資料探勘 DATA MINING Project 1

Q36071059 蔡榮漾

1. IBM Quest Data Generator.exe：

此程式主要之參數有三個：ntrans、tlen 與 nitems，分別意義如下

ntrans	transactions 數量
tlen	每筆 transactions 平均 items 數
nitems	items 種類數

另 IBM Quest Data Generator.exe 程式中輸入的 ntrans 與 nitems 參數會再乘上 1000 才是實際數字，使用時須注意，此外參數可為小數(ex.0.1)；也可利用 fname 參數自訂檔案名稱，否則使用默認檔案名稱。

本報告使用參數為：

lit -ntrans 0.1 -tlen 5 -nitems 0.01

產生之資料格式如下(只取前面小部分)：

1	1	0
1	1	6
1	1	7
1	1	8
1	1	9
2	2	4
2	2	5
2	2	6
2	2	8
3	3	0
3	3	7
4	4	4
4	4	6
4	4	7
4	4	8
5	5	0
5	5	5

根據個人理解，其意義應為：

TID	ITEMS
1	0,6,7,8,9
2	4,5,6,8
3	0,7
4	4,5,7,8
……	

由於 IBM Quest Data Generator.exe 產生之資料格式難以直接利用，
先將其整理，使用以下程式碼(Python2.7)：

```python
def datatrans():
    f = open('data.ntrans_0.1.tlen_5.nitems_0.01', 'r')

    input=f.read()
    list=input.split()
    #print(list)

    data=[list[i:i+3] for i in range(0,len(list),3)] #list fragment
    #print(data)

    data2=[]
    for i in range(0,len(list)/3):
        j=3*i+2
        data2.append(list[j])
    #print(data2)

    data3=[]
    data4=[]
    for i in range(1,len(list)/3):
       data3.append(data2[i-1])
       if(int(list[i*3])-int(list[(i-1)*3])==1):
         data4.append(data3)
         data3=[]
    data3.append(data2[i])
    data4.append(data3)
    #print(data4)
    return data4
```

Figure 1

先讀取資料檔案，將內容放至一個串列 list，再將 list 轉為巢狀串列把每筆資料分開放至 data，去掉 TID 放至 data2，最後以巢狀串列區隔每筆不一樣的 transactions。

得到之 data4 如下：

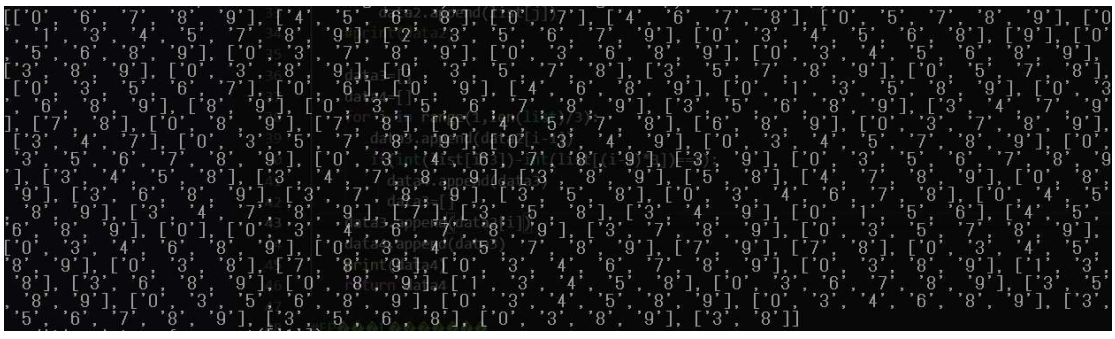


Figure 2

至此資料之處理暫告一段落，後續將提到此 datatrans()函式如何使用。

2. WEKA：

由於 WEKA 似乎只吃 .arff 檔案，
且有特定格式，經查詢後，撰寫程式 trans_arff.py，
將先前 IBM Quest Data Generator.exe 所產生之原始資料
data.ntrans_0.1.tlen_5.nitems_0.01 轉為稀疏型資料 weka_data.arff。

weka_data.arff 之格式如下：

```
@relation 'TestID'
@attribute 0 {F, T}
@attribute 1 {F, T}
@attribute 2 {F, T}
@attribute 3 {F, T}
@attribute 4 {F, T}
@attribute 5 {F, T}
@attribute 6 {F, T}
@attribute 7 {F, T}
@attribute 8 {F, T}
@attribute 9 {F, T}
@data
{0 T, 6 T, 7 T, 8 T, 9 T}
{4 T, 5 T, 6 T, 8 T}
{0 T, 7 T}
{4 T, 6 T, 7 T, 8 T}
{0 T, 5 T, 7 T, 8 T, 9 T}
{0 T, 1 T, 3 T, 4 T, 5 T, 7 T, 8 T, 9 T}
{2 T, 3 T, 5 T, 6 T, 7 T, 9 T}
{0 T, 3 T, 4 T, 5 T, 6 T, 8 T}
{9 T}
{0 T, 5 T, 6 T, 8 T, 9 T}
{0 T, 3 T, 7 T, 8 T, 9 T}
{0 T, 3 T, 6 T, 8 T, 9 T}
{0 T, 3 T, 4 T, 5 T, 6 T, 8 T, 9 T}
{3 T, 8 T, 9 T}
{0 T, 3 T, 8 T, 9 T}
```

Figure 3

簡單舉例，此 data(data.ntrans_0.1.tlen_5.nitems_0.01)共有 0~9 10 種 items，
第一行必須先為此筆資料(@relation)命名，
再來則要列出所有 items(@attribute 0 {F, T}~@attribute 9 {F, T})，
然後以一行@data 隔開，此後一行放置一個 transaction。

先使用 WEKA 對此資料進行測試，進行 FP_Growth，使用參數為：

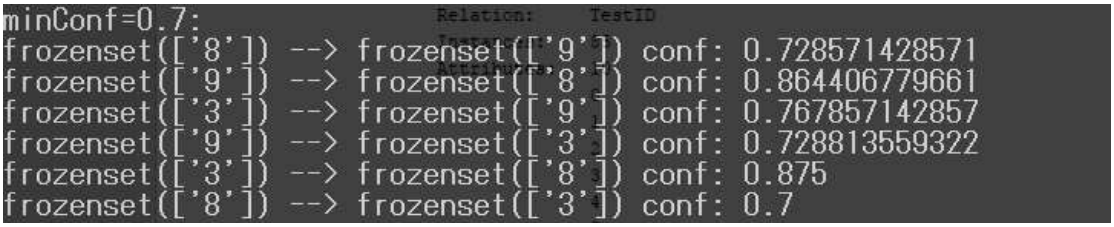
FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.7 -D 0.05 -U 1.0 -M 0.5

得到之結果：

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.7 -D 0.05 -U 1.0 -M 0.5
Relation:     TestID
Instances:    85
Attributes:   10
              0
              1
              2
              3
              4
              5
              6
              7
              8
              9
=== Associator model (full training set) ===

FPGrowth found 6 rules (displaying top 6)

1. [3=T]: 56 ==> [8=T]: 49   <conf:(0.88)> lift:(1.06) lev:(0.03) conv:(1.24)
2. [9=T]: 59 ==> [8=T]: 51   <conf:(0.86)> lift:(1.05) lev:(0.03) conv:(1.16)
3. [3=T]: 56 ==> [9=T]: 43   <conf:(0.77)> lift:(1.11) lev:(0.05) conv:(1.22)
4. [9=T]: 59 ==> [3=T]: 43   <conf:(0.73)> lift:(1.11) lev:(0.05) conv:(1.18)
5. [8=T]: 70 ==> [9=T]: 51   <conf:(0.73)> lift:(1.05) lev:(0.03) conv:(1.07)
6. [8=T]: 70 ==> [3=T]: 49   <conf:(0.7)> lift:(1.06) lev:(0.03) conv:(1.09)
```

Figure 4

以 Apriori 演算法進行驗證(minConf:0.7 minSup 0.5)，得知 WEKA 操作上沒有問題，產生之關聯法則與各信賴度大致相同。



3. FP-Growth 之實作(fpg.py)：

建立 FP-tree：

在開始建樹之前，需先決定節點構造：

```python
class treeNode:
    def __init__(self,nameValue, numOccur, parentNode):
        self.name = nameValue
        self.count = numOccur
        self.parent = parentNode
        self.children = {}
        self.nodeLink = None

    def inc(self, numOccur):
        self.count += numOccur

    def disp(self, ind = 1):
        print "  " * ind, self.name, "  ",self.count
        for child in self.children.values():
            child.disp(ind + 1)
```

Figure 5

name	元素名稱
count	元素出現次數
parent	指向子點
children	指向父點
nodeLink	指向下一相同名稱節點
inc()	計算元素出現次數
disp()	輸出節點&子點的 FP 結構

建立 FP-tree：

```python
def createTree(dataSet, minSup=1):  #creat fp-tree
    headerTable = {}
    for trans in dataSet:
        for item in trans:
            headerTable[item] = headerTable.get(item, 0) + dataSet[trans]
    for k in headerTable.keys():    #remove item < minSup
        if headerTable[k] < minSup:
            del(headerTable[k])
    freqItemSet = set(headerTable.keys())
    if len(freqItemSet) == 0:   #if [] return none
        return None, None
    for k in headerTable:
        headerTable[k] = [headerTable[k], None]
    retTree = treeNode('Null Set', 1, None)     #Tree root
    for tranSet, count in dataSet.items():
        localD = {}
        for item in tranSet:
            if item in freqItemSet:
                localD[item] = headerTable[item][0]
        if len(localD) > 0:
            orderedItems = [v[0] for v in sorted(localD.items(), key=lambda p: p[1], reverse=True)] #sort
            updateTree(orderedItems, retTree, headerTable, count)
    return retTree, headerTable
```

Figure 6

建立 Conditional FP-tree：

```
109 def mineTree(inTree,headerTable,minSup,preFix,freqItemList):
110     bigL = [v[0] for v in sorted(headerTable.items(),key = lambda p:p[1])]
111     for basePat in bigL:
112         newFreqSet = preFix.copy()
113         newFreqSet.add(basePat)
114         freqItemList.append(newFreqSet)
115         condPattBases = findPrefixPath(basePat, headerTable[basePat][1])
116         myConTree,myHead = createTree(condPattBases, minSup)
117 
118         if myHead != None:
119             print 'conditional tree for :', newFreqSet
120             myConTree.disp()
121 
122             mineTree(myConTree, myHead, minSup, newFreqSet, freqItemList)
```

Figure 7

此處流程大致上是，藉由前處 creatTree() 函式所建立之 FP-tree 得到 CPB(Conditional Pattern Base)，並利用 CPB 建立 Conditional FP-tree，然後持續遞迴，直到樹只包含一個 item。

主程式：

```
131 if __name__=="__main__":
132     dataSet = datatrans()
133     freqItems = fpGrowth(dataSet)
134     print freqItems
```

Figure 8

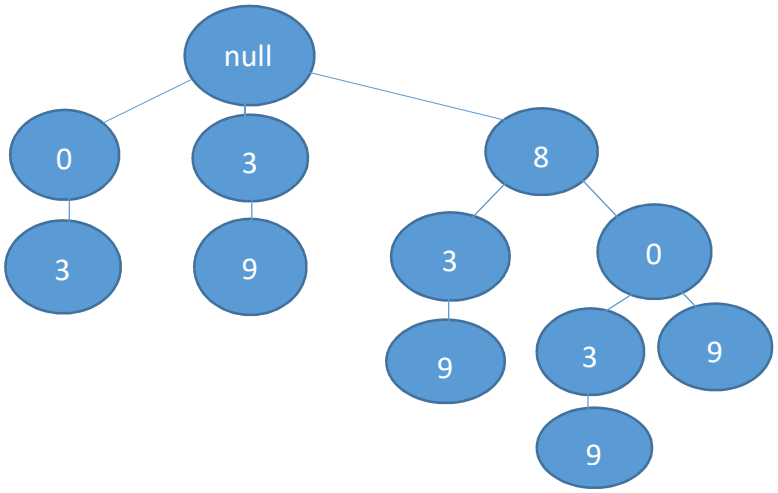
此處利用 datatrans()將 IBM Quest Data Generator.exe 產生之數據轉換格式並放到變數 dataSet 供其他函式使用，之後運行 fpGrowth()函數。

樹的顯示方式：

```
conditional tree for : set(['5', '7'])
  Null Set   1
    8    10
      0    8
        9    1
        3    5
          9    4
      3    2
        9    2
    3    1
      9    1
    0    1
      3    1
```

Figure 9

以此為例，可表示為以下樹：



得到之頻繁項集：

minSup 為 30 時，Frequent Patterns 為：

```
[set(['0']), set(['3']), set(['8', '3']), set(['9']), set(['9', '8']), set(['8'])]
```

minSup 為 20 時，Frequent Patterns 為：

```
[set(['4']), set(['6']), set(['8', '6']), set(['7']), set(['8', '7']), set(['5']), set(['3', '5']), set(['8', '5']
), set(['0']), set(['0', '3']), set(['9', '0']), set(['0', '8']), set(['3']), set(['9', '3']), set(['9', '8', '3']
), set(['8', '3']), set(['9']), set(['9', '8']), set(['8'])]
```

minSup 為 10 時，Frequent Patterns 為：

```
[set(['4']), set(['4', '7']), set(['0', '4']), set(['0', '4', '8']), set(['5', '4']), set(['8', '5', '4']), set(['
9', '4']), set(['9', '3', '4']), set(['9', '8', '4']), set(['3', '4']), set(['8', '3', '4']), set(['8', '4']), set
(['6']), set(['5', '6']), set(['8', '5', '6']), set(['0', '6']), set(['0', '3', '6']), set(['0', '8', '6']), set([
'3', '6']), set(['9', '3', '6']), set(['9', '8', '3', '6']), set(['8', '3', '6']), set(['9', '6']), set(['9', '8',
'6']), set(['8', '6']), set(['7']), set(['5', '7']), set(['8', '5', '7']), set(['0', '7']), set(['0', '3', '7']),
 set(['0', '7', '8']), set(['3', '7']), set(['8', '3', '7']), set(['9', '8', '3', '7']), set(['9', '3', '7']), set
(['9', '7']), set(['9', '8', '7']), set(['8', '7']), set(['5']), set(['0', '5']), set(['0', '3', '5']), set(['0',
'3', '5', '8']), set(['9', '0', '5']), set(['9', '0', '5', '8']), set(['0', '5', '8']), set(['9', '5']), set(['9',
'3', '5']), set(['9', '8', '3', '5']), set(['9', '8', '5']), set(['3', '5']), set(['8', '3', '5']), set(['8', '5'
]), set(['0']), set(['0', '3']), set(['9', '0', '3']), set(['9', '0', '3', '8']), set(['0', '3', '8']), set(['9',
'0']), set(['9', '0', '8']), set(['0', '8']), set(['3']), set(['9', '3']), set(['9', '8', '3']), set(['8', '3']),
set(['9']), set(['9', '8']), set(['8'])]
```

觀察得知，FP-tree 建構時所設立的 min support 值越高，確實如預想的會讓頻繁項集數量減少，亦可作為驗證程式正確性的步驟之一。

利用頻繁項集找尋關聯法則與計算 confident：

```
for item_i in freqItems:
    for item_j in freqItems:
        if item_i != item_j:
            item_i_count=0
            item_j_count=0
            if not (checkIsexist(item_i, item_j) or checkIsexist(item_j, item_i) or checkAnyexist(item_i, item_j)):
                for datas in simpDat:
                    if checkIsexist(item_i, datas):
                        item_i_count += 1

                    if checkIsexist(item_i, datas) and checkIsexist(item_j, datas):
                        item_j_count += 1
                if item_i_count != 0 and item_j_count != 0 and item_j_count/item_i_count>=0.5 and item_j_count!=1: ···
```

Figure 10

得到之結果：

```
{'4'}  --->  {'3'} conf : 0.75
{'4'}  --->  {'9'} conf : 0.7142857142857143
{'4'}  --->  {'8'} conf : 0.8571428571428571
{'6'}  --->  {'3'} conf : 0.7096774193548387
{'6'}  --->  {'9'} conf : 0.7419354838709677
{'6'}  --->  {'8', '9'} conf : 0.7096774193548387
{'6'}  --->  {'8'} conf : 0.8709677419354839
{'5'}  --->  {'3'} conf : 0.7428571428571429
{'5'}  --->  {'8'} conf : 0.9142857142857143
{'7'}  --->  {'8'} conf : 0.8205128205128205
{'0'}  --->  {'9'} conf : 0.7608695652173914
{'0'}  --->  {'8', '9'} conf : 0.717391304347826
{'0'}  --->  {'8'} conf : 0.8478260869565217
{'1'}  --->  {'5'} conf : 0.8
{'1'}  --->  {'3'} conf : 0.8
{'1'}  --->  {'3', '8'} conf : 0.8
{'1'}  --->  {'8'} conf : 0.8
{'3'}  --->  {'9'} conf : 0.7678571428571429
{'3'}  --->  {'8'} conf : 0.875
{'3', '9'}  --->  {'8'} conf : 0.8837209302325582
{'3', '8'}  --->  {'9'} conf : 0.7755102040816326
{'9'}  --->  {'3'} conf : 0.7288135593220338
{'9'}  --->  {'8'} conf : 0.864406779661017
{'8', '9'}  --->  {'3'} conf : 0.7450980392156863
{'8'}  --->  {'3'} conf : 0.7
{'8'}  --->  {'9'} conf : 0.7285714285714285
```

Figure 11

與 WEKA 所得之結果比較：

```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.7 -D 0.05 -U 1.0 -M 2.0
Relation:     TestID
Instances:    85
Attributes:   10
              0
              1
              2
              3
              4
              5
              6
              7
              8
              9
=== Associator model (full training set) ===

FPGrowth found 11 rules (displaying top 10)

 1. [9=T, 3=T]: 43 ==> [8=T]: 38   <conf:(0.88)> lift:(1.07) lev:(0.03) conv:(1.26)
 2. [3=T]: 56 ==> [8=T]: 49   <conf:(0.88)> lift:(1.06) lev:(0.03) conv:(1.24)
 3. [9=T]: 59 ==> [8=T]: 51   <conf:(0.86)> lift:(1.05) lev:(0.03) conv:(1.16)
 4. [0=T]: 46 ==> [8=T]: 39   <conf:(0.85)> lift:(1.03) lev:(0.01) conv:(1.01)
 5. [8=T, 3=T]: 49 ==> [9=T]: 38   <conf:(0.78)> lift:(1.12) lev:(0.05) conv:(1.25)
 6. [3=T]: 56 ==> [9=T]: 43   <conf:(0.77)> lift:(1.11) lev:(0.05) conv:(1.22)
 7. [0=T]: 46 ==> [9=T]: 35   <conf:(0.76)> lift:(1.1) lev:(0.04) conv:(1.17)
 8. [8=T, 9=T]: 51 ==> [3=T]: 38   <conf:(0.75)> lift:(1.13) lev:(0.05) conv:(1.24)
 9. [9=T]: 59 ==> [3=T]: 43   <conf:(0.73)> lift:(1.11) lev:(0.05) conv:(1.18)
10. [8=T]: 70 ==> [9=T]: 51   <conf:(0.73)> lift:(1.05) lev:(0.03) conv:(1.07)
```

Figure 12

發現兩者產生之關聯法則與其 conf 大致相同。
列出前三者為例進行比較：
[9,3]-->[8] conf:0.88
[3]-->[8] conf:0.88
[9]-->[8] conf:0.86
確實到小數點第三位為止並無太大差異，
可知撰寫之程式所產生的結果應該正確；
唯產生之關聯法則數目有差異，
此部分是否為參數設定上之差異，還可進一步測試。