데이터 사이언스 Project1 Report

컴퓨터소프트웨어학부

2016024839 박정민

1. 코드 구성

Input file을 통해 읽은 전체 transaction과 minimum support로 구한 frequent pattern들을 저장할 class를 만들어서 관리하였습니다.

```
class FileOpen:

    def __init__(self, readFile, writeFile):
        self.fread = open(readFile, 'r')
        self.fwrite = open(writeFile, 'w')
        self.trx = []
        self.max_item = 0
        self.freq_itemSet = []
        self.freq_pattern = dict()
```

trx : 전체 transaction을 관리합니다.

max_item : item 번호 중 가장 큰 item입니다.

freq_itemSet : self join과 pruning과정을 거쳐 만들어진 최종적인 frequent itemset을 저장하고 관리합니다. 인덱싱을 하기 위해 List형태로 선언하였고, 개수 별 frequent itemset을 set타입으로 저장하였습니다.

freq_pattern : frequent itemset이 될 수 있는 candidate가 총 transaction에서 몇 번 나오는지 저장하고 관리하는 dictionary형태의 자료구조입니다. Candidate item set을 Key로 tuple타입으로 넣어주었습니다.

2. Class 내 method

```
def parseLine(self):
    lines = self.fread.readlines()

    file_length = len(lines)

    for line in lines:
        temp_str = ''
        parsed = []

        for parse in line:
            if parse != '\t' and parse != '\n':
                temp_str += parse

            else:
                parsed.append(int(temp_str))

                if self.max_item < int(temp_str):
                    self.max_item = int(temp_str)

                temp_str =''

        # Last line of the file doesn't contain '\t' or '\n'
        if line == lines[-1]:
                parsed.append(int(temp_str))

        #print("parsed line : " , parsed)

        self.trx.append(parsed)

    #print(self.trx)

    return self.trx
```

parseLine(self) : input.txt를 읽어 item set들을 parsing하는 함수입니다. 위에 언급한 클래스 내 인스턴스인 self.trx에 각 transaction을 리스트형태로 추가하였습니다.

```
def getRatio(self, val):
    return (val / len(self.trx)) * 100
```

getRatio(self, val) : val값을 parameter로 넣어주면 transaction의 개수에 대한 비율을 반환하는 함수입니다.

```python
# Get Frequent Itemset
def getItemSet(self, minSup):
    #print(minSup)

    # The number of elements of frequent item set
    freq_val = 1

    # First frequent set
    for itemList in self.trx:
        for item in itemList:
            item = (item,)
            if item not in self.freq_pattern.keys():
                self.freq_pattern[item] = 1
            else:
                self.freq_pattern[item] += 1

    first_set = []

    for key, value in self.freq_pattern.items():
        if self.getRatio(value) >= minSup:
            first_set.append(set(key))

    self.freq_itemSet.append(first_set)
```

getItemSet(self, minSup) : float 타입인 minSup을 parameter로 받아 itemset 후보들을 minSup을 기준으로 frequent itemset인지를 판단하는 함수입니다. 하나의 item으로 이루어진 frequent itemset과 두개 이상의 item들로 이루어진 frequent itemset을 구하는 과정을 분리하였습니다.

freq_val로 현재 몇 개의 item들로 이루어진 frequent itemset을 구하고 있는지 나타내었습니다. trx를 읽으면서 각 transaction별로 element를 가져와 tuple화 하여 freq_pattern에 key값으로 element를 가지고 있는지 판단하여 없으면 추가하고, 있으면 해당 value를 1씩 증가시킵니다. Counting이 끝났으면 value값을 getRatio 함수에 넣어 비율을 계산하여 minSup과 비교하여 하나의 item으로 이루어진 frequent itemset을 freq_itemSet에 추가하여줍니다. 각 원소는 나중에 self join을 용이하게 하기 위하여 set로 변환해준 후 list에 추가하였습니다.

```
# Second or more frequent set
while True:
    freq_val += 1

    #print("frequent value : ", freq_val)

    freq_length = len(self.freq_itemSet[freq_val - 2])

    #print("freq_length : ", freq_length)

    comb_set = set()

    # Gathering all of the items of frequent itemset
    for i in range(freq_length):
        comb_set = comb_set | self.freq_itemSet[freq_val - 2][i]
```

2개 이상의 item들로 이루어진 frequent itemset을 구하는 부분입니다. freq_val를 1씩 증가시키고, self join을 위해 바로 전의 frequent itemset에 접근하여 해당 itemset들을 join 연산으로 하나의 set(comb_set)로 만들어줍니다.

```
# Self Join
comb_set = list(itertools.combinations(comb_set, freq_val))

#print(comb_set)
#print("After self join : ", len(comb_set))

# Change tuple to set
for i in range(len(comb_set)):
    comb_set[i] = set(comb_set[i])
```

다음 frequent itemset을 구하기 위해 후보들을 만드는 self join부분입니다. Itertools 모듈의 combinations함수를 이용하여 freq_val개수로 이루어진 조합들을 만들어내어 comb_set에 저장하였습니다. combinations함수내의 원소들은 모두 tuple 타입으로 반환되므로, pruning을 용이하게 하기 위해 set 타입으로 바꾸어 줍니다.

```
# Pruning
# Copy comb_set list to iterate for loop
temp_comb_set = comb_set[:]

for candidate in temp_comb_set:

    candset = list(itertools.combinations(candidate, freq_val - 1))

    #print("candset : ", candset)

    for i in range(len(candset)):

        prune = 1

        # Change tuple to set
        candset[i] = set(candset[i])

        for prev in self.freq_itemSet[freq_val - 2]:

            if prev == candset[i]:
                prune = 0

        # Delete candidate from comb_set list
        if prune == 1:
            comb_set.remove(candidate)
            break
```

만들어진 후보들의 subset을 구하여 frequent itemset에 있는지 확인하여 subset 중 하나라도 없으면 걸러내주는 역할을 하는 Pruning을 실행하는 부분입니다. comb_set은 loop을 도는 동안 실시간으로 수정되기 때문에 복사본인 temp_comb_set을 만들어 temp_comb_set을 loop로 돌면서, 가지치기 해야하는 candidate가 있으면 comb_set에서 삭제하였습니다. Freq_val – 1 개수를 원소로 하는 subset들을 combinations함수를 통해 만든 다음, 각 subset별로 freq_itemSet에 있는지 확인합니다. Subset loop를 다 돌기 전에 하나라도 없으면 prune을 0으로 바꾸지 못합니다. 따라서 prune이 1이되는 loop에서 comb_set을 수정하고, loop를 탈출합니다.

```
# Making k + 1 frequent itemset
for candidate_set in comb_set:

    #print(candidate_set)

    for itemList in self.trx:

        itemList = set(itemList)

        # Check if transaction contains the candidate of k + 1 frequent itemset
        if candidate_set.issubset(itemList):

            temp_candidate_set = list(candidate_set)
            temp_candidate_set.sort()

            if tuple(temp_candidate_set) not in self.freq_pattern.keys():
                self.freq_pattern[tuple(temp_candidate_set)] = 1
            else:
                self.freq_pattern[tuple(temp_candidate_set)] += 1

# k + 1 frequent itemset
nth_set = []

# Check if candidates satisfiy minimum support
for key, value in self.freq_pattern.items():
    if len(key) == freq_val and self.getRatio(value) >= minSup:
        nth_set.append(set(key))

self.freq_itemSet.append(nth_set)
```

pruning까지 끝난 후보들을 transaction을 돌면서 counting하여 minSup보다 많이 등장하는 후보들을 list화하여 freq_itemSet에 추가하는 역할을 하는 frequent itemset을 만드는 함수입니다.

Tuple 내 원소의 순서가 바뀌면 dictionary내에 다른 key로 인식되기 때문에, dictionary에 추가하기 전에 list화하여 sort한 후, 다시 tuple타입으로 dictionary에 key로 넣어주었습니다.

```
if len(nth_set) == 0:
    break
```

만약 만들어진 frequent set이 없다면, loop를 탈출하여 apriori 알고리즘을 종료합니다.

```
def appAssociateRule(self):

    #print("Association Rule")

    iterate = len(self.freq_itemSet)
```

appAssociationRule(self) : 구한 frequent set을 중심으로 association rule을 구하여 output.txt에 저장합니다. Frequent itemSet이 총 몇 개의 원소로 이루어진 것까지 만들어졌는지 loop를 돌기위해 iterate변수에 그 수를 저장합니다.

```
for i in range(1, iterate):

    itemset = self.freq_itemSet[i]

    for freqset in itemset:

        freqset = list(freqset)
        freqset.sort()

        support = 0

        if len(freqset) == 1:
            support = self.freq_pattern[(freqset,)]

        else:
            support = self.freq_pattern[tuple(freqset)]

        for j in range(i):

            comb = list(itertools.combinations(freqset, j + 1))

            for item in comb:

                item = list(item)
                item.sort()

                confidence = support / self.freq_pattern[tuple(item)] * 100
                res_support = self.getRatio(support)

                counterpart = set(freqset) - set(item)

                temp_item = set(map(int, item))
                temp_counterpart = set(map(int, counterpart))

                line = str(temp_item) + '\t' + str(temp_counterpart) + '\t'
                line += str(round(res_support, 2)) + '\t' + str(round(confidence, 2)) + '\n'

                self.fwrite.write(line)
```

2개로 이루어진 frequent itemset부터 association rule을 적용하기 위해 iterate보다 1 적은 횟수의 loop을 수행합니다. 각 루프별로 frequent itemset을 갖고와서 association rule을 구할 조합을 만들어내기 위해 comb에 combinations함수를 적용하여 1개, 2개, ..., i(frequent itemset이 (i + 1)개의 원소로 이루어져있음)개로 이루어진 조합을 만들어냅니다.

Ex) (1, 2)와 같이 두 개의 원소로 이루어진 frequent itemset은 i 값이 1입니다.

이때까지 구한 candidate들의 빈도수가 저장되어 있는 freq_pattern의 dictionary에 key로 접근하기 위해 접근 전 모든 set들을 list화하여 sort한 후, 다시 tuple 타입으로 변환 후 key로 인덱싱합니다.

combinations함수로 만들어진 itemset을 item에 저장하고, associative_itemset을 차집합 연산을 이용하여 counterpart에 저장하였습니다. 그리고 support와 confidence를 freq_pattern에 저장되어 있는 각 candidate별 빈도수를 인덱싱하여 계산하였습니다.

3. 컴파일 환경 및 실행방법

Python 3.9.2 버전을 사용하였고 launch.json파일을 이용하여 input.txt와 minimum support, output.txt를 실행 parameter로 전달하였습니다.

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Current File",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "args": [
                "2.5",
                "input.txt",
                "output.txt"
            ]
        }
    ]
}
```

Parameter 수정 필요 시 launch.json 파일을 수정하시면 됩니다.

```
PS G:\내 드라이브\데이터사이언스\Project1>  g:; cd 'g:\내 드라이브\데이터사이언스\Project1'; & 'C:\Users\pmw14\AppData\Local
\Programs\Python\Python39\python.exe' 'c:\Users\pmw14\.vscode\extensions\ms-python.python-2021.3.658691958\pythonFiles\lib\p
ython\debugpy\launcher' '58353' '--' 'g:\내 드라이브\데이터사이언스\Project1\Apriori.py' '2.5' 'input.txt' 'output2.txt'
\launcher' '58396' '--' 'g:\내 드라이브\데이터사이언스\Project1\Apriori.py' '2.5' 'input.txt' 'output.txt'
```

Visual studio code 내의 python debug console창에서 실행한 스크린샷입니다.