



國立成功大學 資工所

Data Mining

資料探勘

Project 1

Association Analysis

課程教授：高宏宇

學生：葉芯妤

學號：P96074147



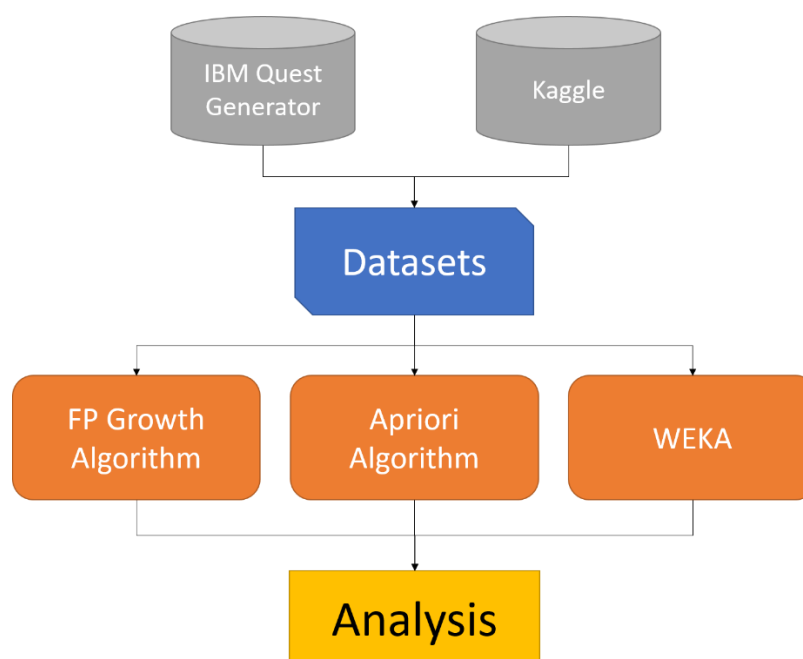
目錄

一、專案內容簡介	3
二、程式實作	4
三、WEKA 分析	6
四、結論	9

一、 簡介

A. 總覽

利用 IBM 資料產生器產生的資料集以及從 Kaggle 網站上找到可用於關聯分析的資料集，實作 Apriori 演算法以及 FP-Growth 演算法的程式碼並使用 WEKA 資料探勘工具來分析兩項資料集的結果，專案架構如圖一所示。



圖一：專案架構

B. 資料

本專案利用 IBM 資料產生器設定不同變因的參數作為輸入，並產生多種測試資料集，對於不同資料集能分析的東西也相對較多。更改 IBM-Quest-Data-Generator.exe 參數產生資料，使用 `lit -ntrans 10 -tlen 50 -nitems 0.1` 指令，得到 dataset 後，將檔案轉為 csv 檔，以利分析，資料內容如下圖二所示。

而在 Kaggle 網站上找尋與關聯分析相關的資料集，本專案使用國際貿易公約 (CITES) 野生動植物交易資料庫，分析野生動植物交易的類型以及用途之間的關聯。

Kaggle 資料來源:

<https://www.kaggle.com/cites/cites-wildlife-trade-database>

此資料為 2016 年和 2017 年進行的 CITES 保護物種的所有合法國際貿易，資料總數共 671621 筆，本專案從中隨機取樣 5000 筆資料進行分析，資料內容如下圖三所示，而表格一為每一欄資料所代表的意思，其中有兩項為字母代碼，其涵義需參閱資料所附的 CITES 文檔。

CITES 文檔網址：

https://trade.cites.org/cites_trade_guidelines/en-CITES_Trade_Database_Guide.pdf

	A	B
1	1	0
2	1	4
3	1	5
4	1	6
5	1	7
6	1	8
7	1	9
8	1	10
9	1	11
10	1	12
11	1	14
12	1	15
13	1	17
14	1	18
15	1	21
16	1	23
17	1	25
18	1	26
19	1	29
20	1	34
21	1	35
22	1	36
23	1	38
24	1	39
25	1	40
26	1	41
27	1	42
28	1	43
29	1	45

圖二 IBM 產生的資料內容

	A	B	C	D	E	F
1	Genus	Importer	Exporter	Term	Purpose	Source
2	Aquila	TR	NL	bodies	T	C
3	Aquila	XV	RS	bodies	Q	O
4	Haliaeetus	BE	NO	feathers	S	W
5	Haliaeetus	BE	NO	specimens	S	W
6	Haliaeetus	DK	IS	specimens	S	W
7	Haliaeetus	XV	RS	bodies	Q	O
8	Harpia	BR	FR	feathers	S	C
9	Harpia	BR	FR	feathers	S	U
10	Harpia	BR	FR	feathers	S	W
11	Acipenser	CH	DE	live	T	C
12	Acipenser	US	IR	caviar	P	I
13	Ailurus	AU	NZ	live	Z	C
14	Ailurus	CA	US	live	Z	F
15	Ailurus	IL	DE	live	Z	C
16	Ailurus	JP	US	live	B	C
17	Ailurus	JP	US	live	Z	C
18	Ailurus	KP	CN	bodies	E	U
19	Ailurus	KR	CN	specimens	E	C
20	Ailurus	KR	JP	live	Z	C
21	Ailurus	US	CA	live	Z	C
22	Alligator	KP	CN	bodies	E	U
23	Melanosuc	US	DK	live	Z	F
24	Anas	CA	US	feathers	S	I
25	Anas	MC	FR	live	Z	I

圖三 Kaggle 資料內容

表一 Kaggle 資料欄位說明

Genus	學名	代表物種的分類
Importer	進口商	進口該物種的國家
Exporter	出口商	出口該物種的國家
Term	型態	出口或進口的型態。活體、標本、獸皮等。
Purpose	目的	物種出口或進口的目的。為字母代碼，其涵義需參閱 CITES 文檔。
Source	來源	描述動植物如何進入市場：繁殖，捕獲野生動物，標本等。為字母代碼，其涵義需參閱 CITES 文檔。

二、程式實作

A. 環境

作業系統：Window 10

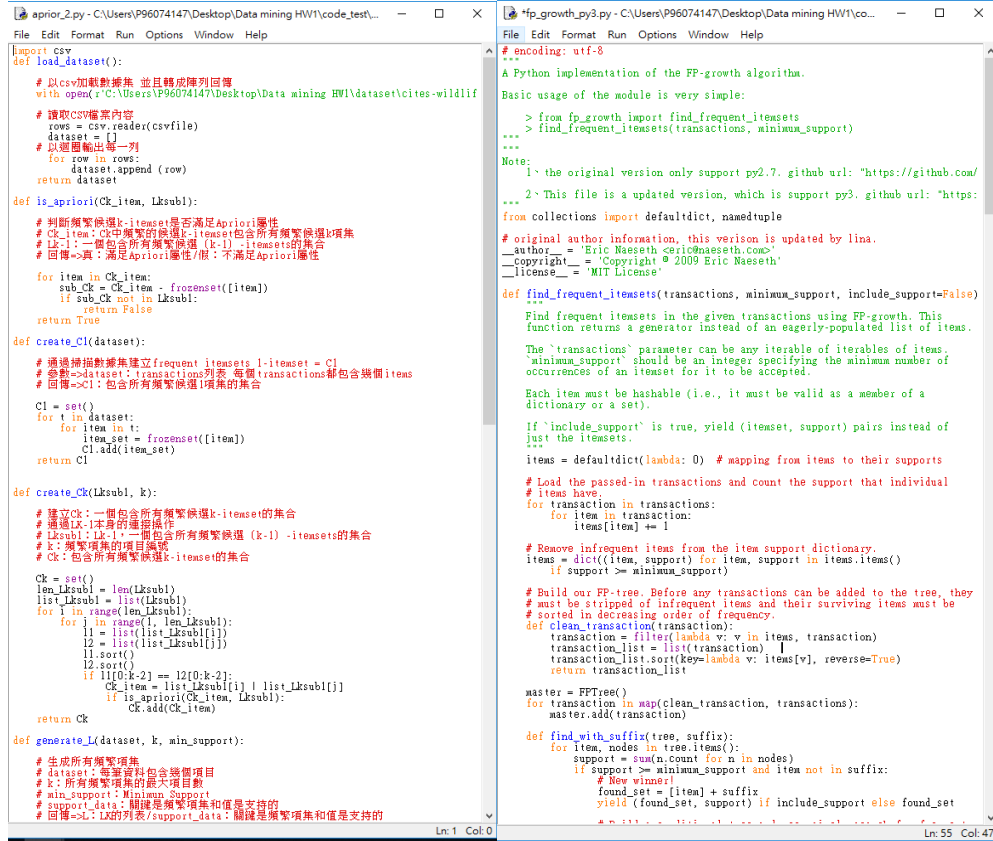
程式語言：Python3.6

額外工具：WEKA、IBM-Quest-Data-Generator、Excel、Kaggle

B. 程式碼

本專案以 Python 撰寫 Apriori 演算法以及 FP-Growth 演算法的程式碼，對 IBM 及 Kaggle 兩項資料進行分析。由於個人程式能力不足，因此 FPgrowth 的程式碼是在 <https://github.com/enaeseth/python-fp-growth> 的基礎上做更改。

下圖四為 Apriori 程式碼，紅色中文字為對程式碼所作的註解。而 FPgrowth 程式碼分為兩部分，一部分為 FPgrowth 的規則，即為上述提及參考的程式碼，如下圖五所示，另一部分為個人撰寫的應用該規則，將 csv 檔案匯入進行分析，並顯示出結果的程式碼，如下圖六所示，紅色中文字為對程式碼所作的註解。



```
import csv
def load_dataset():
    # 以csv加載數據集 並且轉成陣列回傳
    with open(r'C:\Users\P96074147\Desktop\Data mining HW1\dataset\cites-wildlif
    # 讀取CSV檔案內容
    rows = csv.reader(csvfile)
    dataset = []
    # 以迴圈輸出每一列
    for row in rows:
        dataset.append(row)
    return dataset

def is_apriori(Ck_item, Lksubl):
    # 判斷頻繁候選k-itemset是否滿足Apriori屬性
    # Ck_item: Ck中頻繁的候選k-itemset包含所有頻繁候選k項集
    # Lk-1: 一個包含所有頻繁候選 (k-1) - itemsets的集合
    # 回傳=>真: 滿足Apriori屬性/假: 不滿足Apriori屬性
    for item in Ck_item:
        sub_Ck = Ck_item - frozenset([item])
        if sub_Ck not in Lksubl:
            return False
    return True

def create_C1(dataset):
    # 通過掃描數據集建立 frequent itemsets 1-itemset = C1
    # 參數=>dataset: transactions列表 每個 transactions都包含幾個 items
    # 回傳=>C1: 包含所有頻繁候選1項集的集合
    C1 = set()
    for t in dataset:
        item_set = frozenset([item])
        C1.add(item_set)
    return C1

def create_Ck(Lksubl, k):
    # 建立Ck: 一個包含所有頻繁候選k-itemset的集合
    # 通過Lk-1本身的掃描操作
    # Lksubl: Lk-1: 一個包含所有頻繁候選 (k-1) - itemsets的集合
    # k: 頻繁項集的项目編號
    # Ck: 包含所有頻繁候選k-itemset的集合
    Ck = set()
    len_Lksubl = len(Lksubl)
    list_Lksubl = list(Lksubl)
    for i in range(1, len_Lksubl):
        l1 = list(list_Lksubl[i])
        l2 = list(list_Lksubl[i])
        l1.sort()
        l2.sort()
        if l1[0:k-2] == l2[0:k-2]:
            Ck_item = list_Lksubl[i] + list_Lksubl[j]
            if is_apriori(Ck_item, Lksubl):
                Ck.add(Ck_item)
    return Ck

def generate_L(dataset, k, min_support):
    # 生成所有頻繁項集
    # dataset: 每列資料包含幾個項目
    # k: 所有頻繁項集的最大項目數
    # min_support: Minimum_Support
    # support_data: 回傳每個頻繁項集和值是否支持的
    # 回傳=>L: Lx的列表/support_data: 關鍵是頻繁項集和值是否支持的
    L = [C1]
    support_data = {}
    for itemset in C1:
        support_data[itemset] = 0
    for transaction in dataset:
        for itemset in C1:
            if itemset.issubset(transaction):
                support_data[itemset] += 1
    L = [itemset for itemset, support in support_data.items() if support >= min_support]
    return L, support_data
```

```
# encoding: utf-8
"""
A Python implementation of the FP-growth algorithm.
Basic usage of the module is very simple:
> from fp_growth import find_frequent_itemsets
> find_frequent_itemsets(transactions, minimum_support)
"""
Note:
1. the original version only support py2.7. github url: "https://github.com/
2. This file is a updated version, which is support py3. github url: "https:
from collections import defaultdict, namedtuple

# original author information, this version is updated by lina.
__author__ = "Eric Naeemeth eric@naeemeth.com"
__copyright__ = "Copyright © 2009 Eric Naeemeth"
__license__ = "MIT License"

def find_frequent_itemsets(transactions, minimum_support, include_support=False):
    """
    Find frequent itemsets in the given transactions using FP-growth. This
    function returns a generator instead of an eagerly-populated list of items.

    The "transactions" parameter can be any iterable of iterables of items.
    minimum_support should be an integer specifying the minimum number of
    occurrences of an itemset for it to be accepted.

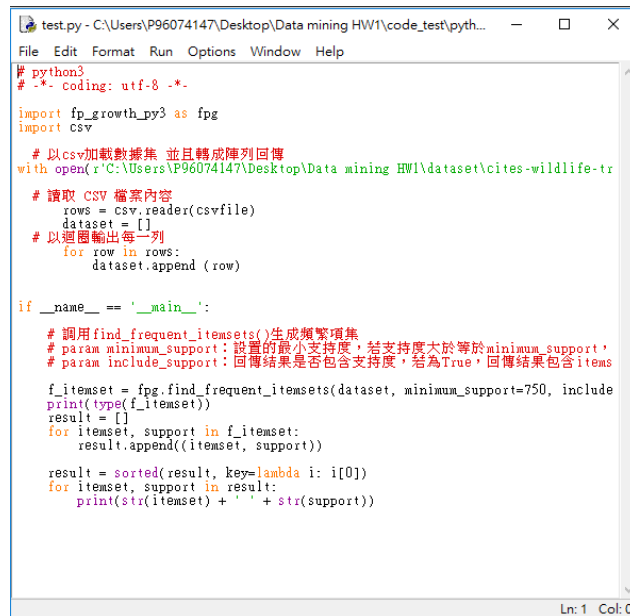
    Each item must be hashable (i.e., it must be valid as a member of a
    dictionary or a set).

    If "include_support" is true, yield (itemset, support) pairs instead of
    just the itemsets.

    items = defaultdict(lambda: 0) # mapping from items to their supports
    # Load the passed-in transactions and count the support that individual
    # items have.
    for transaction in transactions:
        for item in transaction:
            items[item] += 1
    # Remove infrequent items from the item support dictionary.
    items = dict((item, support) for item, support in items.items()
                if support >= minimum_support)
    # Build our FP-tree. Before any transactions can be added to the tree, they
    # must be stripped of infrequent items and their surviving items must be
    # sorted in decreasing order of frequency.
    def clean_transaction(transaction):
        transaction = filter(lambda v: v in items, transaction)
        transaction_list = list(transaction)
        transaction_list.sort(key=lambda v: items[v], reverse=True)
        return transaction_list
    master = FFTree()
    for transaction in map(clean_transaction, transactions):
        master.add(transaction)
    def find_with_suffix(tree, suffix):
        for item, nodes in tree.items():
            support = sum(n.count for n in nodes)
            if support >= minimum_support and item not in suffix:
                # New winner!
                found_set = [item] + suffix
                yield (found_set, support) if include_support else found_set
    found_set, support = find_with_suffix(master, [])
    return found_set, support
```

圖四為 Apriori 程式碼

圖五 FPgrowth 參考程式碼



```
# python3
# -*- coding: utf-8 -*-

import fp_growth_py3 as fpg
import csv

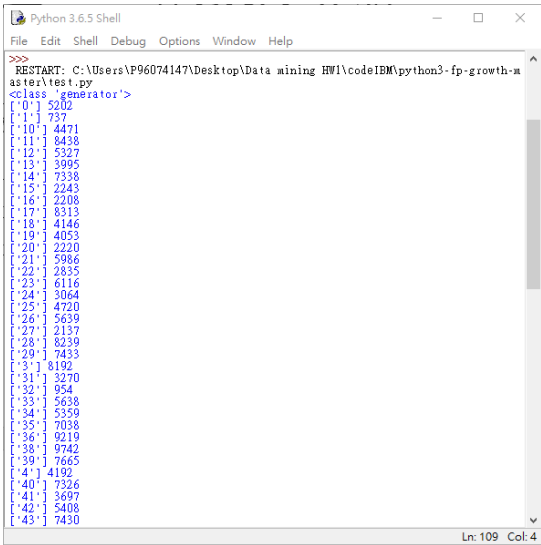
# 以csv加載數據集 並且轉成陣列回傳
with open(r'C:\Users\P96074147\Desktop\Data mining HW1\dataset\cites-wildlife-tr
# 讀取 CSV 檔案內容
rows = csv.reader(csvfile)
dataset = []
# 以迴圈輸出每一列
for row in rows:
    dataset.append(row)

if __name__ == '__main__':
    # 調用find_frequent_itemsets()生成頻繁項集
    # param minimum_support: 設置的最小支持度, 結支持度大於等於minimum_support'
    # param include_support: 回傳結果是否包含支持度, 若為True, 回傳結果包含items
    f_itemset = fpg.find_frequent_itemsets(dataset, minimum_support=750, include
    print(type(f_itemset))
    result = []
    for itemset, support in f_itemset:
        result.append((itemset, support))
    result = sorted(result, key=lambda i: i[0])
    for itemset, support in result:
        print(str(itemset) + ' ' + str(support))
```

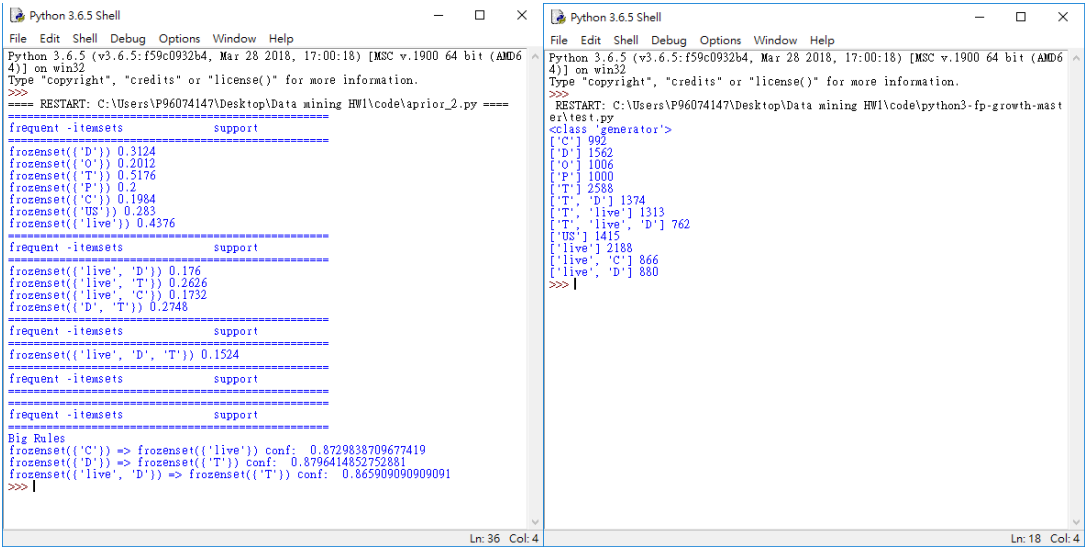
圖六 FPgrowth 實際測試程式碼

由於 IBM 所產生資料數量相當大，在利用 Apriori 演算法運算時，電腦速度變得

相當緩慢，且無法運算出規則，耗費相當多時間，由於 FPgrowth 演算法運算比 Apriori 快速許多，故採用 FPgrowth 演算法對 IBM 資料進行分析比較，程式執行結果如下圖七所示。



圖七 FPgrowth 程式對 IBM 執行結果



圖八 Apriori 程式對 Kaggle 執行結果 圖九 FPgrowth 程式對 Kaggle 執行結果

三、 WEKA 分析

1. IBM

交易筆數	平均每筆交易 物品量	物品種類 數量	Minsup	FP Growth 找到 的規則量	執行時間
1	50	0.1	500	1426	69457

1	50	0.1	600	1687	2943
1	50	0.1	700	2106	944
1	50	0.1	800	1558	507
1	50	0.1	900	230	313

一開始將 Minsup 設定為 500 測試找尋的規則數量多寡與 Minsup 值的關係，由於結果冗長且每個物品項目數量出現的次數幾乎都超過 500，於是將 Minsup 調高以 100 為遞增找尋各個關係。

出乎意料的，在門檻值調動後，一開始找尋到的規則數反而是增加的，而在門檻值為 700 時，達到 2000 以上後，又逐次降低。可能原因是由於門檻值調動，使物品相關的規則數找到更多。如下圖所示。

```

1 31'and'48'and'78'and'83'and'73'and'29'and'0'and'9'and'69'and'47=510
2 86'and'9'and'71=549
3 70'and'73'and'11'and'38'and'71'and'63'and'47=541
4 51'and'8'and'48'and'78'and'11'and'38'and'71'and'63'and'87=510
5 51'and'7'and'73'and'47=517
6 31'and'29'and'0'and'9'and'11'and'38'and'71'and'63'and'87'and'47'and'36=589
7 28'and'72'and'73'and'9'and'87'and'47=563
8 8'and'78'and'83'and'73'and'71'and'63'and'3=608
9 62'and'28'and'9'and'11'and'38'and'71'and'63'and'3'and'87'and'47=574
10 89'and'83'and'29'and'38'and'71'and'63'and'3'and'87'and'36=503
11 31'and'48'and'78'and'73'and'0'and'9'and'38'and'71'and'63'and'3'and'87'and'36=504
12 28'and'78'and'83'and'73'and'38'and'71'and'69'and'3'and'87'and'36=568
13 86'and'9'and'63=551
14 51'and'29'and'38'and'71'and'63'and'3'and'87'and'47=631
15 70'and'73'and'11'and'38'and'71'and'63'and'36=541
16 78'and'83'and'73'and'29'and'0'and'9'and'11'and'38'and'71'and'69'and'63'and'47=660
17 8'and'29'and'0'and'9'and'38'and'71'and'69'and'36=701
18 49'and'48'and'78'and'73'and'29'and'0'and'9'and'11'and'38'and'71'and'63'and'3'and'87'and'47'and'36=514
19 31'and'83'and'73'and'29'and'0'and'9'and'11'and'38'and'71'and'69'and'63'and'87=551

```

圖 Minsup 為 500 時的結果

```

1 Rule:
2 {11'and'38'and'71'and'36=901
3 9'and'3'and'87'and'47=910
4 73'and'47'and'36=921
5 69'and'47=961
6 11'and'38'and'69=906
7 83'and'87=923
8 11'and'38'and'63=913
9 71'and'63'and'87'and'47=905
10 78'and'63=913
11 69'and'3'and'87'and'47'and'36=903
12 29'and'87'and'47=920
13 63'and'47=964
14 11=963
15 11'and'38'and'71'and'47=905
16 69'and'63'and'3'and'87'and'47'and'36=903
17 69'and'47'and'36=953
18 9'and'11=918
19 87'and'47'and'36=971

```

圖 Minsup 為 900 時的結果

i. Minsup 值低時，會有較長的子集

當 Minsup 為 500 時最長的一列(也就是包含最多物品數量的子集)，表示內容的物品項目任意數量皆會超過後面的數值，以 18 行為例，49 與 48 常出現的比例會高過 514。所以規則數量有可能會因為 Minsup 值的調整而有所變動。

ii. 門檻值的訂定

可以看到當 Minsup 值定為 900 時，仍有許多物品集規則是超過 900 的，

所以初訂定為 500 時執行時間為 60000 多秒，實際上是因為 Minsup 值極低，導致運算量大，且規則非常詳細。因此，本專案調整了物品數量以及 Minsup 值，更改後的資料如下表所示。

交易筆數	平均每筆交易物品量	物品種類數量	Minsup	FP Growth 找到的規則量	執行時間
1	10	0.1	100	125	203
1	30	0.1	100	2734	6010
1	30	0.1	300	782	952
1	10	0.1	100	125	203
1	30	0.1	100	2734	6010

從平均每筆交易物品量與 Minsup 可以看到有相當大的差異，在測試程式時，會發現運算速度與找到的規則量都會受到平均每筆交易物品量與 Minsup 值(出現次數)的變動而有所差異。

而當平均每筆交易物品量改變且 Minsup 值固定時，在每筆交易當中，擁有較多物品數量則能找到較多的規則，原因在於建立 FP Tree 時，每筆交易物品越多，樹的深度跟著增加，找到的 Pattern Base 也較長，能夠產生更多的 Frequent Pattern 去找到更多規則。

2. Kaggle

將 Kaggle 搜尋到的資料轉為 .arff 檔後，用 WEKA 開啟，將資料進行轉換之後，使用 Apriori 演算法以及 FPGrowth 演算法進行分析，下圖和圖為使用兩項演算法分析後得出的結果。

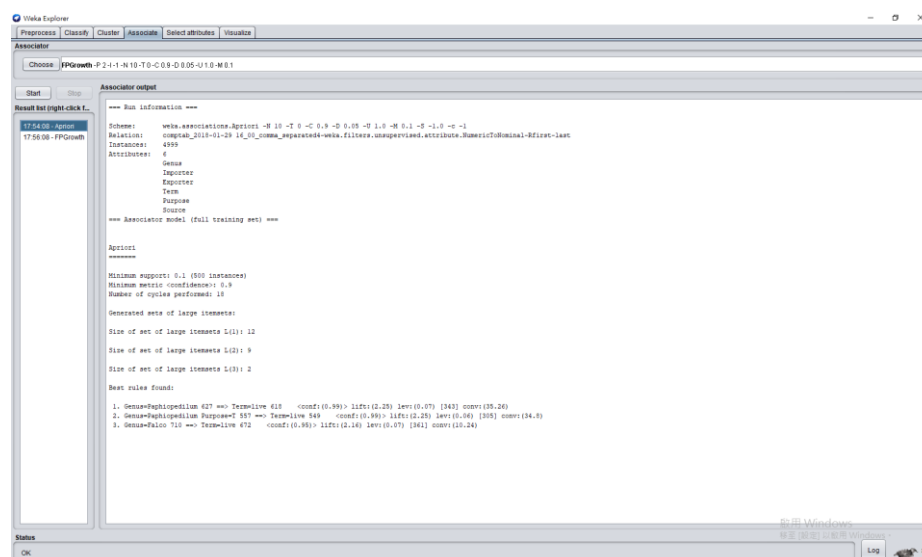


圖 使用 Apriori 演算法分析的結果

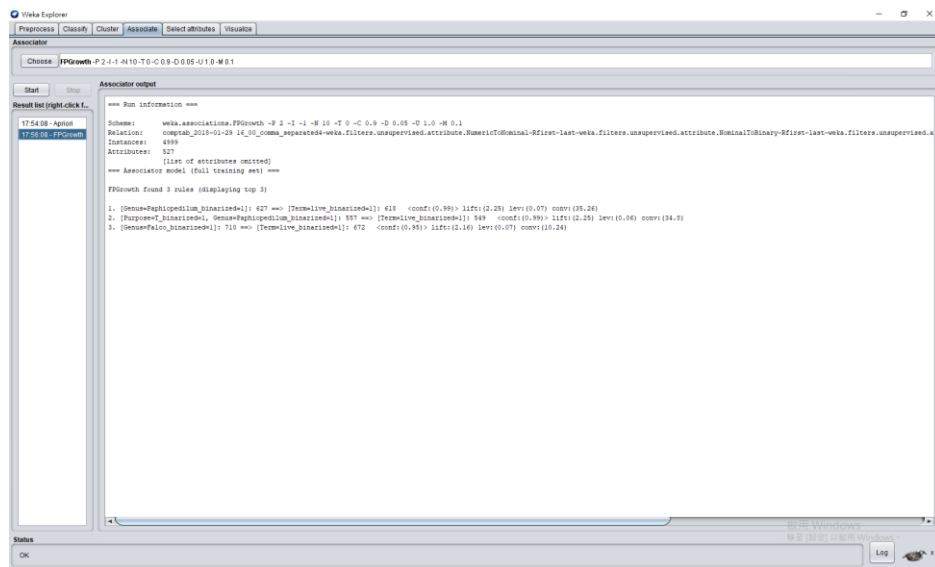


圖 使用 FPgrowth 演算法分析的結果

使用 Apriori 演算法以及 FPgrowth 演算法所分析出來的結果，兩者為一致的，從得出的三項規則可知，兜蘭花 (Paphiopedilum) 在交易時的型態大部分為活體 (live)，且用途為商業用途 (T 代表 Commercial) 居多，經過網路搜尋兜蘭花的資料，推測其商業用途應該是為各國舉辦花卉博覽會以及栽培所用，且此物種在資料中佔了將近一千筆，可見兜蘭的栽培技術對其保育研究及商業生產意義重大。

而鷹 (Falco) 交易的型態多半也是活體 (live)，對尚未取樣的資料進行查找後，發現出口與進口國大部分為中東國家 (出口國以阿拉伯聯合酋長國佔多數)，而用途大部分為個人用途 (P) 以及商業用途 (T)，由此資訊可以聯想到，由於中東國家有馴養獵鷹的傳統，獵鷹捕獲的獵物可以增加沙漠游牧民族的肉食量。如今這項活動還在繼續，且變成中東富人的娛樂，獵鷹大部分會在市場的特殊區域販售，因此獵鷹的買賣被歸類的資料中。

四、 結論

在進行實作的過程中，了解了資料量、Minsup 值對於運算時間、規則數、運算成效有極大的影響。撰寫程式碼與 WEKA 分析的結果一致，透過 IBM Quest Data Generator 產生數個模擬資料，同時用 Weka 的分析結果與撰寫的程式碼做測試，分析結果成效可能因為資料量大會有所差距，在此測試較大資料集的部分將 Minsup 值調高，以利快速找到規則並避免程式崩潰。

本次作業在程式部分耗費許多時間，由於個人程式能力不足，發現程式碼會將單項子集也列出為規則，此項小缺失雖不影響結果，但仍需仔細判別。第一次嘗試分析與結果比較，雖然資料不夠完善，但仍可分析出結果，希望未來能夠將此項技術更精進，妥善運用此方法分析其他資料集。