Instructions for compiling source code

Linux : python apriori.py 5 input.txt output.txt (In terminal)

Window cmd and power shell : apriori.py 5 input.txt output.txt

Summary of algorithm

데이터를 읽어드린 다음 원소가 하나인 itemset을 생성합니다. 이 때, 가능한 itemset의 반복되는 횟수는 나중에 support와 conditional probability를 계산하는데 필요하기 때문에 해쉬 태이블 구조를 가지고 있는 dictionary라는 파이썬에서 활용하는 자료구조를 가지고 가능한 원소와 그 반복되는 횟수를 저장하였습니다.

원소가 하나인 itemset을 구한 다음, 이 itemset이 기준에 맞는지 확인을 하고, 기준에 맞지 않으면 삭제합니다. apriori theorem에 따라서, 기존의 itemset을 가지고 가능한 itemset을 생성합니다. 이 때 검증과정에서 탈락한 itemset을 포함하고 있는 경우, apriori theorem에 맞지 않으므로 삭제합니다. 가능한 itemset의 반복 횟수를 dataset을 전부 조사하여 확인합니다. 조건에 맞지 않는 itemset을 제거합니다. 이 과정을 더 이상 조건에 맞는 itemset이 나오지 않을 때 까지 진행합니다.

itemset을 다 찾으면, support와 conditional probability를 계산하여 외부 파일에 저장합니다.

Detailed description of code

frequent라는 함수는 dictionary와 data를 받아와서 itemset의 반복 횟수를 측정하는 함수 입니다. 아래에서 설명드릴, generate_candidate이라는 함수에서 가능한 itemset과 그 개수를 0이라고 하여 dictionary에 넣어주면, frequent 함수를 통하여 data를 전부 훑어보면서 반복되는 횟수를 dictionary에 저장을 합니다.

반복되는 것은 set를 활용하여 구하였습니다. 특정 itemset이 데이터를 줄 단위로 읽어 왔을 때, 그 속에 포함하는지를 판단하여 있으면 1을 더합니다.

```
def frequent(data,itemset):
  # itemset is dictionary and data is 2d-array, key of dictionary is tuple
  # count the number of element in data
  for i in data:
    for j in itemset.keys():
      if set(j).issubset(set(i)):
        itemset[j] += 1
  return itemset
```

generate_candidate라는 함수는 기존의 itemset들을 받아와서 itemset 속 원소의 개수가 한 개 더 많은 itemset을 생성하는 함수입니다. 이 때, 생성을 할 때는 set을 기반으로 생성을 하고 이를 list로 변환 후, sort를 하여 key가 일정하게 유지되도록 하였다.

```
def generate_candidate(itemset):
    # using apriori theroem
    key = list(itemset.keys())
    key_set = set(key)
    new_key = dict()
    past_length = len(key[0])
    now_length = past_length + 1
    # self joining
    for i in range(len(key)):
        for j in range(len(key)):
            flag = 0
            # 선택된 itemset이 동일할 경우 종료
            if i == j:
                break
            candidate = set(key[i]).union(set(key[j]))
            candidate = list(candidate)
            candidate.sort()
            candidate_element = list()
            # pruncing
            # 만들어진 itemset에서 하나의 원소를 제거한 itemset은 기존의
            # itemset에 모두 포함되어 있어야 하므로 이를 확인
            # 만약 없다면 이를 새로운 itemset으로 하지 않음.
            for k in range(len(candidate)):
                temp = candidate.copy()
                del temp[k]
                temp = tuple(temp)
                candidate_element.append(temp)

            if not set(candidate_element).issubset(key_set):

                flag = 1

            if len(candidate) == now_length and flag == 0:
                new_key[tuple(candidate)] = 0
    return new_key
```

conditional probability를 계산하기 위해서는 itemset을 나누어야 한다. 이를 하기 위하여 subset이라는 함수를 만들어서 가능한 부분 집합을 만들어 그 부분집합과 나머지의 관계 사이에서 conditional probability를 계산하였다.

```
def subset(itemset):
    element = list(itemset)
    count = int(math.pow(2,len(itemset)) - 1)
    answer = list()
    for i in range(1,count):
        subset = list()
        binary = bin(i)
        binary = binary[2:]
        binary = binary[::-1]
        for j in range(len(binary)):
            if binary[j] == '1':
                subset.append(element[j])
```

```
            answer.append(subset)
        return answer



    ## Data 읽어오기
    RawData = list()

    while 1:
        transaction = InputData.readline().replace("\n","").split('\t')
        if transaction == [''] :
            break
        transaction = list(map(int, transaction)) ## str2int in list
        RawData.append(transaction)

    itemset = dict()

    NumberOfData = len(RawData) # The number of subjects

    for i in RawData:
        for j in i:
            if not itemset.get((j,),0):
                itemset[(j,)] = 0



    # Check frequent 1-itemset
    itemset = frequent(RawData,itemset)

    low_key = list()

    for i in itemset.keys():
        if itemset[i]/NumberOfData*100 < MinimumSupport:
            low_key.append(i)
    for i in low_key:
        del itemset[i]

    # Generate candidate itemsets of length (k+1) from frequent itemsets of length k
    new_itemset = itemset



    while True:
      # generate candidate
        new_itemset = generate_candidate(new_itemset)

      # check candidate
        new_itemset = frequent(RawData,new_itemset)

      # remove candidate

        low_key = list()

        for i in new_itemset.keys():
            if new_itemset[i]/NumberOfData*100 < MinimumSupport:
                low_key.append(i)
```

```
        for i in low_key:
            del new_itemset[i]

      # if result is NULL, stop the generate candidate

        if new_itemset == dict():
            break
        else :
            itemset.update(new_itemset)


    # calculate support and confidence

    for i in itemset.keys():
        # 가능한 부분집합을 모두 계산
        element = subset(i)
        # 각각의 부분집합의 support, conditinal probability를 계산
        for j in element:

            if len(i) == 1:
                break

            else :
                before = list(set(i).difference(j))
                before.sort()
                after = j
                support = itemset[i]/NumberOfData*100
                confidence = itemset[i]/itemset[tuple(before)]*100
                support = round(support,2)
                confidence = round(confidence,2)
                string = "{"+','.join(str(int(k)) for k in list(before))+"}\t"+"{"+','.join(str(in
                OutputData.write(string)

    OutputData.close()
    InputData.close()
```

Another specification of your implementation and testing

기존의 제공된 input.txt를 활용한 경우 AMD Ryzen 5 3550H, RAM : 8GB

- apriori.py 5 input.txt output.txt 약 0.5초
- apriori.py 2 input.txt output.txt 약 16초 Intel Core i3-7100U, RAM 8GB
- apriori.py 5 input.txt output.txt 약 1초
- apriori.py 2 input.txt output.txt 약 40초