Assignment #1

Apriori algorithm

한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
데이터사이언스(ITE4005, 12877) – 김상욱 교수님

---

1. 개요

Apriori 알고리즘은 연관규칙(association rule)의 대표적인 형태로써, 데이터들에 대한 발생빈도(빈발, frequent)를 기반으로 각 데이터간의 연관 규칙 탐색을 위한 방법니다. Apriori 알고리즘은 transaction data에서 빈번한 개별 항목(frequent itemset)을 찾고 해당(k) itemset을 기준으로 점점 더 큰(k+1) itemset으로 확장해가며 진행된다.

2. 목적

이 프로젝트의 목적은 주어진 tansaction data와 minimum support를 가지고 apriori 알고리즘을 통해 연관 규칙 탐색을 구현한다. 그리고 각각의 빈번항목집합마다 association rule을 적용하여 support 와 confidence를 구한다.

3. 개발환경

```
OS : macOS Big Sur 11.2.2
Language : Python 3.8.8
Source code editor : Visual Studio Code
```

4. 프로그램 구조 및 알고리즘

1. minimum support, input.txt, output.txt를 입력받는다.
2. input.txt의 데이터를 읽고 table에 저장한다.
3. 해당 table에서 1-candidate itemset을 가져온다.
4. 1-candidate itemset에서 1-frequent itemset을 도출한다.
5. (k-1)-frequent itemset에서 k-candidate itemset을 도출한다.
6. Self-joining과 pruning을 통해 k-candidate itemset에서 k-frequent itemset을 도출한다.
7. 도출된 k-frequent itemset을 frequent list에 저장한다.
8. 다시 5번 과정으로 돌아가 k-candidate itemset 혹은 k-frequent itemset이 없을때 까지 5~7번 과정을 반복한다.
9. 최종 frequent list에서 각 itemset과 associative itemset의 support와 confidence 값을 구하고 output에 저장한다.
10. 저장된 결과값 output을 output.txt에 쓰는 것으로 프로그램을 종료한다.

5. 코드 설명

```
'''
Read file
input 파일의 데이터를 불러와 db에 저장한다.
'''
def read_file(input_file):
    global db, min_sup

    f = open(input_file, 'r')
    lines = f.readlines()
    for line in lines:
        items = line.split()
        items = set(int(item) for item in items)
        db.append(items)
    f.close

    ## minimum support(%)를 count(num)로 변환한다.
    min_sup = len(db) * (min_sup/100)
```

- `read_file` : argv 인자로 받은 input_file에서 data를 가져와 전역변수로 선언된 db에 저장한다. argv의 다른 인자로 받은 min_sup은 사용의 편의성을 위해 %단위에서 count 할 수 있도록 변환한다.

```
'''
Generate 1-candidate itemset
db로부터 1-candidate itemset을 만든다.
'''
def generate_first_candidate(db):
    C1_sup = defaultdict(int)

    for items in db:
        for item in items:
            item = [item]
            C1_sup[tuple(item)] += 1

    return C1_sup
```

- `generate_first_candidate` : db에서 1-candidate itemset을 도출한다. support count를 알기 위해 db를 처음부터 스캔하며 item의 수를 계산한다. 연산이 끝나면 1-candidate itemset과 support count를 return한다.

```
'''
Generate k-candidate itemset
(k-1)-frequent itemset으로부터 k-candidate itemset을 만든다.
'''
```

```
def generate_candidate(Lk, Lk_sup, k):
    ## initialize k-candidate itemset
    Ck = []
    Ck_sup = defaultdict(int)

    for i in range(len(Lk)):
        for j in range(i+1, len(Lk)):
            L_sub1 = Lk[i][:-1]
            L_sub2 = Lk[j][:-1]

            if L_sub1 == L_sub2:

                ## self_joining in sorted
                if Lk[i][-1] < Lk[j][-1]:
                    candidate = Lk[i] + tuple([Lk[j][-1]])
                else:
                    candidate = Lk[j] + tuple([Lk[i][-1]])
                ## pruning
                Ck, Ck_sup = prune(Ck, Ck_sup, candidate, Lk_sup, k)

    return Ck, Ck_sup
```

- `generate_candidate` : (k-1)-frequent itemset으로부터 k-candidate itemset을 뽑아내는 함수다. 정렬 상태를 유지하며 self-joining을 한다. Join을 마친 candidate가 있다면 `prune` 함수를 호출해 k-candidate에 들어갈 수 있는 조건을 만족하는지 확인한다. Self-join과 prune의 과정을 모두 거치면 도출된 k-candidate itemset과 support count를 return한다.

```
'''
Pruning
support가 minimum support를 만족시키지 못 할 경우 제거한다.
'''
def prune(Ck, Ck_sup, candidate, Lk_sup, k):
    for i in range(k+1):
        subset = candidate[:i] + candidate[i+1:]
        sup = Lk_sup[subset]

        ## delete if candidate's support doesn't satify minimum support
        if sup < min_sup:
            del(candidate)
            return Ck, Ck_sup

    ## add candidate if satisfy minimum support
    Ck.append(candidate)
    Ck_sup[candidate] = get_sup_cnt(candidate)

    return Ck, Ck_sup
```

- `prune` : 위의 함수에서 도출된 candidate의 support가 minimum support를 만족하는지 확인한다. 조건을 만족하지 못한다면 candidate를 삭제하고, 만족한다면 k-candidate itemset에 추가한다.

```
'''
Get candidate's support count
candidate의 support count를 db로부터 가져온다.
'''
def get_sup_cnt(candidate):
    cnt = 0
    candidate = set(candidate)

    for trx in db:
        if candidate.issubset(trx):
            cnt += 1

    return cnt
```

- `get_sup_cnt` : candidate의 support count를 계산하기 위한 함수다. db를 처음부터 확인하며 candidate가 db의 부분집합을 만족하는지 확인하며 count를 계산한다. 계산이 끝나면 cnt를 return 한다.

```
'''
Generate k-frequent itemset
k-candidate itemset으로부터 k-frequent itemset을 만든다.
'''
def generate_frequent(Ck_sup):
    ## initialize k-frequent itemset
    Lk = []
    Lk_sup = defaultdict(int)

    for itemset, sup in Ck_sup.items():
        if sup >= min_sup:
            Lk.append(itemset)
            Lk_sup[itemset] = sup

    return Lk, Lk_sup
```

- `generate_frequent` : k-candidate itemset으로부터 k-frequent itemset을 만든다. k-candidate itemset의 각 itemset들의 support가 min_sup를 만족신다면 k-frequent itemset에 추가한다. 함수가 종료될때 k-frequent itemset과 support count를 return한다.

```
'''
Write file
저장된 frequent itemset을 output 파일에 입력한다.
```

```
'''
def write_file(output_file, frequent):
    of = open(output_file, 'w')
    for k in range(len(frequent)):

        for itemset in frequent[k]:
            ## get subsets of itemset
            length = len(itemset)
            subsets = []

            for i in range(1 << length):
                subset = set(itemset[j] for j in range(i) if (i & (1 <<
j)))
                if subset != set() and subset != set(itemset):
                    subsets.append(subset)

            ## calculate support and confidence of subset_1 and subset_2
            ## and make data with in conditions
            for subset_1 in subsets:
                subset_2 = set(itemset) - subset_1

                ## make tuple and sort union(subset_1, subset_2) and
subset_1
                sorted_union = tuple(sorted(subset_1.union(subset_2)))
                sorted_subset_1 = tuple(sorted(subset_1))

                ## get support from each itemset
                sup_union = frequent_sup[sorted_union]
                sup_subset_1 = frequent_sup[sorted_subset_1]

                ## calculate support and confidence of subset_1 and
subset_2
                sup = format(sup_union / len(db)*100, ".2f")
                conf = format(sup_union / sup_subset_1*100, ".2f")

                ## convert itemset list to string
                str_subset_1 = ",".join(str(s) for s in sorted(subset_1))
                str_subset_2 = ",".join(str(s) for s in sorted(subset_2))

                ## make data with in conditions
                output = "{" + str_subset_1 + "}" + "\t" + "{" +
str_subset_2 + "}"
                output += "\t" + str(sup) + "\t" + str(conf) + "\n"

                of.write(output)
    of.close()
```

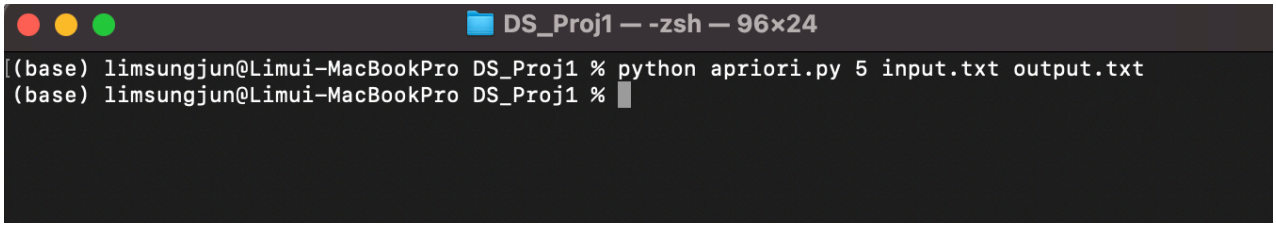
- `write_file` : 저장된 최종 frequent itemset을 output file에 저장한다. 우선 frequent의 subset을 구한다. 그리고 해당 subset에서 itemset을 선택한 다음 해당 itemset과 associative itemset의 support와 confidence를 구한다. 그리고 출력 양식에 맞게 string에 저장하고 output file에 쓴 뒤 함수를 종료한다.

```
if __name__ == "__main__":

    ## load argv
    min_sup = float(sys.argv[1])
    input_file = sys.argv[2]
    output_file = sys.argv[3]

    ## read file
    read_file(input_file)

    ## initialize frequent itemset list
    frequent = []
    frequent_sup = defaultdict(int)

    ## generate 1-candidate itemset
    C1_sup = generate_first_candidate(db)

    ## generate 1-frequent itemset
    L1, L1_sup = generate_frequent(C1_sup)

    ## k-frequent itemset
    Lk = L1
    Lk_sup = L1_sup
    k = 1

    ## generate k-candidate and k-frequent itemset
    while True:
        ## add k-frequent to frequent itemset list
        frequent.append(Lk)
        frequent_sup.update(Lk_sup)

        ## generate k-candidate from (k-1)-frequent
        Ck, Ck_sup = generate_candidate(Lk, Lk_sup, k)

        ## generate k-frequent from k-candidate
        Lk, Lk_sup = generate_frequent(Ck_sup)

        k += 1
        ## stop generating when no more k-candidate or k-frequent
        if not Lk or not Ck:
            break
```

```
    ## write file
    write_file(output_file, frequent)
```

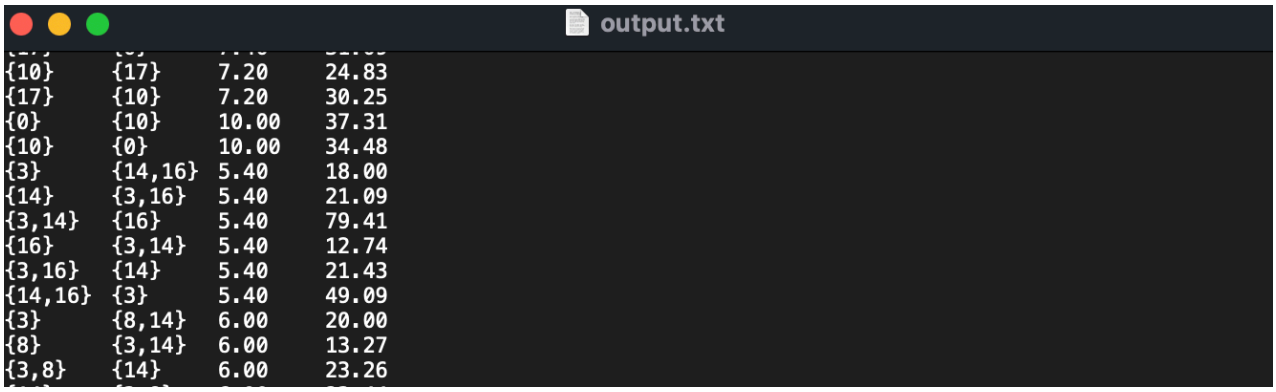
- `main` : 우선 알고리즘의 편의성을 위해 1-candidate itemset과 1-frequent itemset을 도출한다. 그리고 while문을 candidate itemset과 frequent itemset이 더이상 나오지 않을 때까지 돌며 최종 frequent list에 추가한다. 반복문이 종료되면 write_file 함수를 실행해 output file을 만든 뒤, 프로그램을 종료한다.

6. 실행 방법 및 실행 결과



- `python apriori.py 5 input.txt ouput.txt`

python으로 구현했기에 terminal 상에서는 python을 실행하는 명령어 `python` , `파일 이름`, `minimum support값`, `input file`, `output file` 순서로 입력해서 실행한다.



output.txt의 일부이다. `{item set}`, `{associative item set}`, `support`, `confidence` 가 순서대로 출력된다.