder == FP에서 case (FP에서 뽑아낸 조합)을 빼고 난 나머지

confidence == 해당 association rule의 confidence

ptr_case, ptr_remainder, ptr_sup, ptr_confi == 각각의 요소를 출력하기 위해 포멧을 맞춰줌

- main

```
with open(input_file, 'r') as f :
    trxs = [ trx.split('\t') for trx in f.read().splitlines()  ]

    fps = apriori(trxs)
    print_output(trxs, fps)
```

main 함수입니다. 프로그램 실행시 argv[1]로 주었던 input 파일을 열어 database내의 모든 trx를 불러와 trxs에 저장합니다. 이후 apriori 알고리즘을 실행하여 FP들을 구하고 print_output 함수를 실행해 과제의 명세에 맞게 출력합니다.

(iii) Instructions for compiling your source codes at TA's computer

python version == Python 3.7.6

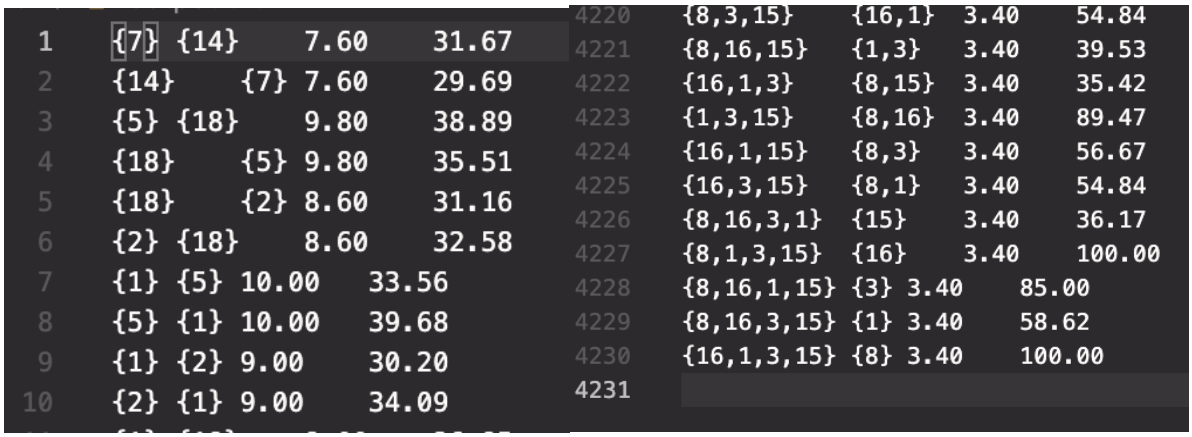
실행법 == python apriori.py min_support input_file_name output_file_name

ex)

```
] python apriori.py 3 input.txt output.txt
```

(iii) Any other specification of your implementation and testing

주어진 input_file에 대한 min_support == 3 결과물 (output3.txt)



```
1    {7} {14}    7.60    31.67
2    {14}    {7} 7.60    29.69
3    {5} {18}    9.80    38.89
4    {18}    {5} 9.80    35.51
5    {18}    {2} 8.60    31.16
6    {2} {18}    8.60    32.58
7    {1} {5} 10.00   33.56
8    {5} {1} 10.00   39.68
9    {1} {2} 9.00    30.20
10   {2} {1} 9.00    34.09
```

```
4220  {8,3,15}    {16,1}  3.40    54.84
4221  {8,16,15}   {1,3}   3.40    39.53
4222  {16,1,3}    {8,15}  3.40    35.42
4223  {1,3,15}    {8,16}  3.40    89.47
4224  {16,1,15}   {8,3}   3.40    56.67
4225  {16,3,15}   {8,1}   3.40    54.84
4226  {8,16,3,1}  {15}    3.40    36.17
4227  {8,1,3,15}  {16}    3.40    100.00
4228  {8,16,1,15} {3} 3.40    85.00
4229  {8,16,3,15} {1} 3.40    58.62
4230  {16,1,3,15} {8} 3.40    100.00
4231
```

주어진 input_file에 대한 min_support == 4 결과물 (output4.txt)

```
1    {14}    {7} 7.60    29.69
2    {7} {14}    7.60    31.67
3    {2} {4} 8.60    32.58
4    {4} {2} 8.60    34.96
5    {18}    {5} 9.80    35.51
6    {5} {18}    9.80    38.89
7    {1} {4} 9.20    30.87
8    {4} {1} 9.20    37.40
9    {18}    {4} 7.80    28.26
10   {4} {18}    7.80    31.71
```

```
1795   {8,1}   {16,15} 4.00    25.97
1796   {8,16}  {1,15}  4.00    13.25
1797   {8,15}  {16,1}  4.00    32.26
1798   {16,1}  {8,15}  4.00    24.69
1799   {1,15}  {8,16}  4.00    37.04
1800   {16,15} {8,1}   4.00    28.99
1801   {8,1,16}    {15}    4.00    34.48
1802   {8,1,15}    {16}    4.00    74.07
1803   {8,16,15}   {1} 4.00    46.51
1804   {16,1,15}   {8} 4.00    66.67
```

주어진 input_file에 대한 min_support == 5 결과물 (output5.txt)

```
1    {7} {14}    7.60    31.67
2    {14}    {7} 7.60    29.69
3    {1} {2} 9.00    30.20
4    {2} {1} 9.00    34.09
5    {5} {2} 6.80    26.98
6    {2} {5} 6.80    25.76
7    {18}    {1} 8.00    28.99
8    {1} {18}    8.00    26.85
9    {4} {1} 9.20    37.40
10   {1} {4} 9.20    30.87
```

```
1056   {14}    {16,8,3}    5.40    21.09
1057   {16,3}  {8,14}  5.40    21.43
1058   {8,3}   {16,14} 5.40    20.93
1059   {3,14}  {16,8}  5.40    79.41
1060   {16,8}  {3,14}  5.40    17.88
1061   {16,14} {8,3}   5.40    49.09
1062   {8,14}  {16,3}  5.40    48.21
1063   {16,8,3}    {14}    5.40    22.50
1064   {16,3,14}   {8} 5.40    100.00
1065   {8,3,14}    {16}    5.40    90.00
1066   {16,8,14}   {3} 5.40    79.41
```