Programming Assignment #1

2017029916
양동해

1. 알고리즘 요약

Apriori 알고리즘을 이용해, association rule을 찾아내는 프로그램이다. 코드는 총 6가지 파트로 구분되며 각 파트당 알고리즘은 다음과 같다.

- 함수 선언
  - comb(itemset, number)
  - candToFreq(candidate_itemset, minimum_support)
- 파일 읽기
- 필요 변수 초기화 및 선언
- 초기값(C1, L1 itemset) 만들기
- apriori 알고리즘
- 파일 쓰기

2. 주요 코드 상세

코드를 위에서 언급한 6가지 파트로 나누어, 각 파트가 무엇을 하는지 기술하였다.

[함수 선언]

- comb(itemset, number)

```
# 인자로 받은 number에 맞게, combination한 itemset을 반환
def comb(itemset, number):
  if number == 2:
    result = list(combinations(itemset, number)) # 모든 조합 구하기
    return list(map(list, result)) # list로 변환
  else:
    # itemset의 아이템들이 tuple로 되어 있음 -> tuple을 list로 바꿈 (2차원 리스트 형성)
    # 2차원 리스트를 1차원 리스트로 변경 (sum 이용)
    # 중복제거(set 이용) 이후, 다시 list로 변환
    all_itemset = list(set(sum(list(map(list, itemset)), [])))

    tmp_result = list(combinations(all_itemset, number)) # 모든 조합 구하기
    tmp_result = list(map(list, tmp_result)) # list로 변환

    # itemset에서 만들어질 수 없는 조합 제거
    result = []
    itemset = list(map(set, itemset)) # tuple로 묶여있던 아이템셋을 set으로 변환
    for candidate_item in tmp_result:
      items_comb = list(combinations(candidate_item, number-1)) # 만들어낸 candidate_item에서 만들 수 있는 조합 생성
      check = True
      for tmp_item in items_comb: # 그 조합들이 itemset에 있는지 확인
        tmp_item = set(tmp_item)
        if tmp_item not in itemset:
          check = False
          break
      if check: result.append(candidate_item) # 모든 조합이 itemset에 존재하는 경우

    return result
```

comb() 함수는 itemset과 number를 인자로 받으며, 인자로 받은 number에 맞게 itemset을 combination한 값을 반환한다. 여기서 combination은 우리가 흔히 아는 "조합"과 같은 것이며, 전체 itemset의 크기를 n이라 하고 number를 r이라고 했을 때, nCr의 결과를 도출한다.
combination 연산은 itertools의 combinations 라이브러리를 이용했다.

number가 2인 경우는, itemset에 들어있는 원소들이 각각 하나의 아이템으로 이루어져 있으므로, 단순히 combination을 한 결과를 반환해주면 된다. (combinations 함수를 이용하면 조합된 값이 tuple 형태로 반환되기 때문에, 이를 list 형태로 바꾸어주는 코드를 추가했다.)

number가 3이상인 경우는, itemset에 존재하는 모든 원소들이 두개 이상의 아이템으로 구성된 경우인데, 이 때 모든 아이템 종류를 하나의 배열로 가져와 all_itemset에 담는다. (자세한 기능은 코드 주석에 첨부했다.) 그리고 all_itemset으로 만들 수 있는 모든 조합의 경우를 구한 뒤, 실제 itemset에서 만들어질 수 있는 조합만 result 배열에 추가하여 리턴한다. 그 조합이 만들어질 수 있는지에 대한 확인은 만들어진 조합(r개의 아이템으로 구성)을 다시 조합(해당 조합에서 r-1개의 조합을 다시 구함)하여, 그 조합들이 itemset에 있는지 확인하는 절차로 이루어진다.

- candToFreq(candidate_itemset, minimum_support)

```
# candidate 아이템셋을 frequent 아이템셋으로 변경 (support 체크)
def candToFreq(candidate_itemset, minimum_support):
  frequent_itemset = []
  for item, sup in candidate_itemset.items():
    if sup >= minimum_support:
      frequent_itemset.append(item)
  return frequent_itemset
```

candToFreq() 함수는 candidate_itemset을 frequent_itemset으로 변경한다. candidate_itemset은 파이썬에서 dict 자료구조로 되어 있어, 해당하는 아이템셋이 얼마의 support를 가지고 있는지 알 수 있다. (해시테이블과 같은 역할을 한다.) 따라서 그 아이템(key)에 해당하는 support값(value)을 보고 minimum_support와 비교해, 그 이상인 것들만 새로운 리스트에 추가하여 리턴한다.

[파일 읽기]

```
# 파일 읽기
transactions = [] # 파일을 읽은뒤, transaction들을 저장할 list

f = open(sys.argv[2], 'r')
lines = f.readlines()
for line in lines:
  transactions.append(list(map(int, line.split('\t'))))
f.close()
minimum_support = len(transactions) * (int(sys.argv[1]) * 0.01) # % 단위로 받은 minimum support를 갯수 단위로 환산하여 저장한 변수
```

파일 읽기 파트는 프로그램 실행 시 두번째 인자로 입력 받은 파일을 읽어드려, transactions 배열에 저장한다. input 파일은 정해진 형식에 맞게 쓰여 있으므로, 그에 맞는 형식으로 파싱하여 배열에 추가한다. 따라서 transactions 배열에는 하나의 transaction이 하나의 배열로 들어가, 2차원 배열의 형태를 갖는다. 그리고 첫번째 인자로 입력 받은 숫자를 통해, minimum_support를 계산하는데 이 때, 이 값은 퍼센트로 되어 있어 후에 다루기 쉽게 개수로 환산하여 저장한다.

[필요 변수 초기화 및 선언]

```
# 필요 변수 초기화 및 선언
k = 1           # 현재 아이템 조합 갯수 (k를 의미)
running = True  # 프로그램이 구동 중인지 확인하는 변수

candidate_itemset = {}  # candidate 아이템셋: dict로 관리
frequent_itemset = []   # frequent 아이템셋: array로 관리
result_itemset = []     # frequent 아이템셋 집합: 결과값
```

k는 현재 아이템셋들의 조합을 몇 개로 할건지(comb() 함수의 number에 들어갈 값) 저장하는 변수이며, running은 apriori 알고리즘을 계속해서 반복 할건지 확인하는 변수이다. 나머지 itemset 변수들은 apriori 알고리즘에서 사용될 변수이며, 변수명과 주석에 나와있는 내용과 같다.

[초기값(C1, L1 itemset) 만들기]

apriori 알고리즘을 실행시키기 위해서, 우선 k가 1인 경우의 frequent_itemset이 필요하다. 이 파트는 이를 계산하는 알고리즘으로, transactions 배열에 있는 transaction들을 보며, 모든 아이템들

을 하나씩 카운팅해 candidate_itemset에 저장하고, candToFreq() 함수를 이용하여 frequent_itemset을 만든다. 함수의 원형은 다음과 같다.

```
# 초기값(C1, L1 itemset) 만들기
for transaction in transactions: # C1 itemset 만들기
  for item in transaction:
    current_count = candidate_itemset.get(item, 0)
    candidate_itemset[item] = current_count + 1
frequent_itemset = candToFreq(candidate_itemset, minimum_support) # L1 itemset 만들기
```

[apriori 알고리즘]

```
# apriori 알고리즘
if len(frequent_itemset) == 0: running = False
while running:
  k += 1 # 아이템 조합 갯수 하나 증가

  tmp_candidate_itemset = comb(frequent_itemset, k) # 가능한 모든 itemset 조합 (candidate_itemset 만들기 위한 전처리)
  if len(tmp_candidate_itemset) == 0:
    running = False
    break

  candidate_itemset = {} # 초기화
  for tmp_candidate_item in tmp_candidate_itemset: # Ck itemset 만들기
    current_count = 0
    for transaction in transactions:
      check = True
      for item in tmp_candidate_item: # transaction에서 tmp_candidate_item의 아이템들이 포함되는지 확인
        if item not in transaction:
          check = False
          break
      if check: current_count += 1 # tmp_candidate_item 여기에 있는 아이템들이 transaction에 있는 경우
    candidate_itemset[tuple(tmp_candidate_item)] = current_count
  frequent_itemset = candToFreq(candidate_itemset, minimum_support) # Lk itemset 만들기

  # print('C',k,':', candidate_itemset)
  # print('L',k,':', frequent_itemset)
  if len(frequent_itemset) == 0:
    running = False
    break

  result_itemset.extend(frequent_itemset)
  # print('result_itemset:', result_itemset)
  # print('')
```

처음 이 코드가 실행될 시, frequent_itemset이 비어 있다면 코드를 중단하고 다음으로 넘어간다. frequent_itemset이 존재한다면, 그 아이템셋들로 조합할 수 있는 경우를 찾아 tmp_candidate_itemset에 추가한다. 그리고 그 배열에서 만들어진 조합을 하나씩 보며, 각각의 transaction들 안에 그 조합이 있는지 확인하고, 있다면 current_count를 1 증가 시킨다. 그리고 모든 transactions을 다 순회한 뒤, candidate_itemset에 기록한다. 위 과정을 반복해 모든 조합의 경우를 다 기록했다면, candToFreq() 함수를 이용하여 frequent_itemset을 만들고, 그 아이템셋에 원소가 하나라도 존재하는 경우, result_itemset에 넣고 전체 과정을 다시 반복한다.

이를 통해 우리는 frequent_itemset에 들어갈 조합을 하나씩 늘려가며 가능한 모든 경우를 탐색할 수 있다.

[파일 쓰기]

```
# 파일 쓰기
f = open(sys.argv[3], 'w')
for itemset in result_itemset:
  itemset_list = list(itemset)
  for i in range(1, len(itemset_list)):
    # itemset_list에 있는 것들로 조합을 만듦 -> 만들어진 결과물: [튜플, 튜플, ..]
    # tuple을 set으로 바꿈
    lefts = list(map(set, list(combinations(itemset_list, i)))) # itemset에서 좌변에 해당할 집합

    for left in lefts:
      right = set(itemset) - left # itemset에서 차집합을 통해 우변에 해당할 집합을 생성

      # support, confidence 계산
      support = 0
      left_appear = 0
      right_appear = 0
      for transaction in transactions:
        transaction = set(transaction) # 부분집합으로 연산하기 위해 set으로 변경
        if set(itemset) <= transaction: support += 1 # transaction에 해당 itemset이 있으면 support 증가
        if left <= transaction: # transaction에 left가 존재하면서 이 때,
```

```
            left_appear += 1
            if right <= transaction: right_appear += 1 # right도 존재하면 confidence 증가
        support = '{:.2f}'.format(support/len(transactions) * 100)    # 포맷에 맞게 변경
        confidence = '{:.2f}'.format(right_appear/left_appear * 100)  # 포맷에 맞게 변경

        line = str(left)+'\t'+str(right)+'\t'+str(support)+'\t'+str(confidence)+'\n'
        f.write(line)
f.close()
```

프로그램 실행 시, 세번째 인자로 입력 받은 파일에 association rule들을 기록한다. 위 알고리즘을 통해서 result_itemset에는 association rule에 해당하는 셋들만 들어가게 되는데, 이들을 하나씩 (꺼낸 아이템셋은 itemset 변수로 다루어진다.) 꺼내어 rule을 만든다.

우선 itemset에서도 조합을 통해 집합을 두 개(여기서는 left와 right로 구분)로 나눈다. 이 때 두 번째 반복문에선, left에 해당할 원소를 뽑는 코드가 존재하는데, 한 개씩 늘려가며 left를 확인한다. 그리고 right는 차집합을 이용하여 itemset에서 left를 빼내어 구한다. 마지막으로 그 left에서 right로 향하는 rule이 얼마의 support를 갖는지, 또 얼마의 confidence를 갖는지 계산해 파일에 저장한다.

3. 컴파일 및 실행

- Python version: 3.9.10
- 프로그램 실행: python3 apriori.py [minimum support] [input file name] [output file name]

```
eastsea@EastSeaui-iMac assignment1 % python3 --version
Python 3.9.10
eastsea@EastSeaui-iMac assignment1 % ls
apriori.py      input.txt       output.txt
eastsea@EastSeaui-iMac assignment1 % cat input.txt
1       3       4
2       3       5
1       2       3       5
2       5
eastsea@EastSeaui-iMac assignment1 % python3 apriori.py 50 input.txt output.txt
eastsea@EastSeaui-iMac assignment1 % cat output.txt
{1}     {3}     50.00   100.00
{3}     {1}     50.00   66.67
{3}     {2}     50.00   66.67
{2}     {3}     50.00   66.67
{3}     {5}     50.00   66.67
{5}     {3}     50.00   66.67
{2}     {5}     75.00   100.00
{5}     {2}     75.00   100.00
{2}     {3, 5}  50.00   66.67
{3}     {2, 5}  50.00   66.67
{5}     {2, 3}  50.00   66.67
{2, 3}  {5}     50.00   100.00
{2, 5}  {3}     50.00   66.67
{3, 5}  {2}     50.00   100.00
eastsea@EastSeaui-iMac assignment1 %
```