

---

# Data Science Assignment#1

Find association rule using the Apriori algorithm using Python

Lee Eunah - 2018년 3월 20일

---

Course name: Data Science (ITE 4005)

Professor: Sang-Wook Kim

TAs: Jangwan Koo

Tae-ri Kim

Student name: Lee Eunah(이 은아)

Student number: 2016025769

Major: Computer Science Engineering

---

## Implementation environment

- OS: Mac OS 10.12.6
- Language: Python 3.6.3

## Summary of my algorithm

주어진 input 파일을 escape character를 제외한 데이터로 잘라서 transaction마다 리스트를 만들어서 넣어둡니다. 우선 전체 data를 scan해서 C1을 만듭니다. 그리고 난 후 data를 scan하면서 각 집합들이 transaction에 있는지 support count를 세어주고, command line argument로 받은 minimum support를 만족하는 candidate인지 확인(Lk)해줍니다. 이 과정을 해당 Ck에서 Lk값이 없을 때까지 반복해줍니다. ( $k = 1, 2, 3, \dots$ ) 이 때까지 나온 Lk값을 바탕으로 association rule을 적용하면 됩니다.

## Detailed description of my codes

```
import sys          # import sys module for using command line arguments
```

- command line arguments를 사용하기 위해 sys 모듈을 import한 것이다.

```
# Read input file data
def readInputFile(fileName):
    transactions = []
    inputfile = open("./"+fileName, 'r')
    lines = inputfile.readlines()
    for line in lines:
        transaction = line[:-1].split('\t')
        transactions.append(transaction)

    inputfile.close()
    return transactions
```

- input 파일을 읽어 transaction별로 데이터를 저장하는 function이다.
- fileName = input할 파일 이름
- 데이터는 기본적으로 [item]\t[item]\t[item]\n 이런 식으로 구성되어 있기 때문에 \n는 잘라주고 \t마다 split하면 각각의 item만 남게 된다. 그리고 transactions list에 넣어주면 된다.

---

```
def generateC1(data):
    C1 = []
    for transaction in data:
        for item in transaction:
            item_set = set()
            item_set.add(item)
            if item_set not in C1: # no duplicate element
                C1.append(item_set)

    return C1
```

- C1을 생성하는 function이다.
- data = 전체 data
- 전체 data를 scan하면서 1원소 집합으로 만들어서 C1을 생성한다.

```
# Count support count & Test minimum support
def scanDB(candidate, data, min_support):
    # support count
    item_candidate = {}
    for transaction in data:
        for item in candidate:
            if item.issubset(transaction):
                item = list(item)
                item.sort()
                item = tuple(item)
                if not item in item_candidate: # Count item
                    item_candidate[item] = 1
                else:
                    item_candidate[item] += 1

    # test minimum support
    Lk={}
    for key, value in item_candidate.items():
        support = (value / len(data)) * 100 # %
        if support >= min_support: # Check minimum support
            Lk[key] = support

    return Lk
```

- 각 item의 support count와 minimum support를 만족하는지 확인하는 function이다.
  - candidate = C<sub>k</sub> (k = 1, 2, 3, ... )
- data = 전체 data  
min\_support = command line argument에서 받아온 min\_support 값
- k=1, 2, 3, ... 일 때, C<sub>k</sub>의 item이 transaction에 있다면 support count를 1 증가시킨다.
  - 각 item의 support count를 바탕으로 support 값을 계산해서 minimum support를 만족시키는지 확인하고, 만족하면 L<sub>k</sub>에 추가한다. (L<sub>k</sub>의 item과 support값은 나중에도 필요한 부분이기 때문에 dictionary type으로 하도록 한다.)
  - Support = (해당 아이템을 포함하는 transaction의 개수)/(전체 transaction의 개수)

---

```
def generateCandidate(Lk, k):
    LkPlus = []
    # generate all candidate
    for i in range(len(Lk)):
        for j in range(i+1, len(Lk)):
            first = set(Lk[i])
            second = set(Lk[j])
            item_union = first | second
            if k == len(first | second) and item_union not in LkPlus:
                LkPlus.append(item_union)

    return LkPlus
```

- Ck를 생성하는 function이다.(k = 2, 3, ...)
- Lk = minimum support를 만족하는 Ck

k = k개의 원소를 가지는 집합

- 가능한 모든 candidate를 생성하기 위해 각 item을 자기 자신을 제외하고 나머지 item과의 합집합을 구하는 방법을 사용했다.

```
def associationRule(Lk, item_support, k, file):
    if k != 2:
        file = open("./" + outputFile, 'a')

    Lk = list(Lk.keys())
    item_list = list(item_support.keys())
    support_list = list(item_support.values())

    for index in Lk:
        n = 0
        index = list(index)
        item = set(index)
        check = False

        if k == 2:
            index, check = subset(index, k, check)

        else:
            while n < k-2:
                index, check = subset(index, k, check)
                n = n + 1

        for association in index:
            association = set(association)
            rest = item - association
            association_sort = list(tuple(association))
            association_sort.sort()
            association_sort = tuple(association_sort)
            union_sort = list(tuple(association | rest))
            union_sort.sort()
            union_sort = tuple(union_sort)

            # Calculate support and confidence
            support = round(item_support.get(union_sort), 2)
            confidence = round(((item_support.get(union_sort)) / (item_support.get(association_sort))) * 100, 2)

            # write to file
            file.write(str(association) + '\t' + str(rest) + '\t' + str(support) + '\t' + str(confidence) + '\n')
    file.close()
```

---

- Lk(k=1, 2, 3, ...)에 대한 association rule을 적용시키고 output 파일에 그 결과를 쓰는 function이다.

- Lk = minimum support를 만족하는 Ck

item\_support = Lk의 각 item과 support를 저장하고 있는 dictionary type

k = k개의 원소를 가진 집합

file = open()함수를 가리키는 것

- 이 함수는 main문에서 while문 안에 위치해 있기 때문에 처음에는 파일을 'w'타입으로 새로 파일을 생성하거나 파일이 기존에 있다면 덮어쓰는 것으로 했고, 그 이외에는 'a'타입을 써서 해당 파일에 내용을 이어서 추가 하는 코드로 구현을 하였다.
- Association rule을 구현하기 위해 Lk item의 모든 부분 집합을 구한 후, 해당 item의 부분 집합 그리고 그 집합과 해당 item 차집합을 연관시켜 주는 코드로 구현했다.
- 각 association의 support값과 confidence를 구해서 파일에 write한다.

```
def subset(Lk_part, k, check):
    subset_item = []
    Lk_set = []

    if check == False:
        check = True
        for i in Lk_part:
            subset_set = set()
            subset_set.add(i)
            Lk_set.append(subset_set)

    else:
        Lk_set = Lk_part

    for i in Lk_set:
        for j in Lk_set:
            part_union = i | j
            if len(part_union) < k and part_union not in subset_item:
                subset_item.append(part_union)

    return subset_item, check
```

- 부분 집합을 구하는 function이다.

- Lk\_part = Lk의 부분이라는 뜻으로 사용. 만약 3원소 집합이라면 subset을 2번 돌아야하기 때문에 한 번 돌았다고 해서 완전한 Lk가 아니라서 이렇게 명명함.

k = k개의 원소를 가진 집합

check = 처음 subset 함수로 들어왔을 때 Lk\_part의 type은 set이 아니다. 그래서 최초에만 set타입으로 지정해주면 되기 때문에 check로 set타입으로 지정했는지의 여부를 따진다.

---

```

# main

# Check the command line arguments
if len(sys.argv) != 4:
    print('Please fill in the command form.
    Executable_file minimum_support inputfile outputfile')
    exit()

min_support = int(sys.argv[1])
inputFile = sys.argv[2]
outputFile = sys.argv[3]

data = readInputFile(inputFile)
C1 = generateC1(data)
Lk = scanDB(C1, data, min_support)

item_support_data = Lk
Lk_before = Lk

file = open("./" + outputFile, 'w')

# repeat until end of the apriori
k = 2
while True:
    Ck = generateCandidate(list(Lk.keys()), k)

    if not Ck:
        break;

    Lk = scanDB(Ck, data, min_support)

    if not Lk:
        break;

    Lk_before = Lk
    item_support_data.update(Lk)

    if not Lk:
        associationRule(Lk_before, item_support_data, k, outputFile)
    else:
        associationRule(Lk, item_support_data, k, file)

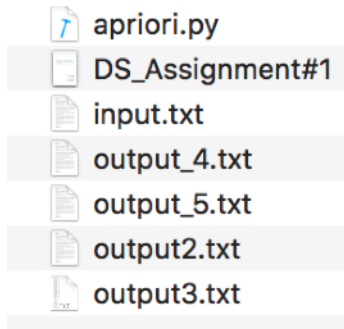
    k = k + 1

```

---

- main function
- Command line argument 4개를 받아야 하므로 그 이외의 경우에는 error command를 적었다.
- While문 안에서 Lk\_before를 설정한 이유는, 만약에 Ck가 생성이 되고 minimum support를 만족하는지 검사했을 때 어떠한 candidate도 만족하지 않으면 Lk의 값은 비어있게 된다. 이럴 경우 코드는 에러가 나게 된다. 따라서 Lk가 비어있으면 그 전의 Lk를 associationRule 함수에 보내야 하므로 Lk\_before를 설정하였다.

## Compiling my code



Macintosh HD > 사용자 > leeeunah > 데스크탑 > Class > 3-1 > Data Science

같은 디렉토리에 apriori.py와 input.txt를 넣어둡니다.

제 컴퓨터에서는 다음과 같은 경로에 apriori.py가 존재하므로 터미널에서 Data Science 디렉토리로 이동 후 코드를 실행하도록 한다.

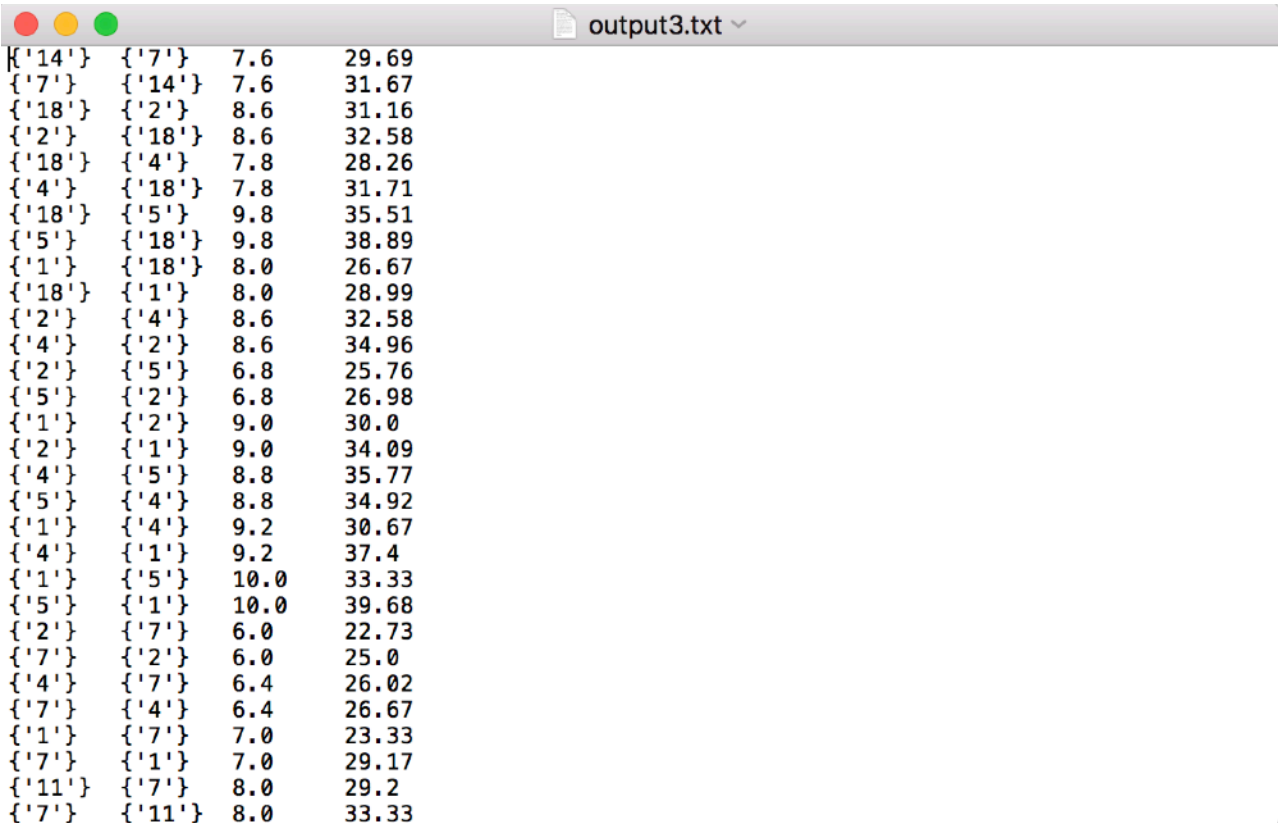
\* minimum support = 2% 일 때,  
output2.txt는 전체 문서 중 일부를 찍은 것이다.

```
Data Science — -bash — 80x24
Last login: Tue Mar 20 10:06:58 on ttys000
Eunahui-MacBook-Pro:~ leeeunah$ cd Desktop/Class/3-1/Data\ Science/
Eunahui-MacBook-Pro:Data Science leeeunah$ python apriori.py 2 input.txt output2
.txt
Eunahui-MacBook-Pro:Data Science leeeunah$
```

output2.txt			
{'14'}	{'7'}	7.6	29.69
{'7'}	{'14'}	7.6	31.67
{'18'}	{'2'}	8.6	31.16
{'2'}	{'18'}	8.6	32.58
{'18'}	{'4'}	7.8	28.26
{'4'}	{'18'}	7.8	31.71
{'18'}	{'5'}	9.8	35.51
{'5'}	{'18'}	9.8	38.89
{'1'}	{'18'}	8.0	26.67
{'18'}	{'1'}	8.0	28.99
{'2'}	{'4'}	8.6	32.58
{'4'}	{'2'}	8.6	34.96
{'2'}	{'5'}	6.8	25.76
{'5'}	{'2'}	6.8	26.98
{'1'}	{'2'}	9.0	30.0
{'2'}	{'1'}	9.0	34.09
{'4'}	{'5'}	8.8	35.77
{'5'}	{'4'}	8.8	34.92
{'1'}	{'4'}	9.2	30.67
{'4'}	{'1'}	9.2	37.4
{'1'}	{'5'}	10.0	33.33
{'5'}	{'1'}	10.0	39.68
{'2'}	{'7'}	6.0	22.73
{'7'}	{'2'}	6.0	25.0
{'4'}	{'7'}	6.4	26.02
{'7'}	{'4'}	6.4	26.67
{'1'}	{'7'}	7.0	23.33
{'7'}	{'1'}	7.0	29.17
{'11'}	{'7'}	8.0	29.2
{'7'}	{'11'}	8.0	33.33

\* minimum support = 3%일 때,  
output3.txt는 전체 문서 중 일부를 찍은 것이다.

```
Eunahui-MacBook-Pro:Data Science leeeunah$ python apriori.py 3 input.txt output3.txt
Eunahui-MacBook-Pro:Data Science leeeunah$
```



{'14'}	{'7'}	7.6	29.69
{'7'}	{'14'}	7.6	31.67
{'18'}	{'2'}	8.6	31.16
{'2'}	{'18'}	8.6	32.58
{'18'}	{'4'}	7.8	28.26
{'4'}	{'18'}	7.8	31.71
{'18'}	{'5'}	9.8	35.51
{'5'}	{'18'}	9.8	38.89
{'1'}	{'18'}	8.0	26.67
{'18'}	{'1'}	8.0	28.99
{'2'}	{'4'}	8.6	32.58
{'4'}	{'2'}	8.6	34.96
{'2'}	{'5'}	6.8	25.76
{'5'}	{'2'}	6.8	26.98
{'1'}	{'2'}	9.0	30.0
{'2'}	{'1'}	9.0	34.09
{'4'}	{'5'}	8.8	35.77
{'5'}	{'4'}	8.8	34.92
{'1'}	{'4'}	9.2	30.67
{'4'}	{'1'}	9.2	37.4
{'1'}	{'5'}	10.0	33.33
{'5'}	{'1'}	10.0	39.68
{'2'}	{'7'}	6.0	22.73
{'7'}	{'2'}	6.0	25.0
{'4'}	{'7'}	6.4	26.02
{'7'}	{'4'}	6.4	26.67
{'1'}	{'7'}	7.0	23.33
{'7'}	{'1'}	7.0	29.17
{'11'}	{'7'}	8.0	29.2
{'7'}	{'11'}	8.0	33.33

## Any other specification about my code

코드 구현을 하면서 조금 어려웠던 것들이나 에러를 해결하기 위해 했던 생각들에 대해서 정리를 해보고자 한다.

1. Dictionary type의 key는 불변이기 때문에 set, list 타입들은 key값이 될 수 없다. Apriori를 구현하면서 모든 것을 set type으로 관리하면 편하긴 하지만 key값의 설정을 위해 tuple type으로 변경하면서 구현하였다.
2. list와 다르게 set는 order가 없다. 그래서 dictionary type에서 key값을 넘겨 줄 때 set type을 list type으로 변환한 후 sort를 해주어야 나중에 다시 꺼내어 쓰기가 편했다.
3. subset함수를 구현하는 것이 어려웠다. Candidate를 생성할 때와 비슷한 원리이긴 하지만 3원소 집합일 때 한 번에 모든 집합을 구할 수 있는 방법에 대해 생각해보았다.
4. 초기에 subset함수에서 인자로 받은 list type의 Lk\_part의 각 원소를 set type으로 만들고 싶어서 이중 for문을 사용해서 코드를 구현하였다. 그런데 `TypeError: unhashable type: 'list'` 이런 에러가 발생하게 되었다. List 안에는 unhashable type 즉, set이 들어갈 수 없다 라는 뜻으로 이해를 했다. set이라는 것이 order가 없기 때문에 호출될 때마다



---

order가 다르기 때문에 for문을 돌 때 제약이 생기는 것이 아닐까? 라는 생각을 하였고 코드를 수정하였다.