Data Mining Lab1

1. Explain how to run your code in Step Ⅱ and Ⅲ.

我跑程式的解釋分成兩個部份:

(a). Apriori:

python3 apriori.py [-f $dataset_path]
[-s $min_support]
[-d $destination]

(b).fp_growth:

python3 fp-growth.py [-f $dataset_path]
[-s $min_support]
[-d $destination]

$dataset_path -> The path of Dataset
$min_support -> The Threshold of frequency itemset
$destination -> The place to store result
我把跑程式的example放在Apriori.sh跟fp-growth.sh

2.Step 2

以下為Step 2的報告:

(a).Report on the mining algorithms/codes:

--> The modifications you made for Task 1 and Task 2 (Task 1):

- 我將原本程式輸入格式為.csv轉換成.data
- 我增加了一些資料後處理，讓輸出結果符合要求

--> The modifications you made for Task 1 and Task 2 (Task 2):

- 我增加了erase_not_closed_function：
  此函數會把前一個大小的結果(k=n-1)跟目前的超集(k=n)比較，如果兩者的support一樣就會把前一個大小的結果刪掉
- 我增加了一些資料後處理，讓輸出結果符合要求

--> The restrictions

- Scalability：
  將大小增加的情況下，處理資料的時間也會拉長非常多，如:
  ntrans=1000 -> 8s
  ntrans=1000000 -> 2hr

--> Problems encountered in mining:

- Computational resource:
  這次作業需要處理非常大的資料，光靠筆電的cpu運算會比較吃力。
- Debug問題:
  這次作業要debug比較難，除了很難知道自己的結果有沒有錯以外，使用人工的方式去看原始dataset也需要一點時間。

--> Any observations/discoveries you want to share

- None

(b).Paste the screenshot of the computation time.For Task 2, you also need to show the ratio of computation time compared to that of Task 1 in your report.

--> A Dataset

- mininal support = 0.002

```
start to do task1
運行時間： 240.1739287
finish task1
start to do task2
運行時間： 230.432787
finish task2
```

230.43/240.17 = 0.96

- minimal support =0.005

```
start to do task1
運行時間： 22.52863359451294
finish task1
start to do task2
運行時間： 23.454061269760132
finish task2
```

22.53/23.45 = 96%

- minimal support =0.01

```
start to do task1
運行時間： 8.805143356323242
finish task1
start to do task2
運行時間： 8.748783588409424
finish task2
```

8.7/8.8 = 98.9%

--> B Dataset

- mininal support = 0.0015

```
start to do task1

運行時間： 13878.631932973862
finish task1
start to do task2
運行時間： 13933.12871336937
finish task2
```

13933.13 / 13878.63 = 100.3%

- mininal support = 0.002

```
start to do task1
運行時間： 365.58728218078613
finish task1
start to do task2
運行時間： 364.8671908378601
finish task2
```

365.87/365.59 = 99.8%

- mininal support = 0.005

```
start to do task1
運行時間： 127.58261442184448
finish task1
start to do task2
運行時間： 128.19715881347656
finish task2
```

128.2 / 127..58 = 100.4 %

--> C Dataset

- mininal support = 0.01

```
start to do task1


運行時間： 11729.569254636765
finish task1
start to do task2
運行時間： 11404.431406259537
finish task2
```

- mininal support = 0.02

```
start to do task1
運行時間： 3597.3553338050842
finish task1
start to do task2
運行時間： 3622.5421328544617
finish task2
```

- mininal support = 0.03

```
start to do task1
運行時間： 1300.0738904476166
finish task1
start to do task2
運行時間： 1281.563426733017
finish task2
```

(c).Paste the screenshot of your code modification for Task 1 and Task 2 with comments and explain it.

--> You need to explain it in clear and well-structured ways.(Task1)

- 把Input的輸入格式從.csv改成.data格式

```python
def dataFromFile(fname):
    """Function which reads from the file and yields a generator"""
    with open(fname, "r") as file_iter:
        for line in file_iter:
            tmp = line
            line = line.strip().split(" ")
            line = line[3:]
            record = frozenset(line)
            yield record
```

從原本用 "," 來split，改成用" "隔開。
前面兩格是TID資訊，所以從第三格開始。

- 要記錄每次candidate跟最後留下來的數量

```python
    index = 1
    oneCSet= returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)
    s = (f'{index}\t{len(itemSet)}\t{len(oneCSet)}\n')
    index += 1
    currentLSet = oneCSet
    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet= returnItemsWithMinSupport(
            currentLSet, transactionList, minSupport, freqSet
        )
        s += (f'{index}\t{len(currentLSet)}\t{len(currentCSet)}\n')
        index += 1
        currentLSet = currentCSet
        k = k + 1
```

s 是負責紀錄每次**candidate**跟最後留下來的數量的數據，其餘這部份沒有什麼更動

- 紀錄每次的compuation time

```python
print("start to do task1")
start = time.time()
items,s = runApriori(inFile, minSupport)
write_file1(items,f'{args.destination}/step2_task1_{args.inputFile.split(".")[0]}_{args.minSupport}_result1.txt')
write_file2(s,items,f'{args.destination}/step2_task1_{args.inputFile.split(".")[0]}_{args.minSupport}_result2.txt')
end = time.time()
print('運行時間：', str(end - start))
print("finish task1")
```

而write_file1跟write_file2就是根據題目要求將輸出按照格式寫出

write_file1:

```python
def write_file1(oneCSet,output_path):
    with open(output_path,'w') as f:
        for item, support in sorted(oneCSet, key=lambda x: x[1],reverse=True):
            a = []
            for i in item:
                a.append(i)
            f.write(f'{round(support*100,1)}   {set(a)}\n')
```

write_file2:

```python
def write_file2(s,oneCSet,output_path):
    with open(output_path,'w') as f:
        f.write(f'{len(oneCSet)}\n')
        f.write(s)
```

--> You need to explain it in clear and well-structured ways.(Task2)

- 把Input的輸入格式從.csv改成.data格式

```python
def dataFromFile(fname):
    """Function which reads from the file and yields a generator"""
    with open(fname, "r") as file_iter:
        for line in file_iter:
            tmp = line
            line = line.strip().split(" ")
            line = line[3:]
            record = frozenset(line)
            yield record
```

從原本用 "," 來split，改成用" "隔開。
前面兩格是TID資訊，所以從第三格開始。

- 增加了erase_non_closed的用法

```python
itemSet, transactionList = getItemSetTransactionList(data_iter)
freqSet = defaultdict(int)
largeSet = dict()
# Global dictionary which stores (key=n-itemSets,value=support)
# which satisfy minSupport
oneCSet= returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)
currentLSet = oneCSet
k = 2
while currentLSet != set([]):
    largeSet[k - 1] = currentLSet
    currentLSet = joinSet(currentLSet, k)
    currentCSet= returnItemsWithMinSupport(
        currentLSet, transactionList, minSupport, freqSet
    )
    largeSet[k-1] = erase_non_closed(largeSet[k-1],currentCSet,freqSet,transactionList)
    currentLSet = currentCSet
    k = k + 1
```

比較大小為K-1的itemset在大小K的itemset有沒有交集，如果有比較兩者的support，如果support相同，則刪除k-1的itemset。

erase_non_closed:

```python
def erase_non_closed(largeSet,currentCSet,freqSet,transactionList):
    if(len(currentCSet)==0):
        return largeSet
    def getSupport(item):
        """local function which Returns the support of an item"""
        return float(freqSet[item]) / len(transactionList)
    ans = set()
    anti_ans = set()
    for i in largeSet:
        sign = True
        for j in currentCSet:
            if i.issubset(j) and getSupport(i)==getSupport(j):
                sign = False
                break
        if(sign):
            ans.add(i)
    return ans
```

- 紀錄每次的compuation time

```python
print("start to do task2")
start = time.time()
result = runApriori3(inFile, minSupport)
write_file3(result,f'{args.destination}/step2_task2_{args.inputFile.split(".")[0]}_{args.minSupport}_result1.txt')
end = time.time()
print('運行時間：', str(end - start))
print("finish task2")
```

而write_file3就是根據題目要求將輸出按照格式寫出

write_file3:

```
def write_file3(oneCSet,output_path):
    with open(output_path,'w') as f:
        f.write(f"{len(oneCSet)}\n")
        for item, support in sorted(oneCSet, key=lambda x: x[1],reverse=True):
            a = []
            for i in item:
                a.append(i)
            f.write(f'{round(support*100,1)}    {set(a)}\n')
```

3.Step 3

(a).Descriptions of your mining algorithm

--> Relevant references:

--> Program flow

1. 首先會先統計k=1的itemset，並將小於threshold的itemset刪掉

```
for transaction in transactions:
    for item in transaction:
        items[item] += 1
```

2. 製作TF-tree

```
master = FPTree()
for transaction in map(clean_transaction, transactions):
    master.add(transaction)
```

3. 找每個保留的itemset的suffix

```
def find_with_suffix(tree, suffix):
    for item, nodes in tree.items():
        support = sum(n.count for n in nodes)
        if support >= minimum_support and item not in suffix:

            found_set = [item] + suffix
            yield (found_set, support/len(transactions)) if include_support else found_set

            cond_tree = conditional_tree_from_paths(tree.prefix_paths(item))
            for s in find_with_suffix(cond_tree, found_set):
                yield s

for itemset in find_with_suffix(master, []):
    yield itemset
```

--> Anything you want to share

None

(b).Differences/Improvements in your algorithm

-->Explain the main differences/improvements of your algorithm compared to Apriori.

因為Aprior都要把candidates去比對整個dataset，所以會需要大量的時間做data mining，而TF-growth演算法透過建立TF-tree，只需要掃描整個dataset兩次，加對TF-tree做遞迴（可用dynamic programming加速）就可以得到答案。

--> You should describe the modifications your made on the original algorithm.

- Change the input format

```
transactions = []
with open(path,'r') as f:
    for line in f:
        tmp = line.strip().split(" ")[3:]
        transactions.append(tmp)

items = defaultdict(lambda: 0)

for transaction in transactions:
    for item in transaction:
        items[item] += 1
minimum_support = ratio_support * len(transactions)
```

因為前面兩個column是TID是不重要的資訊，所以是從第三項開始取資料。

- Change the output format

```
result = sorted(result, key=lambda i: i[1],reverse=True)   # 排序后输出
output_path = f'{args.destination}/{args.minSupport}_fp_growth.txt'
with open(output_path,'w') as f:
    for itemset, support in result:
        a = []
        for i in itemset:
            a.append(int(i))
        f.write(f'{round(support*100,1)}\t{set(a)}\n')
```

我將output變成指定的格式。

(c).Computation time

	0.002/ 0.0015/ 0.01	0.005/ 0.002/ 0.02	0.01/ 0.005/ 0.03
A	98.2 %	95.6 %	91.7 %
B	99.16 %	75.14%	46.53%
C	95.36 %	92.80 %	89.40 %

(d).Discuss the scalability of your algorithm in terms of the size of dataset

- when n = 1000:
  computation time: 4.237065553665161
- when n = 100000:
  computation time: 90.89950513839722

從上述數據來看上升100倍的資料量，大約上升23倍的運算時間