Assignment1 Report

(Apriori)

2016025478 서경욱

1. 개발 환경

- **파이썬 3.8.6**

```
C:₩Users₩user>python --version
Python 3.8.6
```

2. 실행 방법

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment1 (master)
λ python apriori.py 5 input.txt output.txt
```

- **python apriori.py (minimum support) (input 파일 경로) (output 파일 경로)**
- 0 < minimum support <= 100

3. Summary of Algorithm

본 과제에서는 Apriori 알고리즘을 통해 frequent pattern의 특징인 downward closure를 이용하여 frequent pattern들을 찾고 association rules를 찾아 각각의 case의 support, confidence를 정해진 format에 맞게 output file에 출력하는 프로그램을 작성하였다.

이 프로그램은 minimum support(%) 값, input 파일 경로, output 파일 경로를 실행 parameter로 받으며 main에서 **parameter가 알맞은 개수**로 들어왔는지 확인했다. 다음으로 Input 텍스트 파일을 한 줄씩 읽으며 각 transaction의 정보와 item의 인덱스를 저장한 후 이를 이용해 **frequent한 1-itemset**을 만들어 냈다. 그리고 Apriori 알고리즘에 따라 previous itemset을 이용하여 **self-join하여** 길이가 k+1인 candidate를 만들어내고 **pruning**을 통해 previous itemset에 subset이 없는 candidate를 제거했다. 남은 candidate들은 support를 **test**하여 minimum support보다 작은 support를 가지는 candidate는 제거했다. 이 과정을 길이 k값을 증가시켜가며 반복문을 통해 반복하고 output text에 써주었다. candidate가 생성되지 않은 경

우, 더 이상 frequent한 pattern이 없는 경우 종료하도록 했다.

4. Detailed description of each function

```
110  if __name__ == '__main__':
111      start = time.time()
112      # getting parameters
113      min_support = float(sys.argv[1]) * (0.01)
114      input_path = sys.argv[2]
115      output_path = sys.argv[3]
116
117      # check correct parameter
118      if len(sys.argv) != 4:
119          print("Insufficient arguments")
120          sys.exit()
121
122      # Read from input text
123      transactions = []
124      all_items = []
125      input_text = open(input_path, 'r')
126      lines = input_text.readlines()
127      total_transaction = len(lines)
128      input_text.close()
129
130      for line in lines:
131          items = list(map(int, line.rstrip().split("\t")))
132          all_items.extend(items)
133          transactions.append(items)
```

- main 함수의 첫 파트이다
- start 변수는 본 과제가 resonable time안에 수행되는지 확인을 위해 사용했다.
- 118~120 line을 통해 알맞은 개수의 parameter가 들어오지 않으면 프로그램을 종료하도록 했다.
- 125~126 line에서는 input 파일의 경로의 파일을 열고 각 line을 읽어서 tab으로 구분된 item들을 읽었다. 이 때 **각 line은 하나의 transaction**을 의미하고 이들을 transactions에 넣어주었다. Transaction의 개수를 알기 위해 **line의 수를 total_transaction에 저장**해주었다. item들의 종류를 알기위해 all_items에 모든 item들을 저장해 주었다.

```
135     nums = list(set(all_items))
136
137     # Find 1-itemset
138     # There is no duplication of items in each transaction
139     one_itemset = []
140     for num in nums:
141         if all_items.count(num) >= min_support * total_transaction:
142             one_itemset.append([num])
143
144     # Apriori Algorithm
145     k = 1
146     previous_itemset = one_itemset
147     output_txt = open(output_path, 'w')
```

- main 함수의 2번째 파트이다
- 앞에서 받은 모든 item들의 중복을 제거해서 nums에 담았다.
- 1-itemset을 구하기 위해 모든 item들을 받은 all_items에서 각 item(nums의 element)에 대해 count하여 support value가 minimum support보다 크거나 같은 item들을 one-itemset에 넣어주었다. (**이는 각 transaction은 중복되는 index의 item을 가지지 않는다는 과제 조건하에 가능했다**)
- 다음 Apriori 알고리즘을 위해 previous itemset에 1-itemset을 넣어주고 k를 1로 설정하였다. output text를 parameter로 받은 path로 파일을 write 모드로 open 했다.

```
148     while True:
149         # length k+1 구하는 과정
150         # self-joining
151         C = generate_candidates(k, previous_itemset)
152
153         if not C:
154             break
155
156         # pruning (association rule support,confidence와 함께 return / next candidate return)
157         previous_itemset, info = pruning(k, C, previous_itemset)
158
159         if info:
160             write_output_txt(info)
161         if previous_itemset:
162             k += 1
163         else:
164             break
165
166     output_txt.close()
167     print("time: ", time.time()-start)
```

- main 함수의 3번째 파트이다.

- generate_candidates 함수를 통해서 **k+1의 길이를 가지는 candidate**를 C에 저장한다. 만약 candidate가 생성되지 않았다면 반복문을 종료한다.
- pruning이라는 함수를 통해서 pruning을 거친 candidate를 support value가 minimum support 보다 크거나 같은 candidate만 남기도록 한다. pruning 함수 내에서 confidence를 계산하는 calc_confidence 함수를 통해 frequent한 association들에 대해 confidence까지 구해주었다. **전체 정보를 담은 리스트를 info에, 다음 step을 위한 frequent itemset을 previous itemset에 저장**해주었다.
- 그 후 저장할 association rule이 있으면 write_output_txt 함수를 통해 output 파일에 저장해주었다.
- previous itemset이 빈 리스트가 아니라면 k를 증가시켜 다음 step을 진행하도록 하고 **더 이상 frequent한 itemset이 없다면** 반복문을 탈출하도록 했다.
- 모든 동작이 끝난 뒤 output text 파일을 close하고 시작부터 걸린 시간을 출력하도록 했다.

```
6   def generate_candidates(k, previous):
7       itemset = []
8       for p in previous:
9           itemset.extend(p)
10      # 길이 2 itemset
11      if k == 1:
12          comb = combinations(itemset, 2)
13          candidates = []
14          for c in comb:
15              candidates.append(list(c))
16          return candidates
17      # 길이 3 이상 itemset
18      else:
19          candidates = []
20          itemset = list(set(itemset))
21          comb = combinations(itemset, k+1)
22          for c in comb:
23              candidates.append(list(c))
24          return candidates
```

- generate_candidates 함수에서는 **k+1의 길이를 가지는 candidate를 생성**하기 위해 k값과 previous itemset을 인자로 받는다.
- itemset에 previous의 item들을 저장한다. 길이 2의 candidate 생성과 나머지를

분리한 이유는 길이 3이상의 candidate에서는 previous itemset의 item들을 받다 보면 중복이 발생하기 때문이다. **가능한 item들을 중복없이 itemset에 모두 저장한 뒤 combinations함수를 통해 comb에 받아준다**. 이후 candidates에 list 형태로 변환하여 저장하여 return한다.

```
56  def pruning(k, candidates, itemset):
57      # length k subset for Test length k+1
58      test_itemset = []
59      for candidate in candidates:
60          test_itemset.append([candidate])
61          comb = combinations(candidate, k)
62          for c in comb:
63              test_itemset[-1].append(list(c))
64      check = [True for _ in range(len(test_itemset))]
65      # return info with support, confidence
66      info = []
67      next_candidate = []
```

- pruning 함수의 첫번째 파트이다. pruning 함수는 길이 k, pruning할 candidates, previous frequent pattern인 itemset을 인자로 받는다.
- test_itemset에는

  [길이 k+1 candidate, 조사해야 할 길이 k subset(1), 조사해야 할 길이 k subset(2), ...]

  와 같은 형태로 저장된다.
- check는 각 길이 k+1 candidate에 대해 frequent한지 여부를 저장하기 위해 test_itemset과 길이를 같은 boolean list이다.
- **info**는 ouput 파일에

  [*item_set*]₩t[*associative_item_set*]₩t[*support*(%)]₩t[*confidence*(%)]₩n

  형태로 저장하기 위해 해당 정보들을 담는 리스트이다.
- next_candidate는 다음 step을 위한 frequent한 itemset을 담는 리스트이다.

```
69      # subset이 previous에 있는지 확인 + min_support 적용
70      for i, test in enumerate(test_itemset):
71          T = test[1:]
72          for j in range(len(T)):
73              if T[j] not in itemset:
74                  check[i] = False
75                  break
76
77          if check[i]:
78              min_cnt = min_support * total_transaction
79              cur = test[0]
80              cnt = 0
81
82              for j in range(total_transaction):
83                  transaction = transactions[j]
84                  has_all = True
85                  for x in range(k+1):
86                      if cur[x] not in transaction:
87                          has_all = False
88                          break
89                  if has_all:
90                      cnt += 1
91
92              if cnt < min_cnt:
93                  check[i] = False
94              else:
95                  info.extend(calc_confidence(
96                      k, cur, round((cnt/total_transaction)*100, 2)))
97                  next_candidate.append(cur)
98
99      return next_candidate, info
```

- pruning 함수의 2번째 파트이다.
- 모든 test_itemset에 대해서 pruning후 support value를 계산한다. 72~75 line을 통해 **test할 길이 k의 subset이 frequent한지 확인**하고 아닌 경우 check 리스트에서 False값으로 갱신해준다.
- 모든 subset이 frequent한 경우 해당 **candidate가 frequent한지 확인하기 위해** 모든 transaction을 scan하며 candidate의 원소가 모두 포함된 transaction의 수를 count하여 해당 candidate의 support value값이 minimum support 이상인지 확인한다.
- minimum support이상의 pattern은 calc_confidence 함수를 통해 format에 맞게 output file에 저장하기 위한 반환값인 info에 저장된다. 또한 다음 step을 위한 previous frequent pattern을 next_candidate에 저장해준다.

- 모든 연산을 마친 후 next_candiate, info를 반환한다.

```
27  def calc_confidence(k, itemset, support):
28      ret = []
29      for i in range(1, k+1):
30          comb = list(combinations(itemset, i))
31          for j in range(len(comb)):
32              left = list(comb[j])
33              right = list(set(itemset) - set(left))
34              freq_left = 0
35              cnt = 0
36
37              for y in range(total_transaction):
38                  countable = True
39                  for x in range(i):
40                      if left[x] not in transactions[y]:
41                          countable = False
42                          break
43                  if countable:
44                      freq_left += 1
45                      for x in range(k+1 - i):
46                          if right[x] not in transactions[y]:
47                              countable = False
48                              break
49                  if countable:
50                      cnt += 1
51              ret.append([set(left), set(right), support,
52                          round((cnt/freq_left)*100, 2)])
53      return ret
```

- calc_confidence 함수에서는 길이 k값과 association rule을 생성할 itemset, support value를 인자로 받는다.
- 30 line에서 combination함수를 통해 **길이 1~ k의 itemset**을 뽑는다. 각 itemset에 대하여 left로 설정하고 33 line의 set을 이용한 **차집합** 연산을 이용해 itemset의 나머지를 right로 설정한다.
- 37~42 line에서 left가 포함된 transaction을 찾고 아닌 transaction은 countable flag를 통해 건너뛰도록한다.
- 43~48 line에서 left가 포함된 transaction의 개수를 freq_left로 세아리고 right가 포함된 transaction을 찾는다. countable flag는 43 line의 조건문을 이용해 들어오면 True이므로 이를 left와 right가 모두 포함된 transaction의 경우에만 True값을 유지하도록 한다. 그 후 49,50 line에서 left와 right가 모두 포함된 transaction의 개수를 count한다.

- 해당 함수가 return하는 ret 리스트에 출력 형식에 맞춰 left, right, support(**같은 itemset이므로 모든 association rule에 대해 같다**), confidence값을 저장해준다.

```
102    def write_output_txt(info):
103        for i in range(len(info)):
104            line = str(info[i][0]) + '\t' + str(info[i][1]) + \
105                '\t' + str('%.2f' % info[i][2]) + '\t' + \
106                str('%.2f' % info[i][3])
107            print(line, file=output_txt)
```

- write_output_txt 함수는 info 리스트를 parameter로 받는다.
- info에 저장된 정보를 과제 명세에 따른 format으로 line에 저장하여 output text에 write한다.

5. 실행 결과

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment1 (master)
λ python apriori.py 5 input.txt output.txt
time:  0.6749992370605469
```

output.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

{5}	{1}	10.00	39.68
{1}	{6}	7.00	23.49
{6}	{1}	7.00	30.97
{1}	{7}	7.00	23.49
{7}	{1}	7.00	29.17
{1}	{8}	15.40	51.68
{8}	{1}	15.40	34.07
{1}	{9}	9.60	32.21
{9}	{1}	9.60	34.53
{1}	{10}	10.20	34.23
{10}	{1}	10.20	35.17
{1}	{11}	7.40	24.83

```
C:\Users\user\Desktop\4-1\데이터사이언스\2021_ite4005_2016025478\Assignment1 (master)
λ python apriori.py 2 input.txt output.txt
time:  4.917505264282227
```

output.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

{16, 15}	{8, 5}	4.00	28.99	
{8, 5, 15}	{16}	4.00	80.00	
{8, 16, 5}	{15}	4.00	43.48	
{16, 5, 15}		{8}	4.00	68.97
{8, 16, 15}		{5}	4.00	46.51
{5}	{8, 16, 17}		2.60	10.32
{8}	{16, 17, 5}		2.60	5.75
{16}	{8, 17, 5}	2.60	6.13	
{17}	{8, 16, 5}	2.60	10.92	
{8, 5}	{16, 17}	2.60	20.63	
{16, 5}	{8, 17}	2.60	21.31	
{17, 5}	{8, 16}	2.60	40.62	