tags: Data mining

HM1_Report

Step II

- **Report on the mining algorithms/codes**
    - ### The modifications you made for Task 1 and Task 2
        - 我在原本aprior裡面增加了兩個函式 ：
            - purning ： 去針對Task1 去做 aprior
            - closed_frequent_count ：針對 Task2 去做 closed item set。
        - 還增加三個函式 ：
            - result1 : 印出 Task1 的 result1
            - result2 : 印出 Task1 的 result2
            - Task2 : 印出 Task2 result
    - ### The restrictions
        - dataset過大的時候 -> 執行時間也會變得很久
    - ### Problems encountered in mining
        1. punring後 的 frequent itemset ， 不確定是不是對的 但是purning 的過程是按照演算法做
        2. 做出closed itemset ，但也不知道是否正確
    - ### Any observations/discoveries you want to share

        其實 stepII 的問題沒有比想像中難做，只是要按照演算法實做，應該都可以做出來。唯一麻煩的就是輸出的時候，要針對格式去做不一樣的改變。
- **screenshot of the computation time and ratio of task1/task2**

- datasetA:

```
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetA.csv -s 0.05
computation time Task1 : 0.099055s
computation time Task2 : 0.099071s
ratio of computation time : 100.016367%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetA.csv -s 0.075
computation time Task1 : 0.042712s
computation time Task2 : 0.042718s
ratio of computation time : 100.013955%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetA.csv -s 0.1
computation time Task1 : 0.036496s
computation time Task2 : 0.036503s
ratio of computation time : 100.018291%
weishon@weishon:~/data_mining/Apriori_python$
```

- datasetB:

```
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetB.csv -s 0.025
computation time Task1 : 69.551745s
computation time Task2 : 69.551772s
ratio of computation time : 100.000038%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetB.csv -s 0.05
computation time Task1 : 13.466947s
computation time Task2 : 13.466957s
ratio of computation time : 100.000076%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetB.csv -s 0.075
computation time Task1 : 13.225565s
computation time Task2 : 13.225578s
ratio of computation time : 100.000097%
weishon@weishon:~/data_mining/Apriori_python$
```

- datasetC:

```
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetC.csv -s 0.025
computation time Task1 : 655.055179s
computation time Task2 : 655.055208s
ratio of computation time : 100.000004%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetC.csv -s 0.05
computation time Task1 : 130.519536s
computation time Task2 : 130.519547s
ratio of computation time : 100.000008%
weishon@weishon:~/data_mining/Apriori_python$ python3 apriori.py -f datasetC.csv -s 0.1
computation time Task1 : 124.684972s
computation time Task2 : 124.684988s
ratio of computation time : 100.000012%
weishon@weishon:~/data_mining/Apriori_python$
```

- **screenshot of your code modification for Task 1 and Task 2**

  - **TASK1**

下面是我寫了一個函式專門去做purning,在做returnItemsWithMinSupport,的時候 我就會將不符合minsupport的存在 _shouldBePurning, 結束returnItemsWithMinSupport 後, 我就會做purning。

purning 作法：
先用一個大迴圈去跑目前的frequent itemset，
再用一個迴圈去跑不符合的frequent item subset。
若是frequent有包含到不是frequent item的東西，
那代表這個東西應該被踢出，所以我們就應該remove它
e.g. frequenct：ABC
unfrequent：AB
那就代表ABC不應該在frequent裡面，所以會從frequent itemset remove

```python
def purning(currentLSet , _shouldBePurningSet):
    for items in currentLSet:
        for purning_items in _shouldBePurningSet:
            if items.issubset(purning_items):
                currentLSet.remove(items)
                print("purning item : ",items)
                continue
    return currentLSet
```

我會將purning前後的candidates有幾個存起來

前：beforePunning

後：afterPunning

```python
    # 記得第一次purning 結果，其實根本沒有purning
    beforePunning.append(len(currentLSet))
    afterPunning.append(len(currentLSet))

    close_frequent_itemset_list.append(currentLSet)
    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)

        # 記得第k次purning 結果
        beforePunning.append(len(currentLSet))
        currentLSet = purning(currentLSet , _shouldBePurningSet)
        afterPunning.append(len(currentLSet))

        currentCSet , _shouldBePurningSet = returnItemsWithMinSupport(
            currentLSet, transactionList, minSupport, freqSet
        )
        currentLSet = currentCSet
        #先記得目前的frequent item set才能做後面的closed frequent item set
        if currentLSet != set([]) :
            close_frequent_itemset_list.append(currentLSet)
        k = k + 1
```

最後透過 result1 這個 function 印出frequent item set

```python
def result1(items , fd):
    frequence_itemset_counter=0
    for item, support in sorted(items, key=lambda x: x[1])[::-1]:
        fd.write("[{:.1f}%]\t[{:s}]\n".format(support*100,str(item)) )
        frequence_itemset_counter+=1
    return frequence_itemset_counter
```

最後透過 result2 這個 function 印出frequent itemset有幾個, before purning 以及 after purning 的candidates有幾個。

```python
def result2(frequence_itemset_counter , beforePunning , afterPunning , fd2):
    fd2.write('['+str(frequence_itemset_counter)+']'+'\n')
    k = 1
    for i in range(len(beforePunning)):
        fd2.write("[{:d}]\t[{:d}]\t[{:d}]\n".format( k , beforePunning[i] , afterPunning[i]))
        k+=1
```

- **Task2**

  freqSet 會記得 對應item 出現的次數

  for example:
  K:
  K=1: A, B, C, …
  K=2: AB, AC, BC,…
  K=3: ABC, ABD,…

item_sub -> A
tem -> AB
freqSet[item_sub] = A’s 出現次數
freqSet[item] = AB’s 出現次數

closed frequent itemset 作法：
用 close_frequent_itemset_list 先去存frequent item set
用 close_frequent_itemset去存我們要的 closed frequent itemset。

再來用一個迴圈去跑所有frequent itemset，它是根據 K 去跑，
再來用兩個迴圈分別跑 item 以及 item_sub，來看看兩邊的出現次數是不是不一樣。

通常是 item_sub 會大於 item，所以我用大於來跑。
若是的話，就加入close_frequent_itemset，
若不是的話，就繼續跑。

```python
def closed_frequent_count(close_frequent_itemset_list,freqSet):
    close_frequent_itemset=[]
    for k in range(len(close_frequent_itemset_list)):
        if k == len(close_frequent_itemset_list)-1:
            for item in close_frequent_itemset_list[k]:
                close_frequent_itemset.append(item)
            break
        else:
            # item_sub -> A , item -> AB , freqSet[item_sub] = A's support ,freqSet[item] = AB's support
            for item_sub in close_frequent_itemset_list[k]:
                for item in close_frequent_itemset_list[k+1]:
                    if item.issubset(item_sub):
                        if freqSet[item_sub]> freqSet[item]:
                            close_frequent_itemset.append(item_sub)
    # print(close_frequent_itemset)
    return close_frequent_itemset
```

下面這個task2，就是用來印出closed frequent itemset 有幾個，以及印出closed frequent itemset，以及對應的 support

```python
def task2(close_frequent_itemset , freqSet , transactionList , fd3):
    fd3.write('['+str(len(close_frequent_itemset))+']'+'\n')
    for i in range(len(close_frequent_itemset)):
        for j in range(len(close_frequent_itemset)):
            if freqSet[close_frequent_itemset[i]]> freqSet[close_frequent_itemset[j]]:
                temp = close_frequent_itemset[i]
                close_frequent_itemset[i] = close_frequent_itemset [j]
                close_frequent_itemset[j] = temp
    for i in range(len(close_frequent_itemset)):
        support = float(freqSet[close_frequent_itemset[i]])/len(transactionList)
        fd3.write("[{:.1f}%]\t".format(support*100))
        fd3.write('['+str(next(iter(close_frequent_itemset[i])))+']')
        fd3.write('\n')
```

step III

- **Descriptions of your mining algorithm**
    - **Relevant references**

FP_growth algorithm 介紹 : **https://zhuanlan.zhihu.com/p/67653006**

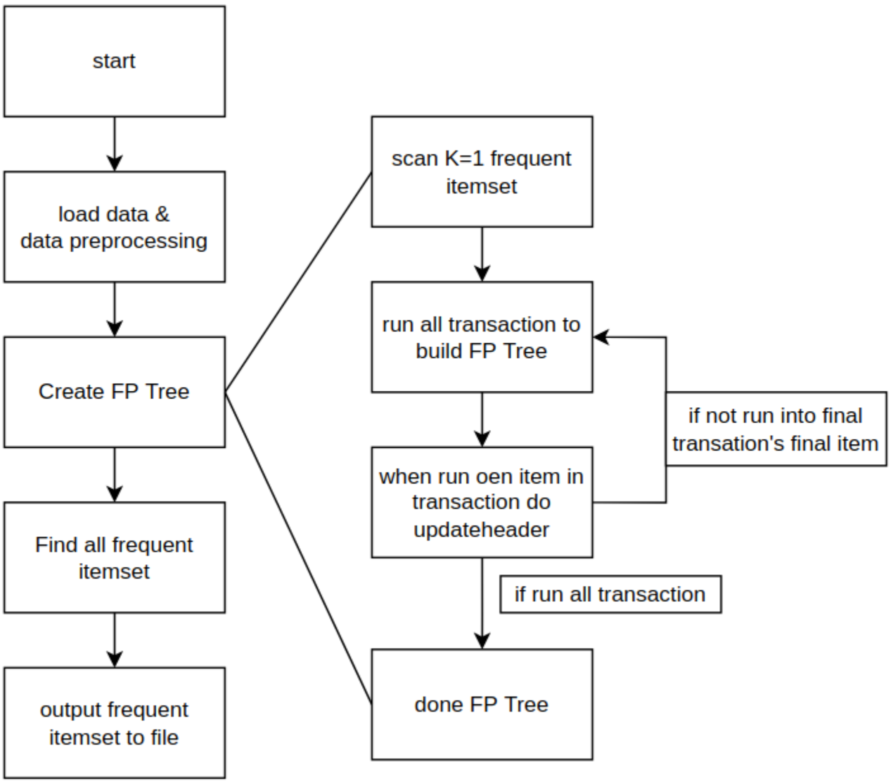
(https://zhuanlan.zhihu.com/p/67653006)

FP_growth github 來源網址: **https://github.com/Ryuk17/MachineLearning**

(https://github.com/Ryuk17/MachineLearning)

- **Program flow**

  這個演算法非Candidate-based



- **Differences/Improvements in your algorithm**

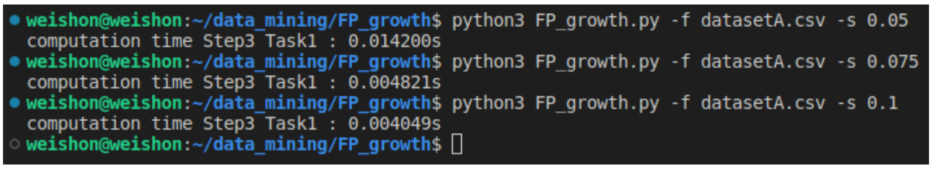
  - **Explain the main differences/improvements of your algorithm compared to Apriori.**

    我的演算法只需要遍歷整個dataset兩次，想比apriori會節省非常多的時間。但在建立FP_Tree的過程，若是滿足min_support的item過多，裡面要updateheader就會花很多時間，因為要遍歷整顆FP_Tree。

  - **If you use open-source code as the base, you should describe the modifications your made on the original algorithm.**

1. dataset的input，以及處理成algorithm能接受的格式
2. Create FP_tree 的 support (原本的support是次數，而我將它改成小數，也就是要除以transation的長度)
3. 增加寫檔案的函式

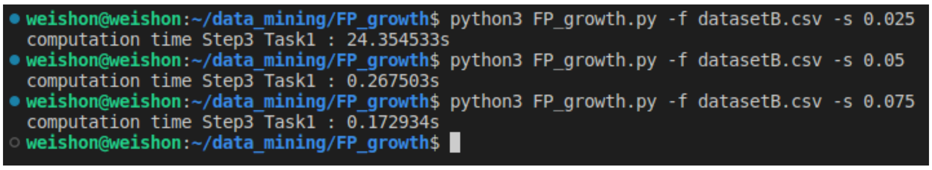
- **Computation time**
  - datasetA



```
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetA.csv -s 0.05
computation time Step3 Task1 : 0.014200s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetA.csv -s 0.075
computation time Step3 Task1 : 0.004821s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetA.csv -s 0.1
computation time Step3 Task1 : 0.004049s
weishon@weishon:~/data_mining/FP_growth$
```

1. min_support = 5.0%
   computation time StepII : 0.099005
   computation time StepIII: 0.014200
   speedup: 85.6%
2. min_support = 7.5%
   computation time StepII : 0.042712
   computation time StepIII: 0.004821
   speedup: 88.7%
3. min_support = 10.0%
   computation time StepII : 0.036496
   computation time StepIII: 0.004049
   speedup: 88.9%

  - datasetB



```
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetB.csv -s 0.025
computation time Step3 Task1 : 24.354533s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetB.csv -s 0.05
computation time Step3 Task1 : 0.267503s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetB.csv -s 0.075
computation time Step3 Task1 : 0.172934s
weishon@weishon:~/data_mining/FP_growth$
```

1. min_support = 2.5%
   computation time StepII : 69.551745
   computation time StepIII: 24.354553
   speedup: 65.0%
2. min_support = 5.0%
   computation time StepII : 13.466947
   computation time StepIII: 0.267503
   speedup: 98.0%

3. min_support = 7.5%

   computation time StepII : 13.225565

   computation time StepIII: 0.172934

   speedup: 98.7%

- datasetC

```
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetC.csv -s 0.025
computation time Step3 Task1 : 1356.336057s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetC.csv -s 0.05
computation time Step3 Task1 : 2.728867s
weishon@weishon:~/data_mining/FP_growth$ python3 FP_growth.py -f datasetC.csv -s 0.075
computation time Step3 Task1 : 1.691766s
weishon@weishon:~/data_mining/FP_growth$
```

1. min_support = 2.5%

   computation time StepII : 655.055179

   computation time StepIII: 1356.336057

   speedup: -107.1%

2. min_support = 5.0%

   computation time StepII : 130.519536

   computation time StepIII: 2.728867

   speedup: 97.9%

3. min_support = 7.5%

   computation time StepII : 124.684972

   computation time StepIII: 1.691176

   speedup: 98.6%

- **Discuss the scalability of your algorithm in terms of the size of dataset**

  當越小的 min_support以及越大的dataset 一起要做的時候，需要花更多的時間去建立FP_Tree，尤其是在我們的datasetC的時候，就可以明顯看出，在建立FP_tree，就花了很多時間，因為在找updateheader時，會去遍歷整樹，導致整個的時間比Apriori多了兩倍。但是在min_support提昇的時候，那個運行的時間就少了非常多。