[Data Science] 1. Apriori

Environment & Goal

- *OS* : Windows
- *Language* : Python
- *Goal* : find association rules using the **Apriori** algorithm

Summary of algorithm

- 전체적인 Algorithm flow는 다음과 같다.
  1. Input path를 통해 .txt file에 접근하여 Transaction들을 줄마다 읽고 ***preprocess***를 통해 split & append
  2. 1번의 return value를 ***class Transactions***의 db에 할당한다.
  3. 2번 객체와 sys.argv[1]를 support로 하는 ***class Apriori***에서 사용되는 method의 흐름은 다음과 같다.
     a. 초기 TDB를 통해 Length가 1인 frequent patterns을 생성 ~ *Initialize*
     b. self-join을 통해 Length가 N인 Candidate set을 생성 ~ *join*
     c. pruning을 통해 min support보다 작은 support를 가지는 pattern들을 모두 pop ~ *pruning*
     d. b, c의 과정을 다음 candidate set이 비어있을 때까지 반복 ~ *mining*
  4. 3번 객체를 통해 만들어진 Frequent Pattern들에 대해, ***class Analyzer***에서 Association rule을 추출한다.
  5. 마지막으로 output path를 통해 .txt file을 열어 extracted association rule을 저장한다.

```
support = sys.argv[1]
input_path = sys.argv[2]
output_path = sys.argv[3]

db = preprocess(input_path)
app = Apriori(db, int(support) * 0.01 * len(db))
freqs = app.mining()

analyzer = Analyzer(freqs, len(app.db.database))
```

```
res = []
for i in range(1, len(freqs)):
   res.append(analyzer.analyze(i))

recording(output_path, res)
```

Detailed description of codes

1. def preprocess(path)

```
def preprocess(path):
    transactions = []
    input_ = open(path, 'r')
    while True:
        ta = input_.readline()
        if not ta:
            break
        temp = ta.split()
        ta_ = list(map(int, temp))
        transactions.append(ta_)
    input_.close()

    return np.array(transactions, dtype=object)
```

- path를 인자로 하여 input.txt file을 열고 각 transaction line에 대해 split 후 결과 list에 append하여 이 결과를 반환한다.

2. class Transaction

```
class Transactions:
    def __init__(self, ta):
        self.database = ta

    def count(self, x):
        cnt = 0
        for data in self.database:
            if len(data) >= len(x):
                res = np.isin(data, x) # res 내의 True 개수가 data와 x 내의 교집합 개수
                if res.sum() == len(x):
                    cnt += 1

        return cnt
```

- 크게 db라는 field와 count라는 method를 가지고 있다.
    - db는 def preprocees의 return value를 저장

- count는 db 내의 각 Transaction에 대해 item set x가 얼마나 존재하는 지 반환

3. class Apriori

```
class Apriori:
    def __init__(self, ta, sup):
        self.db = Transactions(ta)
        self.support = sup

    def initialize(self):
        li = np.sort(np.unique(sum(self.db.database, [])))
        init_set = np.empty((0, 2))
        for element in li:
            cnt = 0
            for data in self.db.database:
                if np.isin(element, data):
                    cnt += 1
            if cnt >= self.support:
                init_set = np.append(init_set, np.array([[element, cnt]]), axis=0)

        return init_set

    def join(self, candidates, level):
        size = len(candidates)
        freqs = []
        temp = []
        for i in range(0, size):
            for j in range(i + 1, size):
                itemset = np.union1d(candidates[i][0], candidates[j][0]) # 2 element의 합집합
                if np.size(itemset) == level + 1:
                    count = self.db.count(itemset)
                    temp.append([itemset, count])

        for tmp in temp: # 중복 제거
            b = False
            for data in freqs:
                if np.array_equiv(tmp[0], data[0]):
                    b = True
            if not b:
                freqs.append(tmp)

        return freqs

    def pruning(self, freqs):
        cand = freqs.copy()

        for i in range(0, len(freqs)):
            if freqs[i][1] < self.support:
                cand.pop(i - len(freqs) + len(cand))

        return cand

    def task(self, freqs, level):
        cand = self.join(freqs, level)
        return self.pruning(cand)
```

```
    def mining(self):
        cnt = 1
        init = self.initialize()
        cand = [list(init)]
        cand_ = self.task(init, cnt)
        cand.append(cand_)
        cnt += 1

        while not len(cand_) == 0:
            cand_ = self.task(cand_, cnt)
            if not len(cand_) == 0:
                cand.append(cand_)
            cnt += 1

        return cand
```

- *def initialize(self):*
  - sum(self.db.database, [])를 통해, 모든 Transaction들을 하나의 Array에 저장하고 np.unique를 통해 중복되는 element를 제거
  - 이후 각 element에 대한 support를 구한 후, minimum support보다 크거나 같은 값을 가지는 element에 대해 {element, support}를 pair로 저장
- *def join(self, candidates, level):*
  - *Arguments*
    - candidates : 이전 level에서 받아온 frequent patterns
    - level : frequent pattern의 Length
  - 전체 N개의 frequent patterns에서 2개를 골라 np.union1d를 통해 새로운 item set을 생성
  - 이후, level + 1의 candidates만 생성하기 위해 길이가 level + 1인 item set들만 count를 호출하고 append
  - 이후 중복되는 candidates에 대해 중복 제거
- *def pruning(self, freqs):*
  - *Argument*
    - freqs : def join의 return value
  - freqs 내의 item set 중 minimum support보다 낮은 support를 가지는 item set들에 대해 pruning을 진행
- *def task(self, freqs, level):*
  - *Arguments*

        - freqs: 이전 level에서 전달받은 candidate set
        - level : 현재 task를 수행하는 level
    - 현재 level에서 진행하는 self-joining과 pruning을 하나의 Task로 정의
- *def mining(self):*
    - class Apriori에서 수행하는 가장 high level task
    - initialize를 호출하여 length가 1인 frequent patterns 생성
    - cand_에 length가 2인 frequent patterns 생성 후, cand_가 비게 될때까지 task를 호출

4. class Analyzer

```
class Analyzer:
    def __init__(self, freq, size):
        self.freq = freq
        self.size = size

    def analyze(self, level):
        res = []

        n = int((level + 1) / 2) + 1
        for i in range(1, n):
            set_x = self.freq[i - 1]
            set_y = self.freq[level - i]

            for x in set_x:
                for y in set_y:
                    items = np.union1d(x[0], y[0])
                    # 아래 조건문을 통해 교집합이 없는 association rule만을 반환하게 한다
                    if not len(items) == level + 1:
                        continue

                    for data in self.freq[level]:
                        sup = 0
                        if np.array_equiv(items, data[0]):
                            sup = data[1]

                        if sup == 0:
                            continue

                        conf_xy = format(sup / x[1] * 100, ".2f")
                        conf_yx = format(sup / y[1] * 100, ".2f")
                        sup = format(sup / self.size * 100, ".2f")
                        array_x = np.asarray(x[0], dtype=int)
                        array_y = np.asarray(y[0], dtype=int)

                        element_x = "{"
                        element_y = "{"
                        for i in array_x:
```

```
                            element_x += str(i)
                            if not i == array_x[-1]:
                                element_x += ", "

                        for i in array_y:
                            element_y += str(i)
                            if not i == array_y[-1]:
                                element_y += ", "

                        element_x += "}\t"
                        element_y += "}\t"

                        first = element_x + element_y + str(sup) + "\t" + str(conf_xy)
                        second = element_y + element_x + str(sup) + "\t" + str(conf_yx)
                        res.append(first)
                        res.append(second)

        return set(res) # 중복 제거
```

- *def analyze(self, level):*
  - 현재 level의 pattern length를 N이라고 했을 때, N을 2개의 set으로 분할하는 경우는 총 (1, N-1), (2, N-2), ... , (N-1 ,1)이다.
  - (K, N-K) 일때, 길이가 K인 frequent pattern에서 하나를, 그리고 길이가 N-K인 frequent pattern에서 하나를 가져와 association rule로 묶고 support와 confidence를 구해준다.
  - 그리고 알맞은 출력 format에 맞게 string으로 변환해준 다음, set 연산을 통해 중복을 제거하여 반환한다.

5. def recording(path, res)

```
def recording(path, res):
    output_ = open(path, 'w')
    for set_ in res:
        for line in set_:
            output_.write(line + "\n")

    output_.close()
    print("Task Completed.")
```

- 결과를 output.txt에 적는 함수

Result & Testing

- input.txt : LMS 과제란에 포함된 input.txt를 기준으로 함
- 실행 결과
  - 총 1066개의 association rule이 출력

```
PS C:\Users\USER\PycharmProjects\data_science_2022\1_Apriori> python .\apriori.py 5 .\input.txt .\output.txt
Task Completed.
```

- 출력 결과인 output.txt를 점검하기 위해 아래와 같은 program 작성
  - 단순하게 중복되는 rule과 minimum support를 넘지 못하는 rule들에 대한 점검

```
PS C:\Users\USER\PycharmProjects\data_science_2022\1_Apriori> python .\test.py 5 .\output.txt
----------[Deduplication Check]----------
Before : 1066
After : 1066
----------[Minimum Support Check]----------
Insufficient rule : 0
Finished.
```

```
import sys

sup = float(sys.argv[1])
input_path = sys.argv[2]
f = open(input_path, 'r')
lines = []
while True:
    line = f.readline()
    if not line:
        break
    lines.append(line)
f.close()

before = len(lines)
lines = set(lines)
after = len(lines)
cnt = 0

for line in lines:
    li = line.split('\t')
    if float(li[2]) < sup:
        cnt += 1

print("----------[Deduplication Check]----------")
print("Before : " + str(before))
print("After : " + str(after))
print("----------[Minimum Support Check]----------")
print("Insufficient rule : " + str(cnt))
print("Finished.")
```