HW1_Report

系級:智能系統 姓名:張宸瑋 學號:312581006

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

如何使用 apriori.py

```
Options:
  -h, --help            show this help message and exit
  -f INPUT, --inputFile=INPUT
                        filename containing csv
  -o OUTPUTFILEPATH, --outputFilePath=OUTPUTFILEPATH
                        the root of result file
  -s MINS, --minSupport=MINS
                        minimum support value
  -t STEP, --step=STEP  step 2 or step 3
```

命令行選項

```
python apriori.py --i dataset/A/A.data -s 0.002 -o dataset/A
```

使用範例

如何使用 fpgrowth.py(請使用這個做為 step IV 所需要使用的演算法)

```
Options:
  -h, --help            show this help message and exit
  -f INPUT, --inputFile=INPUT
                        filename containing csv
  -o OUTPUTFILEPATH, --outputFilePath=OUTPUTFILEPATH
                        the root of result file
  -s MINS, --minSupport=MINS
                        minimum support value
```

命令行選項

```
python .\fpgrowth.py --i ../dataset/A/A.data -s 0.002 -o dataset/fpgrowth_A
```

使用範例(要先進到 fpgrowth 資料夾)

如何使用 negFIN.py

```
Usage: negFIN.py [options]

Options:
  -h, --help            show this help message and exit
  -f INPUT, --inputFile=INPUT
                        filename containing csv
  -o OUTPUTFILEPATH, --outputFilePath=OUTPUTFILEPATH
                        the root of result file
  -s MINS, --minSupport=MINS
                        minimum support value
```

命令行選項
`python negFIN.py --i dataset/A/A.data -s 0.002 -o dataset/negFIN_A`
使用範例

(2) Step II

- Report on the mining algorithms/codes:

說明:

透過下面的計算時間報告我們可以看到，對於 support 的調整所帶來的影響，如果我們將 support 調高，那麼運算速度將會相對於低 support 來說，有明顯的提升，因為較低的 support 會造成更多的 frequent itemset，因此增加更多的計算與儲存開銷。

而我們也可以看到當資料集越來越大，因為 Apriori 算法需要掃描更多次的資料集，並且也會產生更多的候選項集時，因此會造成更多的時間開銷。

- Paste the screenshot of the computation time and the ratio of Task2 computation time compared to that of Task 1 in your report.

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/A/A.data -s 0.
002 -o dataset/A
Start Mining!!!
Count the computation time for task1 is 112.8015706539154s
Count the computation time for task2 is 153.6551365852356s
The ratio of computation time compared to that of Task 1 is 136.21719599690886%
End Mining!!!
資料夾 'dataset/A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/A' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/A/A.data -s 0.
005 -o dataset/A
Start Mining!!!
Count the computation time for task1 is 4.778224945068359s
Count the computation time for task2 is 4.891367673873901s
The ratio of computation time compared to that of Task 1 is 102.3678820086174%
End Mining!!!
資料夾 'dataset/A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/A' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/A/A.data -s 0.
01 -o dataset/A
Start Mining!!!
Count the computation time for task1 is 1.7848803997039795s
Count the computation time for task2 is 1.789872169494629s
The ratio of computation time compared to that of Task 1 is 100.279669707364%
End Mining!!!
資料夾 'dataset/A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/A' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

Configuration for dataset A: {0.2, 0.5, 1.0}

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/B/B.data -s 0.
0015 -o dataset/B
Start Mining!!!
Count the computation time for task1 is 5476.077075719833s
Count the computation time for task2 is 5477.545918226242s
The ratio of computation time compared to that of Task 1 is 100.02682289686757%
End Mining!!!
資料夾 'dataset/B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/B' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/B/B.data -s 0.
002 -o dataset/B
Start Mining!!!
Count the computation time for task1 is 2856.707341194153s
Count the computation time for task2 is 2857.118003129959s
The ratio of computation time compared to that of Task 1 is 100.01437535899755%
End Mining!!!
資料夾 'dataset/B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/B' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/B/B.data -s 0.
005 -o dataset/B
Start Mining!!!
Count the computation time for task1 is 926.0649173259735s
Count the computation time for task2 is 926.083856344223s
The ratio of computation time compared to that of Task 1 is 100.00204510697847%
End Mining!!!
資料夾 'dataset/B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/B' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

Configuration for datasets B: {0.15, 0.2, 0.5}

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/C/C.data -s 0.
01 -o dataset/C
Start Mining!!!
Count the computation time for task1 is 4432.618104696274s
Count the computation time for task2 is 4432.618104696274s
The ratio of computation time compared to that of Task 1 is 100.0%
End Mining!!!
資料夾 'dataset/C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/C' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/C/C.data -s 0.
02 -o dataset/C
Start Mining!!!
Count the computation time for task1 is 1563.7943727970123s
Count the computation time for task2 is 1563.7943727970123s
The ratio of computation time compared to that of Task 1 is 100.0%
End Mining!!!
資料夾 'dataset/C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/C' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python apriori.py --i dataset/C/C.data -s 0.
03 -o dataset/C
Start Mining!!!
Count the computation time for task1 is 572.7544865608215s
Count the computation time for task2 is 572.7544865608215s
The ratio of computation time compared to that of Task 1 is 100.0%
End Mining!!!
資料夾 'dataset/C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
Start write Task1 result2 file!!!
Write file end!!!
資料夾 'dataset/C' 已存在
Start write Task2 result1 file!!!
Write file end!!!
```

Configuration for datasets C: {1.0, 2.0, 3.0}

- Paste the screenshot of your code modification for Task 1 and Task 2 with comments and explain it.

Modification for Task 1

1. dataFromFile

```python
def dataFromFile(fname):
    """Function which reads from the file and yields a generator"""
    with open(fname, "r") as file_iter:
        for line in file_iter:
            line = line.strip(' ').rstrip("\n")  # Remove trailing comma
            record = frozenset(line.split(" ")[3:])
            yield record
```

說明：

這裡我更改了助教所提供的 Apriori 原始碼資料前處理的部分，因為透過 IBMGenerator 所生成的資料格式，每一列的前三個資料分別是 TID、TID、NITEMS，因為我們只需要 ITEMSET 的部分，因此只要取每一列第四個元素開始即可。

2. runApriori

```python
def runApriori(data_iter, minSupport):
    """
    run the apriori algorithm. data_iter is a record iterator
    Return both:
     - items (tuple, support)
    """
    start_time = time.time()
    itemSet, transactionList = getItemSetTransactionList(data_iter)
    resultFileTwoList = []
    freqSet = defaultdict(int)
    largeSet = dict()
    # Global dictionary which stores (key=n-itemSets,value=support)
    # which satisfy minSupport

    oneCSet= returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)
    resultFileTwoList.append(f"{1}\t{len(itemSet)}\t{len(oneCSet)}\n")
    currentLSet = oneCSet

    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet= returnItemsWithMinSupport(
            currentLSet, transactionList, minSupport, freqSet
        )

        resultFileTwoList.append(f"{k}\t{len(currentLSet)}\t{len(currentCSet)}\n")
        currentLSet = currentCSet
        k = k + 1
    def getSupport(item):
        """local function which Returns the support of an item"""
        return float(freqSet[item]) / len(transactionList)
    # print(freqSet)
    toRetItems = []
    for key, value in largeSet.items():
        toRetItems.extend([(tuple(item), getSupport(item)) for item in value])

    resultFileTwoList.insert(0, f"{len(toRetItems)}\n")

    task1_end_time = time.time()
```

圖(一)

```python
closedItemset = []
for key, value in largeSet.items():
    if(key != len(largeSet)):
        for sub_item in value:
            Closed = True
            for super_item in largeSet[key + 1]:
                if(set(sub_item).issubset(set(super_item)) and getSupport(sub_item) == getSupport(super_iten
                    Closed = False
                    break
            if (Closed):
                closedItemset.append((tuple(sub_item), getSupport(sub_item)))
    else:
        for super_item in largeSet[key]:
            closedItemset.append((tuple(super_item), getSupport(super_item)))

task2_end_time = time.time()
task1Time = task1_end_time - start_time
task2Time = task2_end_time - start_time
ratio = (task2Time / task1Time) * 100
print(f"Count the computation time for task1 is {task1Time}s")
print(f"Count the computation time for task2 is {task2Time}s")
print(f"The ratio of computation time compared to that of Task 1 is {ratio}%")

return toRetItems, resultFileTwoList, closedItemset
```

圖(二)

說明:

圖(一)針對 Task1 的部分，透過 start_time 以及 task1_end_time 計算所需的執行時間，而 toRetItem 則是用來記錄 Result file 1 所需要的資料內容，並且透過 resultFileTwoList 來記錄 Result file 2 所需要的資料內容。

圖(二)針對 Task2 的部分，透過 start_time 以及 task2_end_time 計算所需的執行時間，而迴圈的部分，主要是在尋找 Frequent Closed Itemset，一個 Itemset 如果是 Frequent Closed Itemset，那它的超集就不會跟它有相同的 support，而因為 largeSet 每層儲存不同的 K-Itemset，而對於任何一層的每一個 Itemset 來說，如果它是 Frequent Closed Itemset，那它的下一層的 Itemset 絕對不會出現跟它相同的 support，因此根據這個原理，計算出所有的 Frequent Closed Itemset，而 largeSet 的最後一層，因為絕對不會有比它大的 Itemset，所以都會是 Frequent Closed Itemset。

3. Write File

```python
def writeTask1File(options,items, resultFileTwoList):
    if not os.path.exists(options.outputFilePath):
    # 如果資料夾不存在，則使用os.makedirs創建它
        os.makedirs(options.outputFilePath)
        print(f"資料夾 '{options.outputFilePath}' 已創建")
    else:
        print(f"資料夾 '{options.outputFilePath}' 已存在")

    input_string = options.input
    data_index = input_string.find(".data")
    sorted_items = sorted(items, key=lambda x: x[1], reverse=True)
    if data_index != -1:
    # 如果找到了 ".data"，提取它前面的英文字母
        dataset = input_string[data_index - 1]
    else:
        print("沒有找到 '.data'")

    print("\033[101;97m" +"Start write Task1 result1 file!!!" + "\033[0m")
    with open(os.path.join(options.outputFilePath, f"{options.step}_task1_dataset({dataset})_{options.minS}_result1.txt"), "w") as file:
        for i in sorted_items:
            item, support = i
            itemset = "{" + ",".join(item) + "}"
            line = f"{support*100:.1f}\t{itemset}\n"
            file.write(line)
    print("\033[47;30m"+"Write file end!!!"+ "\033[0m")


    print("\033[101;97m" +"Start write Task1 result2 file!!!" + "\033[0m")
    with open(os.path.join(options.outputFilePath, f"{options.step}_task1_dataset({dataset})_{options.minS}_result2.txt"), "w") as file:
        for i in resultFileTwoList:
            file.write(i)
    print("\033[47;30m"+"Write file end!!!"+ "\033[0m")
```

圖(一)

```python
def writeTask2File(options, closedItemsetList):
    if not os.path.exists(options.outputFilePath):
    # 如果資料夾不存在，則使用os.makedirs創建它
        os.makedirs(options.outputFilePath)
        print(f"資料夾 '{options.outputFilePath}' 已創建")
    else:
        print(f"資料夾 '{options.outputFilePath}' 已存在")
    sorted_items = sorted(closedItemsetList, key=lambda x: x[1], reverse=True)
    input_string = options.input
    data_index = input_string.find(".data")
    if data_index != -1:
    # 如果找到了 ".data"，提取它前面的英文字母
        dataset = input_string[data_index - 1]
    else:
        print("沒有找到 '.data'")

    print("\033[101;97m" +"Start write Task2 result1 file!!!" + "\033[0m")
    with open(os.path.join(options.outputFilePath, f"step2_task2_dataset({dataset})_{options.minS}_result1.txt"), "w") as file:
            file.write(f"{len(sorted_items)}\n")
            for i in sorted_items:
                itemSet, support = i
                itemset = "{" + ",".join(itemSet) + "}"
                line = f"{support*100:.1f}\t{itemset}\n"
                file.write(line)
    print("\033[47;30m"+"Write file end!!!"+ "\033[0m")
```

圖(二)

說明：

圖(一)針對 Task1 的部分，進行寫入檔案的步驟，而圖(二)針對 Task2 的部分，進行寫入檔案的步驟。

(3) Step III

- Descriptions of your mining algorithm

說明:

這次我使用了 mlxtend 所提供的 FP-growth 來作為我這次 step Ⅲ 使用的演算法，而在原始碼中，整個 FP-growth 的 Program flow 為

1. 建立頻繁模式樹（FP 樹）：
   - 掃描資料集，統計每個項目的頻度。
   - 根據頻度對項目排序，將資料集轉換為 FP 樹結構。FP 樹是一種緊湊的樹狀結構，用於表示頻繁項集之間的層次關係。

```python
def setup_fptree(df, min_support):
    num_itemsets = len(df.index)  # number of itemsets in the database

    is_sparse = False
    if hasattr(df, "sparse"):
        # DataFrame with SparseArray (pandas >= 0.24)
        if df.size == 0:
            itemsets = df.values
        else:
            itemsets = df.sparse.to_coo().tocsr()
            is_sparse = True
    else:
        # dense DataFrame
        itemsets = df.values

    item_support = np.array(np.sum(itemsets, axis=0) / float(num_itemsets))
    item_support = item_support.reshape(-1)

    items = np.nonzero(item_support >= min_support)[0]

    # Define ordering on items for inserting into FPTree
    indices = item_support[items].argsort()
    rank = {item: i for i, item in enumerate(items[indices])}

    if is_sparse:
        itemsets.eliminate_zeros()

    tree = FPTree(rank)
    for i in range(num_itemsets):
        if is_sparse:
            nonnull = itemsets.indices[itemsets.indptr[i] : itemsets.indptr[i + 1]]
        else:
            nonnull = np.where(itemsets[i, :])[0]
        itemset = [item for item in nonnull if item in rank]
        itemset.sort(key=rank.get, reverse=True)
        tree.insert_itemset(itemset)

    return tree, rank
```

2. 建立條件模式基：
   - 對於每個頻繁項集，建立一個條件模式基。條件模式基是指包含特定頻繁項集的所有路徑。
3. 遞歸挖掘頻繁項集：
   - 對於每個頻繁項集，從 FP 樹中提取條件模式基。
   - 根據提取的條件模式基，建立一個新的 FP 樹，然後遞歸挖掘頻繁項集。
   - 這個遞歸過程持續進行，直到無法生成更多頻繁項集。

```python
def fpg_step(tree, minsup, colnames, max_len, verbose):
    """
    Performs a recursive step of the fpgrowth algorithm.

    Parameters
    ----------
    tree : FPTree
    minsup : int

    Yields
    ------
    lists of strings
        Set of items that has occurred in minsup itemsets.
    """
    count = 0
    items = tree.nodes.keys()
    if tree.is_path():
        # If the tree is a path, we can combinatorally generate all
        # remaining itemsets without generating additional conditional trees
        size_remain = len(items) + 1
        if max_len:
            size_remain = max_len - len(tree.cond_items) + 1
        for i in range(1, size_remain):
            for itemset in itertools.combinations(items, i):
                count += 1
                support = min([tree.nodes[i][0].count for i in itemset])
                yield support, tree.cond_items + list(itemset)
    elif not max_len or max_len > len(tree.cond_items):
        for item in items:
            count += 1
            support = sum([node.count for node in tree.nodes[item]])
            yield support, tree.cond_items + [item]

    if verbose:
        tree.print_status(count, colnames)

    # Generate conditional trees to generate frequent itemsets one item larger
    if not tree.is_path() and (not max_len or max_len > len(tree.cond_items)):
        for item in items:
            cond_tree = tree.conditional_tree(item, minsup)
            for sup, iset in fpg_step(cond_tree, minsup, colnames, max_len, verbose):
                yield sup, iset
```

4. 合併頻繁項集：
   - 將生成的所有頻繁項集合併為最終的頻繁項集列表。

```python
def generate_itemsets(generator, num_itemsets, colname_map):
    itemsets = []
    supports = []
    for sup, iset in generator:
        itemsets.append(frozenset(iset))
        supports.append(sup / num_itemsets)

    res_df = pd.DataFrame({"support": supports, "itemsets": itemsets})

    if colname_map is not None:
        res_df["itemsets"] = res_df["itemsets"].apply(
            lambda x: frozenset([colname_map[i] for i in x])
        )

    return res_df
```

Anything you want to share:

而在這次的實驗中，我也參考了收錄在 sciencedirect 期刊的 negFIN: An efficient algorithm for fast mining frequent itemsets，而這篇文章主要是提出一個高效率的數據結構，來幫助我們找到 frequent itemset，它參考了之前一些透過前綴樹資料結構的方式，(1)Node-list、(2)N-list、(3)Nodeset、(4)DiffNodeset，儘管這些方式在大部分的表現表現優異，但是都存在這一些問題，像是占用過多的記憶體，抑或是對於特定的 dataset 會有執行時間過長的問題，因此作者提出了 NegNodeset，透過位圖編碼的方式，以及

bitwise 的操作，來提供更高效的運算。

而程式碼主要的 Program flow 可以分為下面幾個部分

1. 初始化 F 為空集。
2. 呼叫構建 BMC-tree（DB，min-support）函數，以構建 BMC-tree 並找到 L1，L1 包含頻繁 1 項集。
3. 添加 L1 中的項集到 F 中。
4. 對於 BMC-tree 中的每個節點 N，按照任意順序遍歷 BMC-tree。
5. 將 N 的信息添加到與項 N.item-name 相關的節點集 Nodeset 中。

```
def __generate_NodeSets_of_1_itemsets(self):
    """
    Generate the BMCtree.
    During generation, insert each node to the appropriate NodeSet.
    """
```

6. 創建根節點 root。
7. 將根節點的層級 root.level 設置為 0（根節點位於層級 0）。 8. 初始化根節點的子節點列表 root.children-list 為空。
9. 初始化根節點的項名稱 root.item-name 為空。
10. 初始化根節點的項集 root.itemset 為空。

```
def __create_root_of_frequent_itemset_tree(self):
    """
    Create the root of frequent itemset tree and
    level 1 of frequent itemset tree and.
    This tree is not explicitly built.
    It represents the search space which is traversed in depth-first order.
    """
```

11. 對於 L1 中的每個項 i：
    - 創建一個名為 child_i 的節點。
    - 將 child_i 的層級 child_i.level 設置為 root.level + 1。
    - 將 child_i 的項名稱 child_i.item-name 設置為 i。
    - 將 child_i 的項集 child_i.itemset 設置為{i}。
    - 將 child_i 添加到根節點的 children-list 中。
    - 呼叫構建頻繁項集樹的函數 constructing_frequent_itemset_tree（child_i，∅）

```
for childIndex in range(num_of_children):
    child = root.children[0]
    itemset_buffer[itemset_length] = child.item
    del root.children[0] # We delete this node since it is not used anymore.
    # Call this recursive method to traverse the search space in a depth-first order.
    self.__construct_frequent_itemset_tree(child, itemset_buffer, itemset_length + 1, root.children,
                                           FIS_parent_buffer, FIS_parent_length)
```

12. 返回根節點 root。

Reference:

- [negFIN: An efficient algorithm for fast mining frequent itemsets (philippe-fournier-viger.com)]
- [mlxtend/mlxtend/frequent_patterns/fpgrowth.py at master · rasbt/mlxtend (github.com)]
- [【精选】Chapter 12 使用 FP-growth 算法来高效发现频繁项集_fpgrowth 算法算频繁项集_DB 架构的博客-CSDN 博客]

- Differences/Improvements in your algorithm

```python
def dataFromFile(fname):
    """Function which reads from the file and yields a generator"""
    result = []
    with open(fname, "r") as file_iter:
        for line in file_iter:
            line = line.strip(' ').rstrip("\n")  # Remove trailing comma
            record = (line.split(" ")[3:])
            result.append(record)
    return result
```

```python
dataset = dataFromFile(options.input)
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

這裡我更改了原始代碼中，資料前處理的部分，讓它可以符合原始碼中所需要的 dataframe 資料格式。

說明：

以上我使用的兩個改進算法，跟 Apriori 相比最大的差異就是不需要生成候選項集，因此並不需要頻繁的掃描資料集以及生成候選項集，它們透過樹狀結構來獲得 frequent itemsets，大幅的提升了運算效率。

- Computation time

FP-growth(非 **Candidate-based**)

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/A/A.data -s 0.002 -o fpgrowth_A
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 0.371265172958374s
資料夾 'fpgrowth_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/A/A.data -s 0.005 -o fpgrowth_A
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 0.07997488975524902s
資料夾 'fpgrowth_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/A/A.data -s 0.01 -o fpgrowth_A
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 0.05282306671142578s
資料夾 'fpgrowth_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
```

```
Configuration for dataset A Minsup : 0.2
The performance (on efficiency) improvements is 99.67198581560284%
Configuration for dataset A Minsup : 0.5
The performance (on efficiency) improvements is 98.34381551362684%
Configuration for dataset A Minsup : 1.0
The performance (on efficiency) improvements is 97.02247191011236%
```

Configuration for dataset A: {0.2, 0.5, 1.0}

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/B/B.data -s 0.0015 -o fpgrowth_B
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 7.999716281890869s
資料夾 'fpgrowth_B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/B/B.data -s 0.002 -o fpgrowth_B
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 6.5565104484558105s
資料夾 'fpgrowth_B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/B/B.data -s 0.005 -o fpgrowth_B
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 5.400234699249268s
資料夾 'fpgrowth_B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
```

```
Configuration for dataset B Minsup : 0.15
The performance (on efficiency) improvements is 99.8539282774877%
Configuration for dataset B Minsup : 0.2
The performance (on efficiency) improvements is 99.7705049905363%
Configuration for dataset B Minsup : 0.5
The performance (on efficiency) improvements is 99.41686233275153%
```

Configuration for datasets B: {0.15, 0.2, 0.5}

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/C/C.data -s 0.01 -o fpgrowth_C
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 54.389278411865234s
資料夾 'fpgrowth_C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/C/C.data -s 0.02 -o fpgrowth_C
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 38.26440238952637s
資料夾 'fpgrowth_C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\fpgrowth.py --i ../dataset/C/C.data -s 0.03 -o fpgrowth_C
Start Mining!!!
End Mining!!!
Count the computation time for task1 is 23.058758974075317s
資料夾 'fpgrowth_C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
```

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\
Configuration for dataset C Minsup : 1.0
The performance (on efficiency) improvements is 98.77298246769742%
Configuration for dataset C Minsup : 2.0
The performance (on efficiency) improvements is 97.55313039952833%
Configuration for dataset C Minsup : 3.0
The performance (on efficiency) improvements is 95.97418787123267%
```

Configuration for datasets C: {1.0, 2.0, 3.0}

negFIN(非 **Candidate-based**)

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/A/A.data -s 0.0
02 -o dataset/negFIN_A
Start Mining!!!
資料夾 'dataset/negFIN_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.002
 Number of transactions: 1000
 Number of frequent  itemsets: 31014
 Total time : 0.6908910274505615s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/A/A.data -s 0.0
05 -o dataset/negFIN_A
Start Mining!!!
資料夾 'dataset/negFIN_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.005
 Number of transactions: 1000
 Number of frequent  itemsets: 1781
 Total time : 0.19948840141296387s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/A/A.data -s 0.0
1 -o dataset/negFIN_A
Start Mining!!!
資料夾 'dataset/negFIN_A' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.01
 Number of transactions: 1000
 Number of frequent  itemsets: 353
 Total time : 0.13908934593200684s
==============
```

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\cal_time.py
Configuration for dataset A Minsup : 0.2
The performance (on efficiency) improvements is 99.38829787234043%
Configuration for dataset A Minsup : 0.5
The performance (on efficiency) improvements is 96.01677148846959%
Configuration for dataset A Minsup : 1.0
The performance (on efficiency) improvements is 92.69662921348313%
```

Configuration for dataset A: {0.2, 0.5, 1.0}

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/B/B.data -s 0.0
015 -o dataset/negFIN_B
Start Mining!!!
資料夾 'dataset/negFIN_B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.0015
 Number of transactions: 100000
 Number of frequent  itemsets: 8178
 Total time : 56.7912073135376s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/B/B.data -s 0.0
02 -o dataset/negFIN_B
Start Mining!!!
資料夾 'dataset/negFIN_B' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.002
 Number of transactions: 100000
 Number of frequent  itemsets: 4336
 Total time : 37.04376983642578s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/B/B.data -s 0.0
05 -o dataset/negFIN_B
Start Mining!!!
資料夾 'dataset/negFIN_B' 已創建
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.005
 Number of transactions: 100000
 Number of frequent  itemsets: 661
 Total time : 23.616334915161133s
==============
```

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\cal_time.py
Configuration for dataset B Minsup : 0.15
The performance (on efficiency) improvements is 98.96292546653379%
Configuration for dataset B Minsup : 0.2
The performance (on efficiency) improvements is 98.703297188798602%
Configuration for dataset B Minsup : 0.5
The performance (on efficiency) improvements is 97.66572151569575%
```

Configuration for datasets B: {0.15, 0.2, 0.5}

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/C/C.data -s 0.0
1 -o dataset/negFIN_C
Start Mining!!!
資料夾 'dataset/negFIN_C' 已創建
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.01
 Number of transactions: 1000000
 Number of frequent  itemsets: 325
 Total time : 601.7294237613678s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/C/C.data -s 0.0
2 -o dataset/negFIN_C
Start Mining!!!
資料夾 'dataset/negFIN_C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.02
 Number of transactions: 1000000
 Number of frequent  itemsets: 189
 Total time : 277.8699345588684s
==============
```

```
C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python>python negFIN.py --i dataset/C/C.data -s 0.0
3 -o dataset/negFIN_C
Start Mining!!!
資料夾 'dataset/negFIN_C' 已存在
Start write Task1 result1 file!!!
Write file end!!!
End Mining!!!
=====negFIN - STATS=====
 Minsup = 0.03
 Number of transactions: 1000000
 Number of frequent  itemsets: 108
 Total time : 92.94161868095398s
==============
```

```
PS C:\Users\Steven\Desktop\課程資料\交大\2023-1\Data Mining\hw1\Apriori_python\fpgrowth> python .\cal_time.py
Configuration for dataset C Minsup : 1.0
The performance (on efficiency) improvements is 86.42497503732557%
Configuration for dataset C Minsup : 2.0
The performance (on efficiency) improvements is 82.23109949264418%
Configuration for dataset C Minsup : 3.0
The performance (on efficiency) improvements is 83.77296361090451%
```

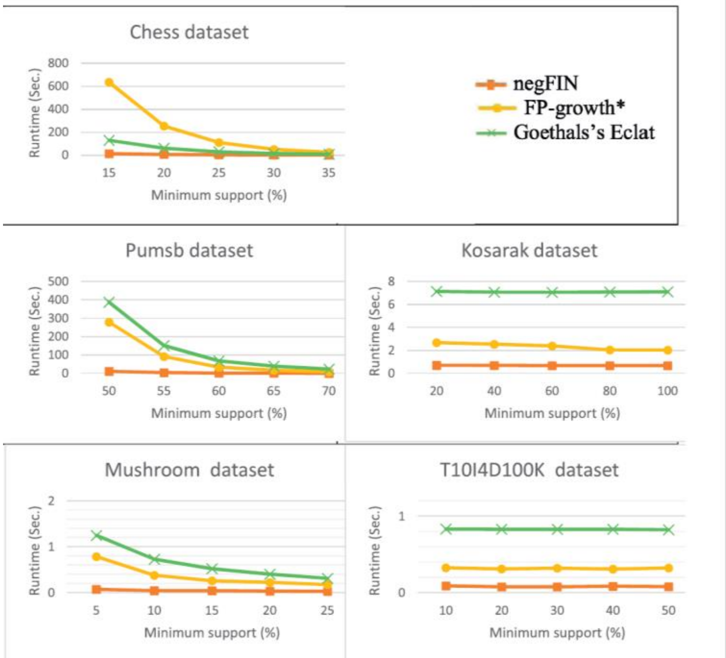
Configuration for datasets C: {1.0, 2.0, 3.0}

- Discuss the **scalability** of your algorithm in terms of the size of dataset (i.e., the rate of change on computing time under different data size; the largest dataset size the algorithm can handle, etc).

討論:

從上面的實驗結果來看，我們可以看到，這兩種改進算法，相對於Apriori，其計算速度都有明顯的提升，而這裡我多實驗了negFIN算法，是因為在查詢資料時，查詢到這篇paper，而我在這篇paper中看到在它的實驗結果中(圖一所示)，它的表現相對於FP-growth，有明顯的提升，因此希望能夠透過這個算法，來嘗試出比FP-growth更快的結果，但是從實驗結果中，我們可以看到，在資料集越來越大的時候，FP-growth跟negFIN的差異越來越大，這裡我探討了可能的結

果，我認為其原因可能是因為在mlxtend所提供的FP-growth原始碼，它是基於pandas這樣的資料結構來去實現演算法，這種資料結構在處理大數據的時候性能十分優異，相較於dict、list(negFIN所使用)這些資料結構來說，因此希望之後有機會，能夠去探討，是否能夠透過pandas來去實現negFIN演算法，但整體而言，我們可以看到，這兩種演算法在對於大數據集的處理速度來說，都是明顯優於Apriori的。



圖(一)