Data Mining HW1 Report 多工所碩一 313553024 蘇柏叡

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

1.1 Introduction to the OptionParser in Command Line Interfaces

和原先spec不一樣的部分是**如同圖一**，-p原先是output_path，不過我在這邊抽換成step也算是為了在輸出檔案時和**Apriori(step2)**、**Our implementations(step3)**做出區別。以下為各command的介紹:

-f: input datasets 包含了datasetA.data、datasetB.data、datasetC.data

-s: 為minimum support value，若為0.3%則為0.003、0.6%則為0.006，依此類推。

-p: 為區分**step2**、**step3**，在**Step2.py**中則是default設為**2**，若在Step3.py則是設3。

```
if __name__ == "__main__":
    optparser = OptionParser()
    optparser.add_option("-f", "--inputFile", dest="input", help="Filename containing the dataset", default='datasetA.data')
    optparser.add_option("-s", "--minSupport", dest="minS", help="Minimum support value", default=0.1, type="float")
    optparser.add_option("-p", "--step", dest="step", help="Step number (2 or 3)", default='2')

    (options, args) = optparser.parse_args()
```

圖1為Command Line 中OptionParser的參數

1.2 Name of output File

由於spec有規定格式，因此我選擇用圖二去生成檔案名稱，以step數字、dataset名稱、最小支持度並且將task事先分好(1or2) 以及result1、result2寫好。至於生成後的檔名如圖三所示。

```
182     task1_itemsets_file = f"step{step}_task1_{dataset_name}_{min_support}_result1.txt"
183     task1_stats_file = f"step{step}_task1_{dataset_name}_{min_support}_result2.txt"
184     task2_closed_itemsets_file = f"step{step}_task2_{dataset_name}_{min_support}_result1.txt"
```

圖2 生成檔名

step2_task1_datasetB_0.006_result1	2024/10/29 上午 01:27	文字文件	7 KB
step2_task1_datasetB_0.006_result2	2024/10/29 上午 01:27	文字文件	1 KB
step2_task1_datasetC_0.005_result1	2024/10/29 上午 03:06	文字文件	10 KB
step2_task1_datasetC_0.005_result2	2024/10/29 上午 03:06	文字文件	1 KB
step2_task1_datasetC_0.010_result1	2024/10/29 上午 02:26	文字文件	4 KB
step2_task1_datasetC_0.010_result2	2024/10/29 上午 02:26	文字文件	1 KB
step2_task1_datasetC_0.015_result1	2024/10/29 上午 01:47	文字文件	3 KB
step2_task1_datasetC_0.015_result2	2024/10/29 上午 01:47	文字文件	1 KB
step2_task2_datasetA_0.003_result1	2024/10/28 下午 10:15	文字文件	77 KB
step2_task2_datasetA_0.006_result1	2024/10/28 下午 10:38	文字文件	12 KB

圖3 生成後之檔名截圖(節錄)

1.3 Command Line

(1) Step2.py部分(含A、B、C dataset)->此部分會生成task1的 result1(frequent itemset)、result2(stats)、task2的result1

python Step2.py -f datasetA.data -s 0.003 -p 2
python Step2.py -f datasetA.data -s 0.006 -p 2
python Step2.py -f datasetA.data -s 0.009 -p 2

python Step2.py -f datasetB.data -s 0.002 -p 2
python Step2.py -f datasetB.data -s 0.004 -p 2
python Step2.py -f datasetB.data -s 0.006 -p 2

python Step2.py -f datasetC.data -s 0.005 -p 2
python Step2.py -f datasetC.data -s 0.010 -p 2
python Step2.py -f datasetC.data -s 0.015 -p 2

(2) Step3.py部分(含A、B、C dataset)->此部分僅會生成result1(因應spec、E3公告)

python Step3.py -f datasetA.data -s 0.003 -p 3
python Step3.py -f datasetA.data -s 0.006 -p 3
python Step3.py -f datasetA.data -s 0.009 -p 3

python Step3.py -f datasetB.data -s 0.002 -p 3
python Step3.py -f datasetB.data -s 0.004 -p 3
python Step3.py -f datasetB.data -s 0.006 -p 3

python Step3.py -f datasetC.data -s 0.005 -p 3
python Step3.py -f datasetC.data -s 0.010 -p 3
python Step3.py -f datasetC.data -s 0.015 -p 3

註記: 生成後的檔名support後綴皆是取至小數點後第三位

2. Step II

2.1 Report on the mining algorithms/codes:

2.1.1 runApriori function

首先調整的是runApriori function，於**85行**處增加stats字典用於追蹤每次迭代的itemset數量變化，包括呈現剪枝前後的數量以及total frequent itemset數量(**100**、**101行**)，其中pruning的標準即是判斷是否有超過minimum support。在每次迭代過程中，**stats**會更新剪枝前後的total frequent itemset的數量(**107行**)，當所有迭代完成後，stats會新增一個total_frequent_itemsets，其值為從所有階段中累計的頻繁項集總數(交由後續生成統計txt檔案時放置於第一行)。最後runApriori會多回傳stats、freqSet以用於後續寫入txt檔案。

```
73  def runApriori(data_iter, minSupport):
74      """
75      run the apriori algorithm. data_iter is a record iterator
76      Return both:
77       - items (tuple, support)
78      """
79      itemSet, transactionList = getItemSetTransactionList(data_iter)
80
81      freqSet = defaultdict(int)
82      largeSet = dict()
83      # Global dictionary which stores (key=n-itemSets,value=support)
84      # which satisfy minSupport
85      stats = {'iterations': {}, 'total_frequent_itemsets': 0}
86
87      oneCSet = returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)
88
89      currentLSet = oneCSet
90      k = 2
91      iteration_count = 1
```

```
93      while currentLSet != set([]):
94          largeSet[k - 1] = currentLSet
95          before_pruning = len(currentLSet)
96
97          currentLSet = joinSet(currentLSet, k)
98          currentCSet = returnItemsWithMinSupport(currentLSet, transactionList, minSupport, freqSet)
99
100         after_pruning = len(currentCSet)
101         stats['iterations'][iteration_count] = {'before_pruning': before_pruning, 'after_pruning': after_pruning}
102
103         currentLSet = currentCSet
104         k = k + 1
105         iteration_count += 1
106
107     stats['total_frequent_itemsets'] = sum(len(v) for v in largeSet.values())
108
```

```
109     def getSupport(item):
110         """local function which Returns the support of an item"""
111         return float(freqSet[item]) / len(transactionList)
112 
113     toRetItems = []
114     for key, value in largeSet.items():
115         toRetItems.extend([(tuple(item), getSupport(item)) for item in value])
116 
117     return toRetItems, stats, freqSet
```

圖4~圖6為runApriori function

2.1.2寫入txt檔案之file(此部分不包含closed_frequent_itemset)

首先將原先printResults(items)、to_str_results(items)註解掉(圖七)，因為接下來要撰寫的是要寫入txt檔案的。接下來要介紹的是**write_itemsets_to_file(itemsets, filename)**，函數內容為利用**sorted(itemsets, key=lambda x: -x[1])**針對support由高到低排序，接著介紹的是**write_statistics_to_file(stats, filename)**，函數內容首先先打印出total_frequent_itemset的數量，並且每次迭代時都會把剪枝前、剪枝後的數量打印出來，而在runApriori中可以看出iteration若是當沒有任何itemset在剪枝前超過support時，就不會再繼續。

```
118 '''
119 ###revise!!!###
120 
121 def printResults(items):
122     """prints the generated itemsets sorted by support """
123     for item, support in sorted(items, key=lambda x: x[1]):
124         print("item: %s , %.3f" % (str(item), support))
125 
126 
127 def to_str_results(items):
128     """prints the generated itemsets sorted by support"""
129     i = []
130     for item, support in sorted(items, key=lambda x: x[1]):
131         x = "item: %s , %.3f" % (str(item), support)
132         i.append(x)
133     return i
134 
135 '''
```

圖7 註解掉原先用來呈現itemset、support的function

```python
def write_itemsets_to_file(itemsets, filename):
    with open(filename, 'w') as f:
        for item, support in sorted(itemsets, key=lambda x: -x[1]):
            f.write(f"{support*100:.1f}\t{{{', '.join(map(str, item))}}}\n")

def write_statistics_to_file(stats, filename):
    with open(filename, 'w') as f:
        f.write(f"{stats['total_frequent_itemsets']}\n")
        for iteration, data in stats['iterations'].items():
            f.write(f"{iteration}    {data['before_pruning']}\t{data['after_pruning']}\n")
```

圖8 write_itemsets_to_file(itemsets, filename)、write_statistics_to_file(stats, filename)

2.1.3 Task2 closed frequent itemset&寫入文檔函數

is_closed_itemset function負責檢查一個項集是否為封閉頻繁項集。它通過對比當前itemset與所有其他較大**superset**的支持度來實現。若是發現存在一個superset其支持度和當前itemset是相同的，則該itemset就不會是close itemset而被本函數濾除，反之**若是回傳為True則是同義於被認定為close itemset**。接著我們繼續撰寫了**findClosedItemsets function**，**並且調用剛完成之is_closed_itemset function**來篩選出所有封閉的頻繁項集。在遍歷每個itemsets之後並檢查其是否封閉，若符合封閉之條件則將其加入到list中，最後並回傳**closed_itemsets**並且打印數量。此外，於**write_closed_itemsets_to_file** 寫入文檔時依然是**按照support由大到小排序**下來，並且在文檔頂端也是要寫入**closed_itemsets的數量**。

```python
def is_closed_itemset(item, itemsets, freqSet):
    for larger_item in itemsets:
        if item < larger_item and freqSet[item] == freqSet[larger_item]:
            return False
    return True

def findClosedItemsets(itemsets, freqSet):
    closed_itemsets = []
    for item in itemsets:
        if is_closed_itemset(item, itemsets, freqSet):
            closed_itemsets.append(item)
    print(f"Number of closed itemsets found: {len(closed_itemsets)}")
    return closed_itemsets
```

```python
def write_closed_itemsets_to_file(closed_itemsets, filename, freqSet, transactionList_length):
    with open(filename, 'w') as f:
        f.write(f"{len(closed_itemsets)}\n")
        for item in sorted(closed_itemsets, key=lambda x: -freqSet[frozenset(x)]):
            support = (freqSet[frozenset(item)] / transactionList_length) * 100
            f.write(f"{support:.1f}\t{{{', '.join(map(str, item))}}}\n")
```

圖9、10包含 is_closed_itemset function、findClosedItemsets function、write_closed_itemsets_to_file 三個function

2.1.4 綜合更改與添加(含資料集、計算時間)

由於我們生成的資料是空格分開，因此要將原始碼更改成是空格分開並且只取第三位數字以後的數值，因為前兩位為序號，第三位數則是該行有多少筆交易紀錄，因此沒有必要保存。至於在計算task1、task2兩者花費時間部分，我們將需要產生itemset、統計文字檔部分皆是算在task1身上，換言之task1負責之部分則是執行runApriori、寫入itemsets、統計文檔。至於，考量因為對於spec中的independent 這個字詞並不是特別明白(筆者想到有兩種可能，其一為**時間獨立也就是task2僅是負責利用task1生成之結果進行查找何者為closed itemset以及寫入文檔，另一則是兩者必須分開獨立運算。雖筆者認為前者較為合理畢竟每次執行結果均不同若是使用後者來做恐有誤差**)，不過最終筆者選擇於終端機輸出兩種方式之比率，211行為僅計算執行尋找closed itemsets加上寫入文檔時間作為task2再除以task1執行時間，212行其實打印出來的也就是整個overall的時間/task1。

```
154  def dataFromFile(fname):
155      """Function which reads from the file and yields a generator"""
156      with open(fname, "r") as file_iter:
157          for line in file_iter:
158              line = line.strip().rstrip(",")  # Remove trailing comma
159              tokens = line.split()
160              record = frozenset(tokens[3:])  #cuz the first three nums will not be included.
161              yield record
```

圖11 為導入資料之函數(僅需保留第三位數字以後的數值即可)

```
188  # Task 1: Frequent itemset mining
189  start_time_task1 = time.perf_counter()
190  itemsets, stats, freqSet = runApriori(inFile, minSupport)
191  write_itemsets_to_file(itemsets, task1_itemsets_file)
192  write_statistics_to_file(stats, task1_stats_file)
193  end_time_task1 = time.perf_counter()
194  task1_time = end_time_task1 - start_time_task1
195
196  # Task 2: Closed itemset filtering
197  start_time_task2 = time.perf_counter()
198  closed_itemsets = findClosedItemsets([frozenset(i[0]) for i in itemsets], freqSet)
199  write_closed_itemsets_to_file(closed_itemsets, task2_closed_itemsets_file, freqSet, len(inFile))
200  end_time_task2 = time.perf_counter()
201  task2_time = end_time_task2 - start_time_task2
202
203  total_time = task1_time + task2_time
204  ratio = (task2_time / task1_time) * 100
205  ovr_ratio = (total_time / task1_time) * 100
206  # Display results
207  print(f"Task 1 computation time in Apriori: {task1_time:.6f} seconds")
208  print(f"Task 2 (Only Searching closed Itemset) computation time in Apriori: {task2_time:.6f} seconds")
209  print(f"Total computation time in Apriori: {total_time:.6f} seconds\n")
210  print("-------------------------------------Time Ratio will be shown below-------------------------------------\n")
211  print(f"[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: {ratio:.6f}%")
212  print(f"[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: {ovr_ratio:.6f}%")
```

圖12 為計算task1、task2花費時間之計算方式

2.2 Computation time in Apriori (with screenshot and some observations):

2.2.1 datasetA (Set number of transactions = 1,000, number of items = 600)

(1) set min_support = 0.3%

由圖十三可知Task2/Task1的比率約為9.645%。換言之，整個流程所花費的時間之於Task1則是約為109.645%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.003 -p 2
Number of closed itemsets found: 5033
Task 1 computation time in Apriori: 15.818278 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 1.525664 seconds
Total computation time in Apriori: 17.343942 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 9.644944%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 109.644944%
```

圖13 為datasetA使用min_support = 0.3%的執行時間與比率

(2) set min_support = 0.6%

由圖十四可看出Task2/Task1的比率約為0.502%。換言之，整個流程所花費的時間之於Task1則是約為100.502%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.006 -p 2
Number of closed itemsets found: 856
Task 1 computation time in Apriori: 3.290804 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.016505 seconds
Total computation time in Apriori: 3.307308 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.501537%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.501537%
```

圖14 為datasetA使用min_support = 0.6%的執行時間與比率

(3) set min_support = 0.9%

由圖十五可看出Task2/Task1的比率約為0.191%。換言之，整個流程所花費的時間之於Task1則是約為100.191%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.009 -p 2
Number of closed itemsets found: 407
Task 1 computation time in Apriori: 2.222030 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.004240 seconds
Total computation time in Apriori: 2.226270 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.190803%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.190803%
```

圖15 為datasetA使用min_support = 0.9%的執行時間與比率

2.2.2 datasetB (Set number of transactions = 100,000, number of items = 500)

(1) set min_support = 0.2%

由圖十六可知Task2/Task1的比率約為0.0147%。換言之，整個流程所花費的時間之於Task1則是約為100.0147%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.002 -p 2
Number of closed itemsets found: 5911
Task 1 computation time in Apriori: 4918.468521 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.723137 seconds
Total computation time in Apriori: 4919.191658 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.014702%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.014702%
```

圖16 為datasetB使用min_support = 0.2%的執行時間與比率

(2) set min_support = 0.4%

由圖十七可知Task2/Task1的比率約為0.0063%。換言之，整個流程所花費的時間之於Task1則是約為100.0063%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.004 -p 2
Number of closed itemsets found: 1273
Task 1 computation time in Apriori: 548.142202 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.034530 seconds
Total computation time in Apriori: 548.176732 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.006299%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.006299%
```

圖17 為datasetB使用min_support = 0.4%的執行時間與比率

(3) set min_support = 0.6%

由圖十八可知Task2/Task1的比率約為0.0017%。換言之，整個流程所花費的時間之於Task1則是約為100.0017%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.006 -p 2
Number of closed itemsets found: 545
Task 1 computation time in Apriori: 396.062270 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.006865 seconds
Total computation time in Apriori: 396.069135 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.001733%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.001733%
```

圖18 為datasetB使用min_support = 0.6%的執行時間與比率

2.2.3 datasetC (Set number of transactions = 500,000, number of items = 500)

(1) set min_support = 0.5%

由圖十九可知Task2/Task1的比率約為0.00058%。換言之，整個流程所花費的時間之於Task1則是約為100.00058%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.005 -p 2
Number of closed itemsets found: 753
Task 1 computation time in Apriori: 2259.422121 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.013117 seconds
Total computation time in Apriori: 2259.435237 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000581%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000581%
```

圖19 為datasetC使用min_support = 0.5%的執行時間與比率

(2) set min_support = 1%

由圖二十可知Task2/Task1的比率約為0.00021%。換言之，整個流程所花費的時間之於Task1則是約為100.00021%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.010 -p 2
Number of closed itemsets found: 298
Task 1 computation time in Apriori: 1214.357689 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.002537 seconds
Total computation time in Apriori: 1214.360226 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000209%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000209%
```

圖20 為datasetC使用min_support = 1.0%的執行時間與比率

(3) set min_support = 1.5%

由圖二十一可知Task2/Task1的比率約為0.00024%。換言之，整個流程所花費的時間之於Task1則是約為100.00024%

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.015 -p 2
Number of closed itemsets found: 241
Task 1 computation time in Apriori: 823.316026 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.001983 seconds
Total computation time in Apriori: 823.318008 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000241%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000241%
```

圖21 為datasetC使用min_support = 1.0%的執行時間與比率

2.2.4 Some Restrictions & Problems & Observations :

(1) 資料規格與時間關係

首先針對資料的量度，A、B、C分別為1K、100K、500K筆資料，因此不論是使用何種演算法，可以預期到的是尋找frequent Itemsets的花費時間必定會有大幅度的增加。然而，**透過額外實驗(三個資料集皆設最小支持度為0.5%如圖二十二至二十四)**我們可發現，隨著資料的規格增加，耗費時間的幅度成長其實也**蠻接近線性倍數成長**，不過考量三種資料集生成的分布無法相同甚至是相近都很難(例如:符合最小支持度的Itemsets的個數幾乎不可能是成比例。)

因此在**看似花費時間由(4.57s->461.80s->2259.44s，恰接近100倍、500倍時間)為近似線性倍數成長的同時**，我們也需要注意的是**最小支持度在不同規格大小的資料中會對資料帶來何種影響。**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.005 -p 2
Number of closed itemsets found: 1391
Task 1 computation time in Apriori: 4.527801 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.042175 seconds
Total computation time in Apriori: 4.569976 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.931468%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.931468%
```

圖22 為datasetA使用min_support = 0.5%的執行時間與比率

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.005 -p 2
Number of closed itemsets found: 777
Task 1 computation time in Apriori: 461.786816 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.014433 seconds
Total computation time in Apriori: 461.801249 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.003126%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.003126%
```

圖23 為datasetB使用min_support = 0.5%的執行時間與比率

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.005 -p 2
Number of closed itemsets found: 753
Task 1 computation time in Apriori: 2259.422121 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.013117 seconds
Total computation time in Apriori: 2259.435237 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000581%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000581%
```

圖24 為datasetC使用min_support = 0.5%的執行時間與比率

(2) Task2/Task1 time ratio與不同min support之關聯性

由2.2.1至2.2.3中我們可看出當closed_itemsets有數千筆時，Task2的時間比率會大幅升高，然根據結果，我們發現當使用較大的資料集時，Task2的運算時間比率大幅降低。為探究此現象，筆者將spec要求針對每種資料集設定三種min support並進行mining所得之frequent itemsets以及closed_itemsets數量做統計如表一至表三。

其中我們可看出在**規格較小時(如datasetA的1K筆資料中)**若是設定0.3、0.6%較小的最小支持度，可能會造成Frequent Itemsets的數量甚至是遠高過於原始data筆數的情形。而結果也就可想而知，**其中必定存在一些非封閉的Itemsets**。至於datasetB、C具有較大規格的交易資料，因為分布較不易有嚴重偏態，因此設定較低的支持度時，比較不會如同較小規模的資料集產生如此高比例的Frequent Itemsets，但又有許多非封閉的Itemsets。最後，由統計表一至表三可發現由於表二、表三的Closed Itemsets均等於Frequent Itemsets數量，也就是說在進行判斷時不會花較多的時間查找其supersets並且刪除不符條件之Itemsets，故datasetB、C的Task2時間也就因而縮短許多。

datasetA \ min_sup	0.3%	0.6%	0.9%
Frequent Itemsets	8631	864	407
Closed Itemsets	5033	856	407

表一 為datasetA 於不同最小支持度時擁有的Frequent Itemsets、Closed Itemsets數量

datasetB \ min_sup	0.2%	0.4%	0.6%
Frequent Itemsets	5911	1273	545
Closed Itemsets	5911	1273	545

表二 為datasetB 於不同最小支持度時擁有的Frequent Itemsets、Closed Itemsets數量

datasetC \ min_sup	0.5%	1.0%	1.5%
Frequent Itemsets	753	298	241
Closed Itemsets	753	298	241

表三 為datasetC 於不同最小支持度時擁有的Frequent Itemsets、Closed Itemsets數量

(3) Apriori Algorithm本身侷限性以及其他Mining時面對到的問題

根據定義，**Apriori Algorithm**是**採用Bottom-Up的廣度優先搜尋方式**來尋找與**遍歷所有可能的子集合(2^k-1個)**並從中尋找符合最小支持度的frequent Itemsets。然而，當如同(1)、(2)兩點中提到的datasetB、datasetC這種scale較大的資料集時，此演算法就容易產生效率較為不佳的情形。

此外，**Apriori Algorithm為candidate-based演算法**，因此在每次生成候選集時都必須掃描整個資料庫來計算支持度，若是選用的最小支持度較小時，候選集的Itemsets數量將會大幅成長，雖未必定是指數型成長，但可想而知必定會是大幅降低運算效率。

由於，參考多方資訊皆指出**Apriori Algorithm**在實作上耗費較多時間的原因為產生大量候選集並挑選符合最小支持度的**frequent Itemsets**，因此在Step III時，筆者將**嘗試使用Non-Candidate-based的演算法**試圖比較和原先使用Apriori演算法在執行效率上之差異。

3. Step III

3.1 Descriptions of my mining algorithm

簡單而言，本階段將實現**FP-Growth Algorithm 以Non-Candidate Based的方式**針對上階段**使用Apriori Algorithm**進行運行時間與效能之評比。有別於Apriori 需要多次掃描數據集並生成大量的候選項集，運行效率往往會隨著資料集規模增長呈指數級下降。**FP-Growth Algorithm**則是先建構FP-Tree並且以**recursive mining Frequent Itemsets**之方式**以將重複掃描數據集的次數降至最低從而達到提升運行效率。**至於，本次實現的**FP-Growth Algorithm**的**program Flow的步驟則是如下**(因是**Non-Candidate Based，故不必輸出result2.txt file**)

- Program Flow:

(1)利用**1. 3**提供之**command line**選擇想要mining的dataset，並且設定最小支持度。

(2)利用**dataFromFile()**讀取交易資料(如同Step2.py般過濾掉前三個數字)。

(3)執行**runFPGrowth()**並且傳入**transactions**、**min_support**之後使用該演算法進行mining Frequent Itemsets。

(4)透過write_itemsets_to_file()將獲得之**Frequent Itemsets寫入txt file**並計算時間。

至於第三步驟則是本階段最為核心之部分，**則是參考[1]、[2]建構FP-Tree及實現FP-Growth Algorithm相關部分**，不過為了符合本Lab之格式與要求，則是先拿**Apriori source code**挖空執行Apriori Algorithm部分並且改採用參考[1]、[2]撰寫之**FP-Growth**再嘗試進行整合，最終完成本階段，不過考量若是僅針對修改部分進行介紹恐有失真且解釋不祥之問題，**因此筆者依然會將全部程式碼撰寫邏輯於下一章進行詳細說明。**

3.2 Implement Details and Differences/Improvements in my algorithm

3.2.1 Implement Details

圖二十五部分和Step2.py相同皆是先針對資料進行處理並且過濾掉前三位數字。

```python
import sys
import time
from collections import defaultdict
from optparse import OptionParser

def dataFromFile(fname):
    """Read transaction data from file and return a generator of ordered items."""
    with open(fname, "r") as file_iter:
        for line in file_iter:
            line = line.strip().rstrip(",")
            tokens = line.split()[3:]  # Assumes the transaction starts from the 4th element
            yield tokens  # Return list to keep order
```

圖25 為dataFromFile函數以及引入相關套件

接著於14-30行建立一個名為FPTree的類別，並且先進行初始化FP-Tree。首先root為根節點(字典形式)，包含了1. name(表示節點的名稱)、2. count(該節點代表的項目的出現次數)、3. parent(指向該節點的父節點)、4. children包含該節點的所有子節點，並且以字典形式存儲，此字典的**key**、**value**分別為**子節點的名稱**及**子節點本身**。使用**headers**可存每個項目的節點列表，可方便後續查找**frequent Itemsets**。

至於**add_transaction function**會遍歷交易中的每個項目，並逐步在FP-Tree中構建對應的節點。對於每個項目，首先檢查**當前節點的children是否已存在相應的項目**。

case(1): 若該項目已**存在**於子節點中，則更新該子節點的count，**並增加該項目在當前交易中的出現次數。**

case(2): 若該項目**不存在**於子節點中，則**創建一個新的節點**，並將其加入到當前節點的 **children** 中，同時更新 **headers**，將新節點加入到對應項目的節點鏈表中。

```
14  class FPTree:
15      def __init__(self):
16          """Initialize FP-Tree"""
17          self.root = {'name': None, 'count': None, 'parent': None, 'children': {}}
18          self.headers = defaultdict(list)
19
20      def add_transaction(self, transaction, count=1):
21          """Add a transaction to the FP-Tree preserving the original order."""
22          current_node = self.root
23          for item in transaction:  # transaction is a list preserving the order
24              if item in current_node['children']:
25                  current_node['children'][item]['count'] += count
26              else:
27                  new_node = {'name': item, 'count': count, 'parent': current_node, 'children': {}}
28                  current_node['children'][item] = new_node
29                  self.headers[item].append(new_node)
30              current_node = current_node['children'][item]
```

圖26為FP Tree類別以及其initial、add_transaction function

接著於32-40行撰寫**mine_patterns**函數，首先去遍歷**headers**中所有的項目，並且可查找到該項目對應之節點，接著透過**相加count**可以得到該項目在所有交易中的總出現次數並且計算支持度**並判斷是否為frequent Itemsets**。當某個項目被確定為頻繁項集後，**mine_patterns()**會進一步調用 **_mine_conditional_tree()** 方法，並根據這棵樹繼續挖掘更大的**frequent patterns**。**至於_mine_conditional_tree()**的主要功能是構建**conditional FP-Tree**，並基於這棵樹繼續挖掘更大的**frequent patterns**。首先，會遍歷

headers中找到所有包含該項目的節點，然後往上找每個節點的父節點以生成**conditional pattern base**，而這些**conditional pattern base**會被視為新的交易，並被添加到條件**FP-Tree**中，從而構建出以該項目為基礎的子樹。

當條件 **FP-Tree** 構建完成後，**_mine_conditional_tree()**會再次調用**mine_patterns()**，以**recursive**的方式在條件樹中挖掘**frequent patterns(**此為**FP Growth**之精髓所在**)**，直到找出所有可能的**frequent patterns**。最後再於

```python
    def mine_patterns(self, min_support):
        """Mine frequent patterns"""
        patterns = {}
        for item, nodes in self.headers.items():
            support = sum(node['count'] for node in nodes)
            if support >= min_support:
                patterns[frozenset([item])] = support
                self._mine_conditional_tree(item, min_support, patterns, suffix=[item])
        return patterns

    def _mine_conditional_tree(self, item, min_support, patterns, suffix):
        """Recursively mine conditional pattern tree"""
        conditional_tree = FPTree()
        for node in self.headers[item]:
            path = []
            parent = node['parent']
            while parent and parent['name']:
                path.append(parent['name'])
                parent = parent['parent']
            conditional_tree.add_transaction(path, node['count'])

        pruned_patterns = conditional_tree.mine_patterns(min_support)
        for new_item, new_support in pruned_patterns.items():
            if new_support >= min_support:
                new_pattern = suffix + list(new_item)
                patterns[frozenset(new_pattern)] = new_support

        return patterns
```

圖27 為 mine_patterns、_mine_conditional_tree函數

至於執行runFPGrowth時，此函數會遍歷所有交易，將每個交易添加到 FP-Tree中，並對交易中的項目進行排序，以確保交易中項目的順序和原始data一致。接著，再調用**tree.mine_patterns()**來挖掘 FP-Tree中的**frequent Itemsets**。**至於，寫入txt檔案的function依然是將支持度由大至小排序並輸出(並未改變)。**

```python
def runFPGrowth(transactions, minSupport):
    """Run FP-Growth algorithm and generate frequent itemsets"""
    tree = FPTree()
    for transaction in transactions:
        tree.add_transaction(sorted(transaction))
    return tree.mine_patterns(minSupport * len(transactions))

def write_itemsets_to_file(itemsets, filename, total_transactions):
    """Write frequent itemsets to file preserving the order within itemsets."""
    with open(filename, 'w') as f:
        for itemset, support in sorted(itemsets.items(), key=lambda x: -x[1]):  # Sort by support
            support_percent = (support / total_transactions) * 100
            f.write(f"{support_percent:.1f}\t{{{', '.join(itemset)}}}\n")
```

圖28 為 runFPGrowth、write_itemsets_to_file函數

最後，主函數與Step2.py不同之處在於因為Step3.py為Non-Candidate-Based的方式，因此不必輸出result2.txt此外也僅需要計算Task1之時間，因此則是計算從開始執行FP Growth至輸出文檔為止。其餘程式碼僅有更動檔名以及刪除非Task1部分，並未做多餘之更動。

```python
# Main program
if __name__ == "__main__":
    optparser = OptionParser()
    optparser.add_option("-f", "--inputFile", dest="input", help="Input file containing dataset", default='datasetA.data')
    optparser.add_option("-s", "--minSupport", dest="minS", help="Minimum support value", default=0.1, type="float")
    optparser.add_option("-p", "--step", dest="step", help="Step number (2 or 3)", default='3')

    (options, args) = optparser.parse_args()

    if options.input is None:
        sys.exit("Dataset file must be provided.")
    else:
        transactions = list(dataFromFile(options.input))
        total_transactions = len(transactions)

    dataset_name = options.input.split('.')[0]
    minSupport = options.minS

    # Task 1: Frequent itemset mining
    start_time_task1 = time.perf_counter()
    frequent_itemsets = runFPGrowth(transactions, minSupport)
    task1_itemsets_file = f"step{options.step}_task1_{dataset_name}_{minSupport:.3f}_result1.txt"
    write_itemsets_to_file(frequent_itemsets, task1_itemsets_file, total_transactions)

    end_time_task1 = time.perf_counter()
    task1_time = end_time_task1 - start_time_task1

    print(f"Task 1 computation time: {task1_time:.6f} seconds")

    print("files generated.")
```

圖29 為 Step3.py主函數之內容

3.3 Computation time

3.3.1 datasetA (Set number of transactions = 1,000, number of items = 600)

(1) set min_support = 0.3%

- **Task1(Apriori): 15.818278s、Task1(FP-Growth): 0.165210s**
- **ratio of the speedup: 99.0357%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.003 -p 2
Number of closed itemsets found: 5033
Task 1 computation time in Apriori: 15.818278 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 1.525664 seconds
Total computation time in Apriori: 17.343942 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 9.644944%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 109.644944%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetA.data -s 0.003 -p 3
Task 1 computation time: 0.165210 seconds
files generated.
```

圖30、31為datasetA使用min_support = 0.3%於Apriori、FP-Growth執行時間與比率

(2) set min_support = 0.6%

- **Task1(Apriori): 3.290804s、Task1(FP-Growth): 0.065440s**
- **ratio of the speedup: 98.0114%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.006 -p 2
Number of closed itemsets found: 856
Task 1 computation time in Apriori: 3.290804 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.016505 seconds
Total computation time in Apriori: 3.307308 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.501537%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.501537%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetA.data -s 0.006 -p 3
Task 1 computation time: 0.065440 seconds
files generated.
```

圖32、33為datasetA使用min_support = 0.6%於Apriori、FP-Growth執行時間與比率

(3) set min_support = 0.9%

- **Task1(Apriori): 2.222030s、Task1(FP-Growth): 0.057782s**
- **ratio of the speedup: 97.3996%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetA.data -s 0.009 -p 2
Number of closed itemsets found: 407
Task 1 computation time in Apriori: 2.222030 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.004240 seconds
Total computation time in Apriori: 2.226270 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.190803%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.190803%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetA.data -s 0.009 -p 3
Task 1 computation time: 0.057782 seconds
files generated.
```

圖34、35為datasetA使用min_support = 0.9%於Apriori、FP-Growth執行時間與比率

3.3.2 datasetB (Set number of transactions = 100,000, number of items = 500)

(1) set min_support = 0.2%

- **Task1(Apriori): 2225.649055s、Task1(FP-Growth): 15.746798s**
- **ratio of the speedup: 99.2925%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.002 -p 2
Number of closed itemsets found: 5911
Task 1 computation time in Apriori: 2225.649055 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.657269 seconds
Total computation time in Apriori: 2226.306324 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.029532%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.029532%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetB.data -s 0.002 -p 3
Task 1 computation time: 15.746798 seconds
files generated.
```

圖36、37為datasetB使用min_support = 0.2%於Apriori、FP-Growth執行時間與比率

(2) set min_support = 0.4%

- **Task1(Apriori): 548.142202s、Task1(FP-Growth): 10.988542s**
- **ratio of the speedup: 97.9953%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.004 -p 2
Number of closed itemsets found: 1273
Task 1 computation time in Apriori: 548.142202 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.034530 seconds
Total computation time in Apriori: 548.176732 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.006299%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.006299%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetB.data -s 0.004 -p 3
Task 1 computation time: 10.988542 seconds
files generated.
```

圖38、39為datasetB使用min_support = 0.4%於Apriori、FP-Growth執行時間與比率

(3) set min_support = 0.6%

- **Task1(Apriori): 396.062270s、Task1(FP-Growth): 9.337185s**
- **ratio of the speedup: 97.6425%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetB.data -s 0.006 -p 2
Number of closed itemsets found: 545
Task 1 computation time in Apriori: 396.062270 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.006865 seconds
Total computation time in Apriori: 396.069135 seconds

----------------------------------------Time Ratio will be shown below----------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.001733%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.001733%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetB.data -s 0.006 -p 3
Task 1 computation time: 9.337185 seconds
files generated.
```

圖40、41為datasetB使用min_support = 0.6%於Apriori、FP-Growth執行時間與比率

3.3.3 datasetC (Set number of transactions = 500,000, number of items = 500)

(1) set min_support = 0.5%

- **Task1(Apriori): 2259.422121s、Task1(FP-Growth): 51.476087s**
- **ratio of the speedup: 97.7217%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.005 -p 2
Number of closed itemsets found: 753
Task 1 computation time in Apriori: 2259.422121 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.013117 seconds
Total computation time in Apriori: 2259.435237 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000581%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000581%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetC.data -s 0.005 -p 3
Task 1 computation time: 51.476087 seconds
files generated.
```

圖42、43為datasetC使用min_support = 0.5%於Apriori、FP-Growth執行時間與比率

(2) set min_support = 1.0%

- **Task1(Apriori): 1214.357689s、Task1(FP-Growth): 42.996650s**
- **ratio of the speedup: 96.4593%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.010 -p 2
Number of closed itemsets found: 298
Task 1 computation time in Apriori: 1214.357689 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.002537 seconds
Total computation time in Apriori: 1214.360226 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000209%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000209%
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetC.data -s 0.010 -p 3
Task 1 computation time: 42.996650 seconds
files generated.
```

圖44、45為datasetC使用min_support = 1.0%於Apriori、FP-Growth執行時間與比率

(3) set min_support = 1.5%

- **Task1(Apriori): 823.316026s、Task1(FP-Growth): 41.429617s**
- **ratio of the speedup: 95.0962%**

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step2.py -f datasetC.data -s 0.015 -p 2
Number of closed itemsets found: 241
Task 1 computation time in Apriori: 823.316026 seconds
Task 2 (Only Searching closed Itemset) computation time in Apriori: 0.001983 seconds
Total computation time in Apriori: 823.318008 seconds

--------------------------------------Time Ratio will be shown below--------------------------------------

[Task 2 (Only Searching closed Itemsets) / Task 1) computation time ratio]: 0.000241%
[Task 2 (Includes the operations of Task1) / Task 1) computation time ratio]: 100.000241%
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetC.data -s 0.015 -p 3
Task 1 computation time(FP-Growth): 41.429617 seconds
Number of closed itemsets found: 241
```

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetC.data -s 0.015 -p 3
Task 1 computation time: 40.373872 seconds
files generated.
```

圖46、47為datasetC使用min_support = 1.5%於Apriori、FP-Growth執行時間與比率

3.4 Discuss the scalability/restrictions of my algorithm

根據3.3章節統整spec要求在各組最小支持度下，使用我們設計的FP-Growth演算法相較於Apriori演算法之加速比率如**表四至六**。我們根據3.3章節數據以及四至六可看出**當資料集較大且設定較低的最小支持度時，使用FP-Growth演算法加速的比率是相當之高的**，不過從datasetA、B、C中運行速度可以逐漸看出我們設計的FP-Growth演算法可能在更往上一些量級時，其運算的時間也逐漸大幅升高。

因此，我們分別使用**$ ./gen lit -ntrans 5000 -tlen 10 -nitems 0.5 -fname datasetX -ascii**生成名為**datasetX具有500萬筆**的交易資料(**min_support設為5%**)，試圖找到我們設計之FP-Growth演算法在更大量級時所能最大承受的規格大小為何，其結果如**圖48大約花了4.5個小時才執行完畢**，然而，此時改使用Apriori(圖49)則是花了大約6474.6秒就執行完畢，不過由先前3.3章節可看出Apriori演算法對於使用較大最小支持度時會因為剩餘Itemsets較少而使得執行時間大幅縮短，然而，**這現象在FP-Growth演算法似乎是較不明顯的，因此不能武斷地推斷當資料集超過500萬筆時使用**Apriori演算法會較快，因為min_support若是設1%可能就又是**FP-Growth演算法較有效率。**

不過可以確定的是(1)**當資料集的scale成長至一定程度(如數百萬筆)**且**(2)最小支持度設定較高**，符合此兩種情況時，**則使用FP-Growth演算法未必會較Apriori演算法有效率**。

datasetA(1K) \ min_sup	0.3%	0.6%	0.9%
Speedup ratio≒	99.04%	98.01%	97.40%

表四為datasetA 於不同最小支持度時使用FP-Growth演算法相較於Apriori演算法之加速比率

datasetB(100K) \ min_sup	0.2%	0.4%	0.6%
Speedup ratio≒	99.29%	98.00%	97.64%

表五為datasetB 於不同最小支持度時使用FP-Growth演算法相較於Apriori演算法之加速比率

datasetC(500K) \ min_sup	0.3%	0.6%	0.9%
Speedup ratio≒	97.72%	96.46%	95.10%

表六為datasetC 於不同最小支持度時使用FP-Growth演算法相較於Apriori演算法之加速比率

```
PS C:\Users\user\Desktop\DM_HW1_313553024_蘇柏叡\Apriori_python> python Step3.py -f datasetX.data -s 0.05 -p 3
Task 1 computation time: 16353.618729 seconds
files generated.
```

圖48 為datasetX使用min_support = 5%於FP-Growth執行時間

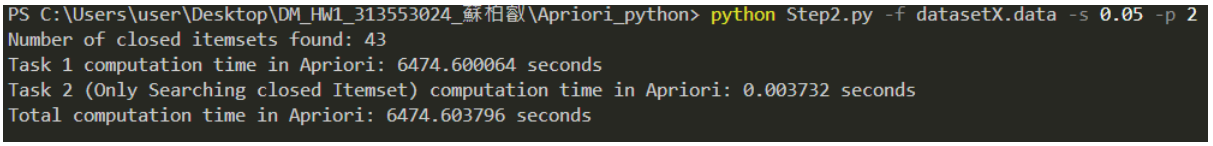

圖49 為datasetX使用min_support = 5%於Apriori執行時間

Supplement

為驗證Step2.py、Step3.py生成之檔案正確與否，筆者採用助教提供之**Toy dataset、Verification Code**進行測試，經測試後所得之結果皆和助教提供之**groundtruth file**匹配結果圖如**圖50、51、52，並且計算兩者Task1之speedup ratio為97.1712%**

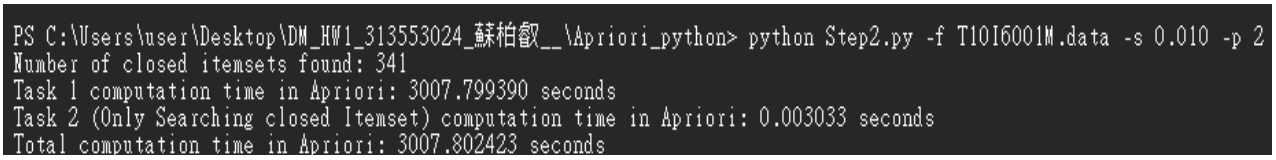

圖50 Toy dataset使用min_support = 1%於Apriori執行時間(Task1執行3007.799390s)

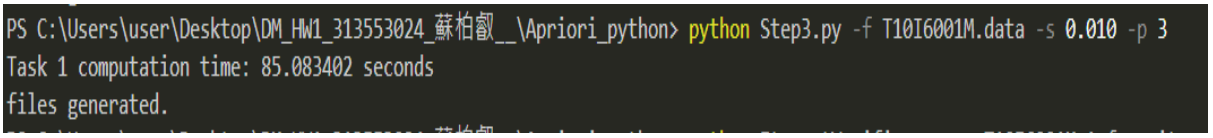

圖51 Toy dataset使用min_support = 1%於FP-Growth執行時間(Task1執行85.083402s)

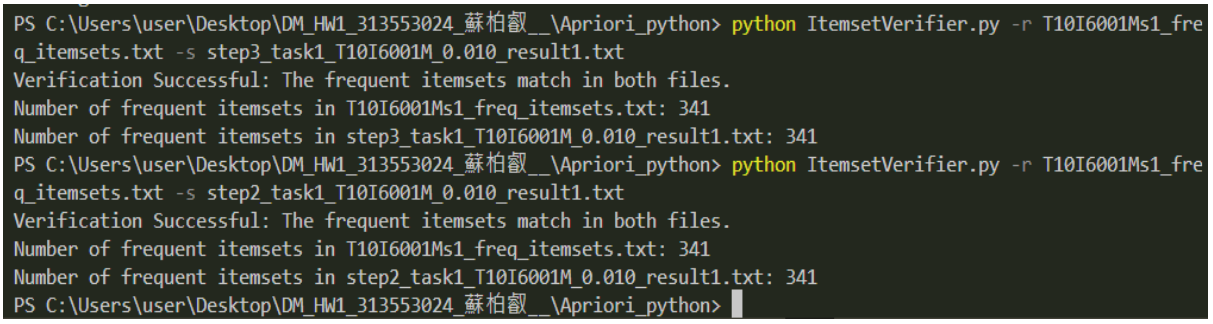

圖52為使用Step2.py、Step3.py生成之檔案與groundtruth比對後之結果(皆符合)。

Reference

[1] https://github.com/calee0219/Python3-Fp-growth

[2] https://github.com/university-subject/Python-FP-growth