程式運行步驟說明

1. 步驟Ⅱ:程式運行說明
   - Task1

     呼叫指令: python task1.py -f my_dataset\DataA.csv -s 0.002

     生成 mini_suppot 0.002，DataA 的頻繁資料集，寫入路徑為 f"result\\output_in_step2\\task1\\Result_file1\\step2_task1_{filename}_{minSupport_task1}.txt"

     紀錄時間路徑:

     "result\\output_in_step2\\task1\\computation_time_task1.txt"

     紀錄運行時間
   - Task2

     呼叫指令: python task2.py -f my_dataset\DataA.csv -s 0.002

     生成 mini_suppot 0.002，DataA 的挖掘所有頻繁閉項集，寫入路徑為

     f'result\\output_in_step2\\task2\\Result_file1\\task2_{filename}_{minSupport_task2}.txt

     紀錄時間路徑:

     "result\\output_in_step2\\task2\\computation_time_task2.txt"

     紀錄運行時間

2. 步驟Ⅲ：程式運行說明

   呼叫指令: 檔案名:

   python task3.py -f my_dataset\DataA.data -c result\output_in_step3\DataA.csv -s 0.002

   參數:

   -f 或 --inputFile：原始數據檔的路徑。

   -c 或 --csvFile：（選填）CSV 檔保存的路徑。

   -s 或 --minSupport：（選填）FP-Growth 演算法的最小支援度閾值，預設為 0.1。

步驟Ⅱ 程式說明

1. Task1

   使用 split 的方式去掉不必要的 index

```python
# 從文件中讀取數據
def dataFromFile(fname):
    with open(fname, "r") as file_iter:
        for line in file_iter:
            line = line.strip().rstrip(",")
            # 從第三個元素到列表末尾
            line =','.join(line.split(',')[3:])
            record = frozenset(line.split(","))
            yield record
```

針對原本的 apriori.py 程式加入計算剪枝後候選集數目，以便完成 task1 Result_file2，並且計算 candidates_before_pruning, candidates_after_pruning 數量

```python
# 返回滿足最小支持度的項目集
def returnItemsWithminSupport_task1(itemSet, transactionList, minSupport_task1, freqSet):
    _itemSet = set()
    localSet = defaultdict(int)
    # 剪枝前的候選項目集數
    candidates_before_pruning = len(itemSet)

    for item in itemSet:
        for transaction in transactionList:
            if item.issubset(transaction):
                freqSet[item] += 1
                localSet[item] += 1      HUGOKANG890308, 前天 • git commit
    # 剪枝後的候選項目集數
    candidates_after_pruning = 0
    for item, count in localSet.items():
        support = float(count) / len(transactionList)
        if support >= minSupport_task1:
            _itemSet.add(item)
            candidates_after_pruning += 1

    statistics_data_task1.append((candidates_before_pruning, candidates_after_pruning))
    return _itemSet
```

在 runApriori 函數改寫 return 方法，以便達成老師 sort 的要求

return sorted(toRetItems, key=lambda x: x[1], reverse=True)

```python
# 運行 Apriori 算法
def runApriori(data_iter, minSupport_task1):
    global total_frequent_itemsets_task1
    total_frequent_itemsets_task1 = 0
    itemSet, transactionList = getItemSetTransactionList(data_iter)
    freqSet = defaultdict(int)
    largeSet = dict()
    oneCSet = returnItemsWithminSupport_task1(itemSet, transactionList, minSupport_task1, freqSet)
    currentLSet = oneCSet
    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet = returnItemsWithminSupport_task1(
            currentLSet, transactionList, minSupport_task1, freqSet
        )
        currentLSet = currentCSet
        total_frequent_itemsets_task1 += len(currentLSet)
        k = k + 1

    # 計算支持度
    def getSupport(item):
        return float(freqSet[item]) / len(transactionList)

    toRetItems = []
    for key, value in largeSet.items():
        toRetItems.extend([((item), getSupport(item)) for item in value])
    return sorted(toRetItems, key=lambda x: x[1], reverse=True)  # Sort by support, from large to small
```

列印 Result_file1 & Result_file2

```python
# 打印結果
def printResults(items):
    global total_frequent_itemsets_task1
    with open(f"result\\output_in_step2\\task1\\Result_file1\\step2_task1_{filename}_{minSupport_task1}.txt", "w") as f1, open(f"result\\
        f1.write(f"{filename} : minimum support = {minSupport_task1}\n")
        # 已經排序
        for item, support in items:
            # \t 代表一個制表符（Tab字符），用來對齊
            f1.write(f"{round(support*100,4)}\t{{{' ,'.join(map(str, item))}}}\n")      HUGOKANG890308, 17 小時前 • git commit

        f2.write(f"total_frequent_itemsets_task1:{total_frequent_itemsets_task1}\n")
        for index, (before, after) in enumerate(statistics_data_task1, 1):
            f2.write(f"{index}\t{before}\t{after}\n")
```

Task1 計算時間結果(單位:秒)

Data / mini_sup	0.002	0.005	0.01
DataA	1.7565	3.7155	23.9179
Data / mini_sup	0.015	0.002	0.005
DataB	1638.5590	7935.8080	2807.2296
Data / mini_sup	0.01	0.02	0.03
DataC	4373.144	1804.2595	689.107

2. Task2

`getFrequentClosedItemsets` 函數的作用是從一組頻繁項目集中找出所有的頻繁閉項目集。這裡的「閉項目集」指的是不能再添加更多元素使支援度不變的項目集。具體來說，閉項目集滿足條件：沒有任何真超集合有相同的支援度計數。

以下是函數的邏輯步驟：

1. 初始化一個空列表 `frequent_closed_itemsets` 來存儲所有的頻繁閉項

目集。

2. 遍歷給定的頻繁項目集列表 `frequent_itemsets`。對於列表中的每一個項目集 `itemset1` 和其對應的支持度 `support1`：
   a. 設置一個標記 `is_closed` 為 `True`，假定當前的項目集是閉的。
   b. 再次遍歷 `frequent_itemsets` 列表，與列表中的其他項目集 `itemset2` 及其支持度 `support2` 進行比較。
   c. 如果存在一個不同的項目集 `itemset2`（即索引 `i` 不等於 `j`），使得 `itemset1` 是 `itemset2` 的子集，且兩者的支持度相等，那麼 `itemset1` 就不是閉項目集。在這種情況下，將 `is_closed` 設置為 `False`，並中斷內層循環。
3. 如果經過內層循環後，`is_closed` 仍然為 `True`（即沒有找到任何真超集合具有相同的支援度），則將 `itemset1` 和 `support1` 作為一個元組添加到 `frequent_closed_itemsets` 列表中。
4. 繼續對所有頻繁項目集進行此檢查。
5. 最終返回包含所有頻繁閉項目集的列表 `frequent_closed_itemsets`。

這個函數通過雙層循環，確保只有那些沒有支援度相等的真超集合的項目集被認定為閉項目集。

```python
# 獲取頻繁閉項目集
def getFrequentClosedItemsets(frequent_itemsets):
    # 初始化一個空列表來儲存頻繁閉項目集
    frequent_closed_itemsets = []

    # 遍歷所有頻繁項目集
    for i, (itemset1, support1) in enumerate(frequent_itemsets):
        # 初始化一個標記來檢查當前的項目集是否是閉的
        is_closed = True

        # 再次遍歷所有頻繁項目集以進行比較
        for j, (itemset2, support2) in enumerate(frequent_itemsets):
            # 檢查當前的項目集（itemset1）是否是另一個項目集（itemset2）的子集，
            # 並且它們的支持度是否相等
            if i != j and itemset1.issubset(itemset2) and support1 == support2:
                # 如果是，則將標記設置為False並中斷循環
                is_closed = False
                break
        # 如果標記仍然是True，則將當前的項目集添加到頻繁閉項目集列表中
        if is_closed:
            frequent_closed_itemsets.append((itemset1, support1))
    # 返回頻繁閉項目集列表
    return frequent_closed_itemsets
```

列印 task2 所需結果

```python
# 打印 Task 2 結果
def printResultsForTask2(items):
    frequent_closed_itemsets = getFrequentClosedItemsets(items)
    with open(f'result\\output_in_step2\\task2\\Result_file1\\task2_{filename}_{minSupport_task2}.txt', "w") as f:
        f.write(f"{filename} : minimum support = {minSupport_task2}\n")
        f.write(f"{len(frequent_closed_itemsets)}\n")
        for itemset, support in sorted(frequent_closed_itemsets, key=lambda x: x[1], reverse=True):
            f.write(f"{round(support * 100, 1)}%\t{{{' ,'.join(map(str, itemset))}}}\n")
```

Task1 與 Task2 時間比較 ratio 如下

Data / mini_sup	0.002	0.005	0.01
DataA	146.8785%	103.9780%	97.5721%
Data / mini_sup	0.015	0.002	0.005
DataB	103.0798%	99.6141%	100.1967%
Data / mini_sup	0.01	0.02	0.03
DataC	100.7974%	100.7705%	100.2057%

步驟Ⅲ 程式說明

- 演算法描述
  - 相關參考文獻

    Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 1-12).
  - 開源代碼引用

    本程式使用了 mlxtend 庫實現的 FP-Growth 演算法。mlxtend 是一個提供機器學習和資料採擷演算法的 Python 庫。
  - 程式流程
    I. 解析命令列參數以獲取原始數據檔路徑、CSV 檔案名和 mini_support。
    II. 檢查輸出目錄是否存在，不存在則創建。
    III. 將原始數據檔轉換為 CSV 格式。
    IV. 讀取 CSV 檔，將數據轉換為交易清單。
    V. 使用 FP-Growth 演算法找出頻繁項目集。
    VI. 將頻繁項目集和支援度寫入到結果檔。
    VII. 計算並記錄執行時間。
  - 算法差異/改進 與 Apriori 算法相比，FP-Growth 算法的主要改進:
    I. 不需要生成候選項目集，從而減少了計算量。
    II. 使用 FP-Tree 結構可以更有效地存儲和計算頻繁項目集。
    III. 整體上，FP-Growth 有更好的時間和空間效率。
- 主要程式碼

根據助教要求將.Data 檔案轉換為.csv 檔案

```python
def convertToCSV(input_file_path, output_dir):
    """
    將原始數據文件轉換為 CSV 格式。

    :param input_file_path: str, 原始數據文件的完整路徑。假設數據項目是以空格分隔的。
    :param output_dir: str, CSV 文件應該被保存的目錄的路徑。如果此目錄不存在，函數將創建它。
    :return: str, 新創建的 CSV 文件的路徑。

    使用範例:
    >>> input_data = 'path/to/your/datafile.data'
    >>> output_directory = 'path/to/output/directory'
    >>> csv_file_path = convertToCSV(input_data, output_directory)
    >>> print(csv_file_path)
    'path/to/output/directory/datafile.csv'
    """

    # 確保輸出目錄存在，如果不存在，則創建它
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
        print(f"Created directory {output_dir}")

    # 從原始文件名中獲取基本文件名，並將其擴展名從 .data 替換為 .csv
    output_csv_path = os.path.join(output_dir, os.path.basename(input_file_path).replace('.data', '.csv'))

    try:
        # 讀取原始文件並將數據寫入到 CSV 文件中
        with open(input_file_path, 'r') as infile, open(output_csv_path, 'w', newline='') as outfile:
            for line in infile:
                # 假設數據項目是以空格分隔的
                line_items = line.strip().split()
                # 用逗號連接數據項目，創建 CSV 格式的字符串
                csv_line = ','.join(line_items)
                # 將 CSV 字符串寫入文件
                outfile.write(csv_line + '\n')
        print(f"CSV file created at {output_csv_path}")
        return output_csv_path
    except Exception as e:
        # 如果在轉換過程中出現錯誤，打印錯誤訊息並退出程序
        print(f"Error converting to CSV: {e}")
        sys.exit(1)
```

Run_FPGrowth 運行 FP_Growth 演算法函式

```python
def runFPGrowth(transactionList, minSupport_task3):
    """
    使用 FP-Growth 算法找出頻繁項目集。
          You, 31 分鐘前 • Uncommitted changes
    :param transactionList: list of lists, 交易列表，每個交易包含一組項目。
    :param minSupport_task3: float, 尋找頻繁項目集時使用的最小支持度閾值。
    :return: DataFrame, 頻繁項目集及其支持度的數據框。

    使用範例:
    >>> transactions = [['牛奶', '麵包'], ['牛奶', '尿布'], ['麵包', '尿布', '啤酒'], ['牛奶', '麵包', '尿布', '啤酒']]
    >>> min_support = 0.5
    >>> frequent_itemsets = runFPGrowth(transactions, min_support)
    >>> print(frequent_itemsets)
    """

    # 初始化 TransactionEncoder 來轉換交易數據
    te = TransactionEncoder()
    # 將交易列表轉換為一個 bool 矩陣
    te_ary = te.fit(transactionList).transform(transactionList)
    # 將 bool 矩陣轉換為 DataFrame
    df = pd.DataFrame(te_ary, columns=te.columns_)
    # 使用 FP-Growth 算法找出頻繁項目集，並指定最小支持度
    frequent_itemsets_task3 = fpgrowth(df, min_support=minSupport_task3, use_colnames=True)
    # 返回找到的頻繁項目集
    return frequent_itemsets_task3
```

打印所需結果

```python
# 打印結果
def printResults(frequent_itemsets_task3, filename, minSupport_task3):
    # 結果輸出路徑可能需要根據您的文件結構進行調整
    output_path = f"result\\output_in_step3\\task1\\step_task3_{filename}_{minSupport_task3}.txt"      You, 33 分
    with open(output_path, "w") as f:
        for itemset in frequent_itemsets_task3.itertuples():
            # 頻繁項目集和支持度
            f.write(f"{round(itemset.support * 100, 4)}\t{{{' ,'.join(itemset.itemsets)}}}\n")
```

Task3 時間表 (單位:秒)

Data / mini_sup	0.002	0.005	0.01
DataA	0.4378	103.9780%	0.0528
Data / mini_sup	0.015	0.002	0.005
DataB	4.6201	7.5244	6.0755
Data / mini_sup	0.01	0.02	0.03
DataC	50.7432	38.0962	25.2208

以下是 Task 3 相較於 Task 1 的速度提升百分比的表格：

Data / mini_sup	0.002	0.005	0.01
DataA	75.08%	98.58%	97.82%
Data / mini_sup	0.015	0.002	0.005
DataB	99.72%	99.91%	99.78%
Data / mini_sup	0.01	0.02	0.03
DataC	98.84%	97.89%	96.34%