Programming Assignment #1 Report

2016024766

김서현

1. 구현한 Apriori 알고리즘에 대한 설명

Apriori 알고리즘은 주어진 transaction을 분석하여 frequent할 가능성이 있는 candidate를 만들고, 그 candidate가 정말 frequent한지 테스트하는 방식으로 frequent patterns를 만들고, 그들 사이의 association rules를 찾는 것을 말합니다.

1) Input file을 scan하면서 length가 1인 candidate를 만들고, candidate마다 support를 계산합니다.

2) Length가 1인 candidate 중 support가 minimum support 이상인 candidate만 걸러서 length가 1인 frequent pattern으로 저장하고, support를 freq_dict 안에 저장합니다.

3) Length를 하나씩 증가시키면서 새로운 frequent pattern이 생성되지 않을 때까지 아래의 과정을 반복합니다.

   A. (Length – 1) 길이의 frequent pattern 중에서 두개를 뽑아 합집합을 구했을 때 길이가 length가 된다면 pruning으로 검증 후에 candidate에 추가합니다.

   B. Input file을 scan하면서 candidate들의 support를 계산합니다. 이 때 계산한 모든 support들을 freq_dict에 저장합니다.

   C. Candidate 중 support가 minimum support 이상인 것들만 frequent pattern으로 저장합니다. 이 때 minimum support보다 작은 candidate들의 support는 freq_dict에서 제거합니다.

4) Frequent pattern을 가지고 association rules를 계산합니다. Association rule에서 support와 confidence를 계산할 때 freq_dict를 활용합니다.

5) 결과로 나온 association rules를 output file에 출력합니다.

2. 코드에 대한 설명

1) Main

```python
if len(sys.argv) < 4:
    print(
        "USAGE: python apriori.py <minimum support> <input filename> <output filename>")
    sys.exit()
min_support = int(sys.argv[1])
input_file = sys.argv[2]
output_file = sys.argv[3]
# check minimum support
if min_support <= 0:
    print("minimum support must be a possitive.")
    sys.exit()
f_in = open(f'./{input_file}', 'r')
```

Input을 받는 부분입니다. Argument가 누락되거나 Minimum support가 0 이하이면 프로그램을 종료하도록 하였습니다.

```python
# transaction의 총 개수
total_cnt = 0
while True:
    line = f_in.readline()
    if not line:
        break
    total_cnt += 1

    # 문자열을 int로 변환
    line = [int(i) for i in line.split()]
    for i in line:
        if i in can_1:
            can_1[i] += 1
        else:
            can_1[i] = 1
min_sup = total_cnt*min_support/100
# 길이가 1인 freq patterns
L1 = []
for i in can_1:
    if can_1[i] >= min_sup:
        L1.append(set([i]))
        freq_dict[frozenset([i])] = can_1[i]
```

길이가 1인 frequent pattern을 찾는 부분입니다. Freq_dict라는 Dictionary의 key로 frozenset을 사용하였습니다. Input file을 scan하면서 길이가 1인 candidate들과 그 support를 계산한 뒤, support가 minimum support 이상인 경우만 frequent pattern으로 저장하였습니다.

```
leng = 2
L_now = L1
while True:
    Ck = make_candidate(L_now, leng)
    Ck_cnt = scan_db(f_in, Ck, freq_dict)
    L_next = find_freq(Ck_cnt, min_sup, freq, freq_dict)

    if not L_next:
        break
    L_now = L_next
    leng += 1
f_in.close()
```

길이가 2 이상인 frequent pattern을 구하는 부분입니다. 새로 계산된 frequent pattern이 비었으면 whlie문을 종료하도록 하였습니다. 모든 frequent pattern을 찾은 뒤에 input file을 닫았습니다.

```
# association rule
result = association(freq, freq_dict, total_cnt)
# print(result)
f_out = open(f'./{output_file}', 'w')
f_out.write(result)
f_out.close()
```

위에서 찾은 Frequent pattern을 가지고 association rule을 계산하는 부분입니다. Association rule을 output file에 출력하고 프로그램이 종료됩니다.

2) make_candidate

```
# leng 길이의 candidates 생성
def make_candidate(Lk, leng):
    ret = set([])
    for i in range(len(Lk)):
        for j in range(i+1, len(Lk)):
            can = Lk[i] | Lk[j]
            if len(can) == leng:
                # pruning
                if leng > 2:
                    can_com = list(combinations(can, leng-1))
                    check_prun = set([])
                    for item in can_com:
                        check_prun.add(frozenset(item))
                    check_prun = frozenset(check_prun)
                    if check_prun.issubset(Lk):
                        ret.add(frozenset(can))
                else:
                    ret.add(frozenset(can))
    return ret
```

길이가 leng-1인 frequent pattern과, leng 값을 인자로 받아 leng 길이의 candidate를 생성해 리턴하는 함수입니다. Frequent pattern에서 두개씩 조합해서 합집합의 길이가 leng이 되는 경우만 candidate로 추가했습니다. Leng이 3 이상인 경우부터는 pruning 과정을 통해 모든 부분집합이 leng-1 길이의 frequent patterns에 속한다는 검증 후에 candidate으로 추가하도록 하였습니다. Set()의 합집합 기능과 중복이 허용되지 않는다는 점을 이용해 구현했습니다.

3) scan_db

```
# input file을 scan하면서 candidate들의 개수를 count
def scan_db(f, Ck, freq_dict):
    f.seek(0)
    cnt = {}
    while True:
        line = f.readline()
        if not line:
            break
        line = [int(i) for i in line.split()]
        # transaction마다 candidate들을 모두 확인하고 개수 count함
        for can in Ck:
            if can.issubset(line):
                if frozenset(can) in cnt:
                    freq_dict[frozenset(can)] += 1
                    cnt[frozenset(can)] += 1
                else:
                    freq_dict[frozenset(can)] = 1
                    cnt[frozenset(can)] = 1
    return cnt
```

Input file을 스캔하면서 candidate들의 support를 계산하는 함수입니다. Input file의 한 transaction을 읽을 때마다 transaction 안에 포함되어있는 모든 candidate의 support를 올려주는 방식을 사용하여 input file의 scan을 한 번만 하도록 구현했습니다. freq_dict 안에는 우선 모든 candidate의 support를 다 저장하도록 하였습니다.

4) find_freq

```python
# candidate들 중 frequent pattern인 것만 찾아냄
def find_freq(Ck, min_sup, freq, freq_dict):
    ret = []
    for item in Ck:
        if Ck[item] >= min_sup:
            ret.append(item)
            freq.add(item)
        else:
            del(freq_dict[item])
    return ret
```

Candidate의 support가 minimum support 이상인 경우에만 frequent pattern이라고 저장했습니다. Frequent pattern이 아닌 경우에는 freq_dict에 넣었던 candidate와 support를 제거해서 freq_dict 안에는 frequent pattern의 support들만 남게 만들었습니다.

5) Association

```python
# association rule에 따라 frequent pattern들 사이의 support, confidence 찾음
def association(freq, freq_dict, total):
    freq = list(freq)
    res = ""
    for items in freq:
        items = list(items)
        sup_cnt = freq_dict[frozenset(items)]
        sup = format(round(sup_cnt/total*100, 2), ".2f")
        size = len(items)
        for i in range(1, size):
            comb = list(combinations(items, i))
            for X in comb:
                X = frozenset(X)
                Y = frozenset(items) - X
                conf_X = format(round(sup_cnt/freq_dict[X]*100, 2), ".2f")
                res += f'{set(X)}\t{set(Y)}\t{sup}\t{conf_X}\n'
    return res
```

Frequent patterns를 인자로 받아 association rules를 생성하는 함수입니다. 각 frequent pattern마다 combination을 이용해 가능한 모든 조합을 구하고, 두 조합의 support와 confidence를 계산했습니다. 계산을 할 때 freq_dict에 저장했던 support값들을 이용하기 때문에 input file을 다시 스캔하지 않습니다. 생성한 association rules는 output file에 출력할 형태로 res에 저장하고 res를 return합니다.

3. 컴파일 방법

Python 3.9.1 버전으로 작성한 프로그램입니다.

테스트는 windows10 내의 command prompt를 사용하여 진행했습니다.

1) Apriori.py와 input.txt. 파일을 같은 디렉토리 안에 넣습니다.

2) "python apriori.py <minimum support> <input file> <output file>"의 형식으로 실행시킬 수 있습니다.

   Ex) `python apriori.py 5 input.txt output.txt`

4. 특이사항

- Argument를 누락하거나 입력 받은 minimum support가 0 이하인 경우에는 메시지를 프린트한 뒤 프로그램을 종료하도록 하였습니다.
- Minimum support를 3으로 실행한 결과와 5로 실행한 결과를 git에 함께 제출하였습니다.