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

1. 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

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

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

一張含有 文字, 螢幕擷取畫面, 軟體, 作業系統 的圖片

自動產生的描述

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)

一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

列印 Result\_file1 & Result\_file2

一張含有 螢幕擷取畫面, 文字, 字型 的圖片

自動產生的描述

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 |

1. 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`：
   * 1. 設置一個標記 `is\_closed` 為 `True`，假定當前的項目集是閉的。
     2. 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。
     3. 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。

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述

列印task2所需結果

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

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
  1. 解析命令列參數以獲取原始數據檔路徑、CSV 檔案名和mini\_support。
  2. 檢查輸出目錄是否存在，不存在則創建。
  3. 將原始數據檔轉換為 CSV 格式。
  4. 讀取 CSV 檔，將數據轉換為交易清單。
  5. 使用 FP-Growth 演算法找出頻繁項目集。
  6. 將頻繁項目集和支援度寫入到結果檔。
  7. 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:
  1. 不需要生成候選項目集，從而減少了計算量。
  2. 使用 FP-Tree 結構可以更有效地存儲和計算頻繁項目集。
  3. 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

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

Run\_FPGrowth 運行FP\_Growth 演算法函式

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

打印所需結果

一張含有 螢幕擷取畫面, 文字, 字型 的圖片

自動產生的描述

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% |