在交易資料庫中利用頻繁樣式來找出離異交易之研究

李建億
國立臺南大學數位學習科技學系
leeci@mail.nutn.edu.tw

林吟美
國立臺南大學數位學習科技學系
yinmei@mail.nutn.edu.tw

摘要

在一個資料庫裡，異常是指特性相異於其他資料的資料點。偵測出這些異常在許多應用中是很重要的。大部分傳統異常探勘的資料屬性都著重在數值屬性，但是現實生活中有很多類別屬性的資料，所以傳統的異常探勘方法就不適合使用。FindFPOF (frequent pattern outlier factor)演算法是利用頻繁項目集找出異常交易，但是 FindFPOF 會產生出許多頻繁樣式集。我們的目的是找出異常而不是在找出頻繁樣式，因此，本論文提出減少頻繁項目集(reduce frequent pattern set ; Rfp)來找出異常交易的方法，稱之為 RfpFPOF。實驗結果顯示本論文所提出的方法在類別資料庫中發現異常交易的效能優於 FindFPOF。

關鍵詞：異常探勘、關聯法則探勘、類別資料。

Abstract

An outlier in a database is an observation or a point that is considerably dissimilar to or inconsistent with the remainder of the data. Outlier detection is very import for many applications. Most existing methods are designed for numeric data. They will have problems with real-life applications that contain categorical data. In order to solve this problem, some scholars propose the FindFPOF (Frequent Pattern Outlier Factor) algorithm to detect outliers from discovered frequent patterns. However, FindFPOF generates too many frequent patterns. The purpose of this study is to identify outlier not to find frequent patterns. Therefore, in this study, we present an efficient method to detect outliers by reducing the number of frequent patterns, called the reduce frequent pattern FPOF (RfpFPOF) algorithm. The experimental results have shown that RfpFPOF outperformed the FindFPOF method on identifying outliers execution time.

Keywords: Outlier detection, Association rules, Categorical data

1. 前言

由於現今電腦軟硬體設備趨於便宜、儲存容量越來越大，導致資料累積的快速成長。然而在原始蒐集資料中，有用且有趣的資訊太少。因此，為了要找出這些有用的資訊，資料探勘(data mining)已成為近年來資料庫相關研究領域的熱門焦點。資料探勘的定義是指從大量的資料庫或資料倉儲中，萃取出先前未知並且可能是有用資訊的一個資訊發現過程，探勘出來資訊可提供組織做為決策的參考(Yu, Sheikholeslami, & Zang, 2002)。資料探勘依據核心技術與產生的知識型態，可分為關聯分析(association analysis)、分類與預測(classification and prediction)、叢集分析(cluster analysis)等三類(Chen, Han, & Yu, 1996 ; Han & Kamber, 2000)。關聯分析是發現關聯法則，這些關聯法則顯示，在給定資料庫中，屬性-值(attribute-value)經常一起出現。分類與預測探勘是資料分析的兩種型式，用來淬取出描述重要資料類別的模型或預測未來資料的趨勢，分類是用來預測資料物件的類別標籤，預測是建立連續數值函數模型。叢集分析是將資料物件分組成多個叢集，使得在同一個叢集中的物件之間具有較高的相似度，不同的叢集中物件差別會比較大。

在大型的資料庫中，資料探勘存在著許多問題，像是資料重複、屬性值不明確、資料不完整以及資料異常(outlier)(Breunig, Kriegel, Ng, & Sander, 2000)。所謂異常常被定義為該資料點相異於其他資料。這樣的資料點可提供系統異常行為有用的資訊(Aggarwal & Yu, 2005)。而異常偵測(outlier detection)是叢集演算法的子方法之一。從叢集演算法的觀點來看，異常是指沒有落在資料集群(cluster)裡的物件，因此，稱之為雜訊(noise) (Breunig et al., 2000)。異常偵測或異常探勘是指在一群資料裡辨識出異常的一個過程。異常偵測問題是資料探勘研究中，新興的研究議題。許多資料探勘演算法試著要將異常的影響減到最小，或者忽視這些異常。但是，這可能導致遺失重要隱藏的資訊，因為，對某個人所謂的雜訊，有可能是其他人有用的資訊(Knorr, Ng, & Tucakov, 2000)。換句話說，異常也許具有特殊的重要性。譬如，在詐欺偵測的情況下，異常也許就是詐欺活動(Han et al., 2000)。異常偵測技術可應用在信用卡欺騙、網路強健度分析、網路入侵偵測、財政應用和行銷上(Aggarwal et al., 2005)。因此，異常偵測和分析是一項有趣極重要的資料探勘議題。

在高維度的空間中，現存的技術嘗試利用距離來偵測出異常(He, Xu, & Deng, 2005)。但是在最近 Z. He 等人的研究結果中顯示，在高維度的空間中鄰近的概念未必是有用的(He et al., 2005)。由於維度的關係，在高維度的空間中以距離為基礎的異常

偵測方法並不適合用來偵測異常。傳統的異常偵測方法通常都著重在處理數值屬性的資料，然而真實世界中，資料大多都含有類別屬性，所以傳統的異常偵測方法就不適合在這些類別屬性資料中使用。而關聯法則探勘，除了可處理數值資料外，更有效處理類別屬性的資料，所以本篇論文就是要利用關聯法則探勘的特性來來找出異常。

在 2005 年 Z.He 等人提出了 Find frequent pattern outlier factor (FindFPOF)方法(He, Xu, Huang, & Deng, 2005)來偵測異常。首先，透過關聯法則探勘的 Apriori 方法產生出頻繁項目集，然後利用這些頻繁項目集來計算出異常的交易。但是，一般情況下，在 Apriori 的方法中，會產生過多的候選項目集(candidate itemsets)問題。而且當支持度門檻值(support threshold)越低時，所產生出的候選項目集的數量就會越多，但最後真正成為頻繁項目集的數量卻不多，以致於浪費了龐大的運算時間和記憶體空間。再則，異常偵測的目的是在找出異常的交易，而不是在找頻繁項目集。所以，本研究提出了 reduce frequent pattern FPOF (RfpFPOF)方法，來解決 FindFPOF 因為需要全部的頻繁項目集而造成整個方法執行速度緩慢的問題。

2. 文獻探討

2.1 異常探勘

異常資料的定義是存在一個資料，偏離一般的行為或資料模型，且不同於資料集裡其他資料的資料。異常資料的存在通常是使用者錯誤的輸入而導致量測或執行的錯誤，因此在分析的過程中通常會將異常給移除，以加強資料處理的正確性。相反的，異常的資料通常也會反映著特殊的訊息，例如：“詐欺”行為。異常探勘的應用範圍很廣，例如：信用卡異常使用、電信服務需求突然增加、行銷上高所得與低所得的消費行為模式分析、醫學上一些罕見疾病的分析(Otey, Ghoting, & Parthasarathy, 2004)。

異常探勘的定義：在 n 個點的資料集裡，找到 k 個點，在此 k 點中每一點，都和剩下的 $n-k$ 個點的性質不同，這 k 個點就叫做異常。異常探勘的重點在於順利探勘出所有異常資料。

異常探勘主要分為：統計基礎方法(statistics based method)、距離基礎方法(distance based method)、偏離基礎方法(deviation based method)、叢集基礎方法(clustering based method)、密度基礎方法(density based method)、子空間基礎方法(sub-space based method)、類神經網路基礎方法(neutral network based method)(Ramaswamy, Rastogi, & Kyuseok, 2000)，以下分別簡述異常探勘的主要方法。

(1) 統計基礎方法(statistics based method)：先假設檢驗的資料集具有某種資料分佈(data distribution)，然後使用不一致性檢驗(discordancy test)來偵測出異常。檢驗需要運用到統計的相關知識，如機率分佈、變異數分析等(Barnett & Lewis, 1994)。統計基礎方法的缺點是不一致性檢驗通常是針對單一屬性，然而異常探勘經常使用在多維的資料，並且統計基礎的異常探勘方法都需要在未知的情況下假設資料集具有某種資料分佈，所以無法精確的探勘出所有的異常。

(2) 距離基礎方法(distance based method)：最初是由Knorr和Ng所提出來的(Knorr et al., 2000)。此外，Ramaswamy等人(2000)延伸了距離基礎的異常偵測演算法：最大值D_k的前n個點被認為是異常，$D_k(p)$表示p的第k個最近鄰居的距離。此類方法使用叢集演算法將資料集分成好幾群(group)。而後，在這些群裡做修剪和批次處理以改進異常偵測的效能(Aggarwal & Yu, 2001)。相較於統計基礎方法，距離基礎的異常偵測隱含了對不同標準分佈“不一致性檢驗(discordancy test)”的一般概念，距離基礎的異常偵測可將原始資料的分佈調整成標準分佈來避免過度的計算(Knorr & Ng, 1998)。然而，距離基礎方法也有其缺點存在，由於它是考慮整個資料集之中資料點的距離，因此，能成功地偵測出全域性(global)的異常。一但資料庫中，資料點距離的分佈非常不平均，這一類的方法就不適用了，因為，它無法偵測出那些區域有異常。

(3) 偏離基礎方法(deviation based method)：此異常偵測法，並沒有使用統計檢定或基於距離測量辨識例外的物件。反而，它是利用資料點(object)在一個資料群組(group)中的特性來偵測異常。若資料點偏離資料群組描述(description)，此資料點就稱為異常。因此，在這個方法中偏差就是指異常(Han et al., 2000)。

(4) 叢集基礎方法(clustering based method)：此異常偵測法是透過負面列表的觀念來呈現，也就是說透過各種叢集演算法求出各群集的資料集(clustering dataset)之後，剩下無法歸納入各群中的資料就是異常的可能對象。把小的叢集當成異常(He, Xu, & Deng, 2003 ; Jiang, Tseng, & Su, 2001)，或從原始的資料集透過移除群來辨識異常(Yu et al., 2002)。但是這中間可能有雜訊的存在，需要進一步的處理和解釋，所以叢集基礎方法用來偵測異常的作法是一種間接的方法。在效率上視叢集演算法的效率而決定，並無法直接改善異常探勘搜尋的時間成本。

(5) 密度基礎方法(density-based method)：此法是由Breunig 等人(2000)所提出。此方法主要是為了改善距離基礎方法中無法偵測出區域性異常的缺點，它的基本觀念為針對資料集中的每個資

料點，只考慮該點鄰近區域(neighborhood)中的鄰居點與自己的關係，定義一個區域異常係數(local outlier factor ; LOF)，再根據這個係數來判斷該資料點為異常的可能性。異常係數值越高，越有可能是異常。此法最大的優點是能夠準確的偵測出區域性異常，但是仍然有些特殊的情況是無法解決的。例如，異常較集中時，此法會把這些異常誤判為一個群聚(cluster)。基於叢集的異常偵測技術會把小群聚當作是異常(Jiang et al., 2001)。

(6) 子空間基礎方法(sub-space based method)：此法為 Aggarwal 和 Yu (2005)所介紹在高維度空間中，異常探勘的一個新投影基礎的技術。因高維度對應到低維度其投影可能性之組合數太多，無法單純使用窮舉法來列舉，因此發展出此一新式的投影法。此方法可解決距離基礎方法不能克服的高維度問題。在高維度環境下使用有效率的漸進式搜索技術(evolutionary search technique)，成功處理複雜的高維度資料問題。Z.He 等人另外提出一個頻繁樣式基礎的異常探勘方法(He, Xu, Huang, & Demg, 2004)。Wei 等人(Wei, Qian, Zhou, Jin, & Yu, 2003)並引進了 hypergraph 模型應用在類別資料集之異常偵測。

(7) 類神經網路基礎方法(neutral network based method)：自動關聯類神經網路(auto-associative neural networks)最初目的是為了簡化維度所提出的模型，近年來也有學者將其用來做為異常偵測的方法(Harkins, He, Williams, & Baxter, 2002 ; Willams, Baxter, He, Hawkins, & Gu, 2002)。這個方法的優點是不需要作分佈假設而且容易適應類別變數。缺點是訓練的時間太長且對於模型的參數過於敏感。

2.2 頻繁樣式基礎之離異探勘

Z. He等人所提出的FindFPOF方法使用頻繁樣式來找出異常(He et al., 2005)，此法，可有效處理有類別屬性之資料。使用者可以使用任一種資料探勘軟體去執行關聯規則裡的Apriori方法，將輸出的頻繁樣式來找出交易資料中的異常。

從知識發現(knowledge discovery)的觀點，頻繁樣式也就是共同樣式(common patterns)，適用於許多物件(objects)或在資料集裡物件有很大的百分比。相較之下，離異偵測集中在資料物件是非常小的百分比。因此，FindFPOF方法的想法是非常直覺的，利用頻繁樣式來偵測出離異的交易。

在說明FindFPOF方法之前，先簡單了解Z. He等人對於如何使用頻繁樣式及來找出異常的定義。

定義一：令 $D=\{t_1, t_2, \ldots, t_n\}$ 為包含n筆交易集合之資料庫，I為所有可能項目的集合，X唯一任意的項目集，給予一個門檻值*min_sup*。令*FPS*(*D, min_sup*)是所有頻繁樣式的集合，即*FPS*(*D, min_sup*) = $\{\forall X \subseteq I \mid support(X) \geq min\_sup\}$。對於每一筆交易$t$，它的頻繁樣式異常係數(*Frequent Pattern Outlier Factor*; *FPOF*)被定義為：

$$FPOF(t) = \frac{\sum_{X \subseteq t,\ X \in FPS(D,\ min\_sup)} support(X)}{\left\| FPS(D,\ min\_sup) \right\|} \quad (1)$$

如果一筆交易t包含很多頻繁樣式，則它的*FPOF*值將會是大的，這就表示這筆交易是異常的機率越小。相反的，若是交易的*FPOF*值越小，則此筆交易會是異常的機率越大。明顯地，*FPOF*值會介在0和1之間。

定義二：每一筆交易t，如果 $X \not\subset t$ ，項目集X被認為對t是傾向於矛盾的(contradictive)。X對t的*contradict-ness*被定義為：

$$Contradict\text{-}ness(X,t) = \left(\|X\| - \|t \cap X\|\right) * support(X) \quad (2)$$

在這個方法，公式(1)是使用來當作辨識異常的基本測量。描述為什麼辨識出的異常是不正常的原因，項目集不包含在交易內(就是說那個項目集是自相矛盾的交易)都是好的解釋原因。

在公式(2)裡，項目集X的支援(support)越大，對於t交易的X的*contradict-ness*值會越大，因為一個大的X支援值暗示著有大的偏離(deviation)。並且較長的項目集比較短的項目集會給予更好的解釋。

定義三：即前K個傾向於矛盾頻繁樣式(*TKCFP-Top K Contradict Frequent Pattern*)：給定的D、*min_sup*和*FPS*(*D, min_sup*)。對一筆交易t，一項目集X被認為是其前K個傾向於矛盾頻繁樣式，則不存在超過(K − 1) 項目集的 *contradict-ness* 高於 X ，且 $X \in FPS(D,\ min\_sup)$ 。

FindFPOF探勘異常的交易總共包含三個步驟：

1. 使用現有的資料探勘軟體裡的關聯法則從資料庫中探勘出所有的頻繁樣式。
2. 計算出每一筆交易的FPOF值。
3. 發現前n筆異常使用前K個傾向於矛盾頻繁樣式來解釋這n筆異常。

3. 以頻繁樣式為基礎之異常交易偵測演算法

3.1 方法概念

FindFPOF在探勘的過程中，利用關聯法則探勘裡的Apriori方法產生出出來的頻繁項目集來計算出交易資料庫裡異常的交易。但是，在Apriori的方法中，會有產生過多的候選項目集(candidate itemsets)的問題，而且最後會成為頻繁項目集的數

量卻不多，以致於浪費了龐大的運算時間和記憶體空間，讓整個探勘的時間變長。且找出異常的交易才是目標所在，而非在找頻繁項目集，因此，本論文提出減少產生頻繁樣式集(reduce frequent pattern set; Rfp)的方法來快速找出異常交易，稱之為RfpFPOF方法。用來解決FindFPOF會因為需要產生許多頻繁項目集數量的問題。

減少頻繁樣式集主要的概念在於：「每一階段的支持度最高的那一個項目不要去組下一階段的候選項目集」。例如在一個資料庫內，頻繁 1-項目集(L_1)有 $A=0.8$、$B=0.65$、$C=0.59$、$D=0.45$，Apriori 的候選 2-項目集(C_2)有{*A, B*}, {*A, C*}, {*A, D*}, {*B, C*}, {*B, D*}, {*C,D*}六個候選項目集，而刪除最高支持度的{*A*}之後，候選 2-項目集(C_2)就只有{*B, C*}, {*B, D*}, {*C, D*}三個候選項目集。因此，可以有效的減少所產生的頻繁樣式，且有助於加速尋找出異常的速度。以下為一範例：

範例一：表 1 為範例資料庫，共分二個欄位：TID 欄位代表每一筆交易的唯一識別號，Items 欄位代表每一筆交易所購買的項目。資料庫內共有九筆交易與五項不同的交易項目 *A*、*B*、*C*、*D*、*E*，因此 *I* = {*A, B, C, D, E*}。設最小支持度為 20%(也就是需有二次或二次以上的購買)，執行 RfpFPOF 演算法如下：

表 1 交易資料庫

TID	Items
T1	A, B, E
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

首先在 Pass 1 產生頻繁 1-項目集。Pass 2 是由頻繁 1-項目集產生候選 2-項目集，但是在組候選 2-項目集的時候，忽略在頻繁 1-項目集裡支持度最高的那個項目，而 *B* 的支持度最高，所以候選 2-項目集原本由{*A, B*}, {*A, C*}, {*A, D*}, {*A, E*}, {*B, C*}, {*B, D*}, {*B, E*}, {*C, D*}, {*C, E*}, {*D, E*}10 個減少為{*A, C*}, {*A, D*}, {*A, E*}, {*C, D*}, {*C, E*}, {*D, E*}此 6 個候選 2-項目集。接著 Scan *DB* 取得 Count 留下支持度大於等於 20%者成為頻繁 2-項目集。基於向下閉合性質：「一項目集，其子集皆為頻繁時，則此項目集則為繁頻項目集。」，所以不用執行到 Pass 3，若由頻繁 2-項目集產生候選 3-項目集，因為在頻繁 2-項目集裡支持度最高的是{*A, C*}這個項目，所以此時可以組的頻繁 2-項目集只剩下{*A, E*}而已，因此減少頻繁樣式集的方法即可結束。表 2 為各個項目與其支持度，資訊可由圖 1 得知。

表 2 頻繁樣式與支持度

ID	Itemset	Support
1	A	0.67
2	B	0.78
3	C	0.67
4	D	0.22
5	E	0.22
6	A, C	0.44
0	A, E	0.44

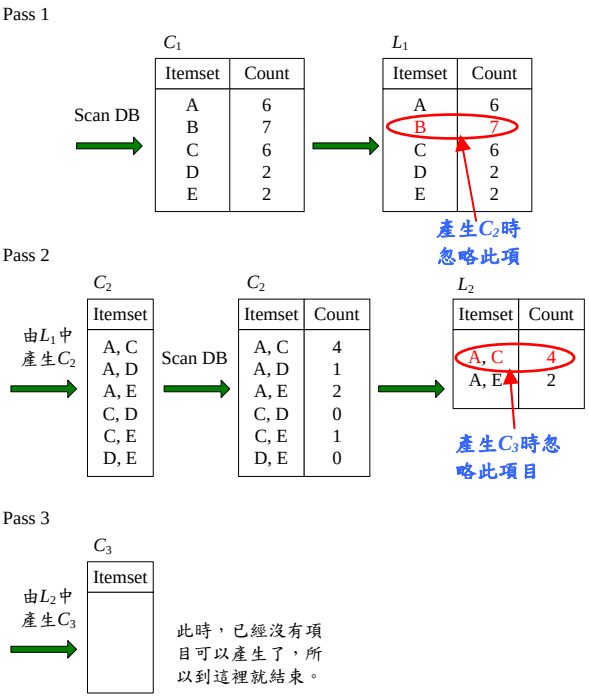

圖 1 減少頻繁樣式集之演算法範例

FindFPOF 方法經過第一次掃描交易資料庫後，會產生五個候選 1-項目集，十個候選 2-項目集，二個候選 3-項目集，共 17 個。而 RfpFPOF 方法將原本 17 個候選項目集減少到 11 個，兩演算法所產生之候選項目集整理如表 3 所示。

表 3 FindFPOF 法和 RfpFPOF 法所產生的候選項目集合

演算法	產生的候選項目集合C_k		數量
FindFPOF法	C_1	A、B、C、D、E	17
	C_2	AB、AC、AD、AE、BC、BD、BE、CD、CE、DE	
	C_3	ABC、ABE	
RfpFPOF法	C_1	A、B、C、D、E	11

	C_2	AC、AD、AE、CD、CE、DE	
	C_3	∅	

由範例三得知，RfpFPOF 方法可以減少候選項目集的產生，以縮短第二次掃描資料庫時搜尋候選項目集出現次數的時間，可造成加速程式執行的結果。

3.2 演算法分析

以下為RfpFPOF演算法：

Algorithm RfpFPOF

Input：*D* //the transaction database
min_sup //user defined threshold for the permissible minimal support
top-n //user define threshold value for *top n outliers*

Output：*The top-n outliers*

```
01. begin
02.    L1 = {large 1-itemset}
03.    for(k = 2 ;  L_{k-1} != ∅ ; k++){
04.       C_k = new apriori_gen( L_{k-1} );
05.       for each transaction t ∈ D {
06.          C_t = subset(C_k, t);
07.          for each candidate c ∈ C_t
08.             c.count++;
09.       }
10.       L_k = {c ∈ C_k | c.count ≥ min_sup}
11.    }
12.    L = ∪L_k;//the set of all discovered frequent
       patterns is donated as: FPS(D,min_sup)
13.    for each transaction t in D{
14.       for each frequent pattern X in FPS(D,
          min_sup){
15.          if t contains X then
16.             FPOF(t) = FPOF(t) + support(X)
17.       }
18.    }
19.    Output the transactions in the ascending order
       of their FPOF value. Stop when it output
       top-n transaction.
20. end
```

procedure new apriori_gen(L_{k-1})

```
01.  C_k = ∅
02.  for each itemset X ∈ L_{k-1}
03.     if support(X) is maximum then
04.        continue;
05.     for each itemset Y ∈ L_{k-1}  and  X ≠ Y
06.        if support(Y) is maximum then
07.           continue;
08.      if(X[1] = Y[1]) ∧ ... ∧ (X[k-2] = Y[k-2])
         then{
09.           c = X + Y[k-1];
10.           if infrequent_subset(c, L_{k-1}) then
11.              delete c;
12.           else  C_k := C_k + c ;
13.        }
14.  return Ck;
```

procedure infrequent_subset(c, L_{k-1})

```
01.  for each(k-1)-subset s of c
02.     if s ∉ L_{k-1}  then
03.        return TRUE;
04.  retrun FALSE;
```

RfpFPOF 的第 2 行是找出頻繁項目集的集合 L1。第 3 行到第 12 行是 L_{k-1} 用於產生候選 C_k ，以便找出 L_k 。new apriori_gen 是在產生候選項目集的，第 3 行和第 6 行是先找出在第 k-頻繁樣式集中支持度最高的項目，然後這個項目不要去組 $k+1$ 階段的候選項目，第 8 行是使用 apriori 性質刪除哪些具有非頻繁子集的候選，第 9 行是來產生頻繁項目。第 10 到第 12 行是使用 apriori 性質來刪除非頻繁子集的候選。infrequent_subset 是非頻繁子集的測試。如果某一項目的子集沒有出現再上一階段的頻繁樣式集當中(infrequent_subset 的第 3 行)，就把這一項目刪除(new apriori_gen 的第 11 行)，若此項目的子集有出現在上一段的頻繁樣式中(infrequent_subset 的第 4 行)，就把這一項目加到候選項目集 C_k 裡面(new apriori_gen 的第 12 行)。RfpFPOF 的第 6 行是對於每一筆交易，使用 subset 函數找出所有子集，之後第 7 行到第 8 行是在對每一個相應的候選項目做累加，第 10 行將所有滿足最小支持度的候選項目形成頻繁項目集 L。第 13 行到第 18 行是計算出每一筆交易的 FPOF 值。第 19 行是找出前 n 筆異常的交易。

4. 實驗結果與效能分析

4.1. 實驗環境與評估方式

本實驗所使用到的電腦其中央處理器為AMD Athlon(tm) 64 Processor 2800+ (1.80GHz)，記憶體容量為512MB，所使用的作業系統是Microsoft Windows XP Service Pack 2，而實驗中所有程式之執行檔都是以Dev-C++ 5 Beta 9.2 (4.9.9.2) Compiler編譯而成。為了產生實驗用的資料，採用了IBM 所提供的人工資料產生器。關於效能評比的部分，主要的準則為：執行時間與找出異常交易之準度。執行時間評估則是包含找出頻繁項目集所需的時間及找計算各演算法的關聯法則與計算出異常交易所需要的時間；另外，找出異常交易之準度的評估

是使用 $FPOF(t) = \frac{\sum support(X)}{\|FPS(D, min\_sup)\|}$ 公式來評估 (He et al., 2005)，若FPOF的值越小，就越有可能是異常。

關於比較的演算法，本論文將採用 FindFPOF 演算法為基準與 RfpFPOF 演算法比較。FindFPOF 演算法為近年所提出之在類別資料庫中使用頻繁項目集找出異常交易演算法中效能卓越者，並且與本論文所提出之 RfpFPOF 演算法同屬於使用頻繁項目集找出異常交易的演算法。故本論文採用演算法 FindFPOF 為比較之基準。

4.2. 實驗資料集

本論文所使用的實驗資料為 IBM 公司所提供的模擬資料集產生器來產生，其模擬資料集產生器有數個參數分別列於表 4。

表 4 IBM 模擬資料集產生器一些參數的涵意

參數名稱	意義
$\|D\|$	交易資料庫內交易筆數
$\|T\|$	每筆交易的平均交易數量
$\|I\|$	最大可能頻繁項目集合大小
$\|L\|$	最大可能頻繁項目集之數
N	交易項目的數量

在產生模擬資料集時，本論文第一個實驗設定 $N = 500$、$|L| = 1000$、$|I| = 4$ 以及 $|T| = 10$，$|D|$ 則選用五種數值：20k、40k、60k、80k、100k，此實驗目的是不同交易筆數各演算法執行時間是否有差距，且在不同的支持度下所執行的時間之研究。第二個實驗設定 $N = 500$、$|L| = 1000$、$|I| = 4$ 以及 $|D| = 100k$，$|T|$ 則選用五種數值：8、9、10、11、12，此實驗目的是要探討不同的平均交易數量是否會影響執行時間。

4.3. 效能評估

本論文之實驗各使用了五個人造資料集，依 RfpFPOF 與 FindFPOF 方法論進行實驗比較。圖 2 在最小支持度為 1% 的情形下，RfpFPOF 與 FindFPOF 在不同交易筆數的資料庫所執行的時間之比較。由該圖可以得知 RfpFPOF 所執行的時間皆少於 FindFPOF，隨著交易筆數的增加，RfpFPOF 與 FindFPOF 所執行的時間差距稍增。在交易筆數為 20k 的資料庫中，RfpFPOF 的執行時間比 FindFPOF 快了 5.84 秒，減少 21.2%的時間。在交易筆數為 40k 的資料庫中，RfpFPOF 的執行時間比 FindFPOF 快了 8.08 秒，降低 16.4%的時間。在交易筆數為 60k 的資料庫中，RfpFPOF 的執行時間比 FindFPOF 快了 10.29 秒，減少 14.2%的時間。在交易筆數為 80k 的資料庫中，RfpFPOF 的執行時間比 FindFPOF 快了 12.49 秒，減少 13.3%的時間。在交易筆數為 100k 的資料庫中，RfpFPOF 的執行時間比 FindFPOF 快了 14.48 秒，減少 11.5%的時間。

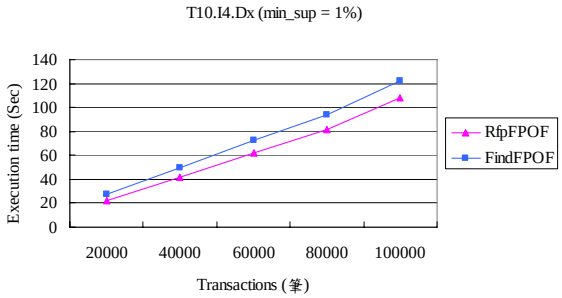

圖 2 不同交易筆數的模擬資料庫下執行時間之比較

圖 3 為固定資料集 20k 的情況下，在不同最小支持度下執行的時間之比較。本研究實驗最小支持度會從 0.7 開始，而不是 0.5 或 0.6，是因為支持度愈低所花的時間會愈長，所以本研究所設定最低的最小支持度就從 0.7 開始。由該圖發現，當支持度在 0.7%以下時，RfpFPOF 與 FindFPOF 所執行的時間明顯增加並且差距也變大，是因為該資料集裡的項目之支持度都大於最小支持度，以至於頻繁項增多，導致執行時間也更久。雖然，本論文所提出的 RfpFPOF 方法雖然時間也有變多，但是時間的成長率沒有 FindFPOF 來得快。

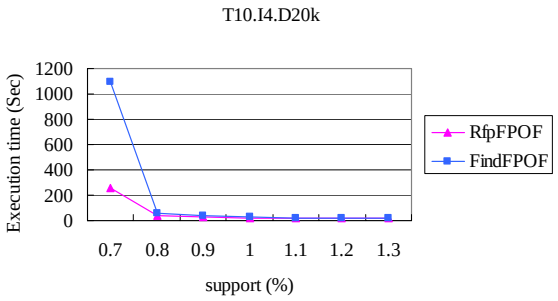

圖 3 D = 20k 時不同最小支持度下執行的時間之比較

圖 4 與圖 5 各為固定資料集 40k 與 60k 的情況下，在不同最小支持度執行時間之比較。兩個方法隨著支持度的減少，執行時間明顯增加。因為資料集裡項目的支持度大於最小支持度的數量越來越多，所以頻繁項目集的數量增加許多。在計算頻繁項目集與 FPOF 值的時候會花費很多時間，故執行時間也跟著增加。

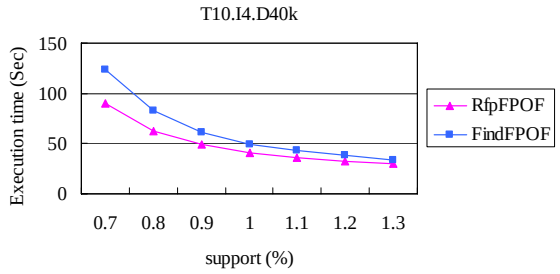

圖 4 D = 40k 時不同最小支持度下執行的時間之比較

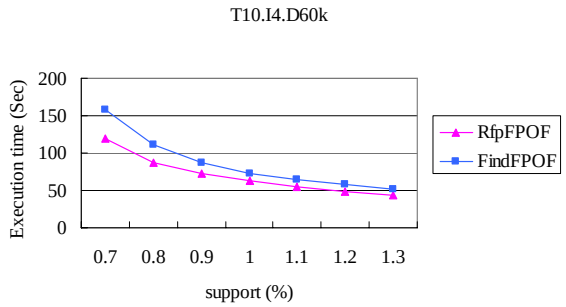

圖 5 D = 60k 時不同最小支持度下執行的時間之比較

圖 6 與圖 7 各為固定資料集 80k 與 100k 的情況下，在不同最小支持度下執行的時間之比較。由該圖發現兩個方法執行時間的差距不大。因為此資料集的項目支持度大多比最小支持度小，所以頻繁項目不會很多，執行時間相對的也比較快。

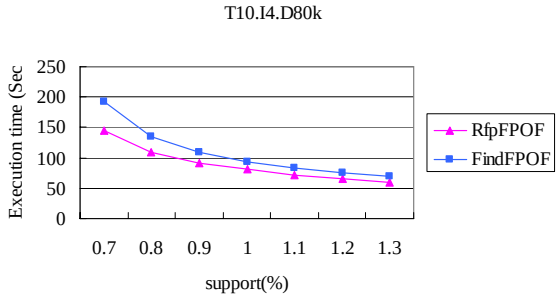

圖 6 D = 80k 時不同最小支持度下執行的時間之比較

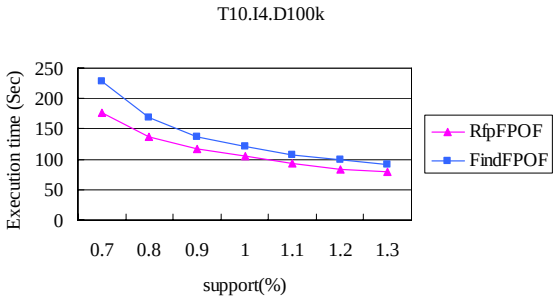

圖 7 D = 100k 時不同最小支持度下執行的時間之比較

由圖 3 到圖 7 可以得到以下結論：當支持度越小，RfpFPOF 與 FindFPOF 所執行的時間差距愈大，支持度越大執行的時間差距也相對的減少。原因是因為支持度越大，其通過最小支持度的頻繁項相對的變少，RfpFPOF 與 FindFPOF 兩方法所產生的頻繁項目集的數量會越接近，所以執行的時間幾無差異。

圖 8 在最小支持度為 1%的情形下，RfpFPOF 與 FindFPOF 在不同平均交易長度的資料集中所執行時間之比較。由該圖可以得知 RfpFPOF 所執行的時間皆少於 FindFPOF，隨著平均交易長度的增加，RfpFPOF 與 FindFPOF 所執行的時間差距稍增。在平均交易長度為 8 的資料集中，RfpFPOF 的執行時間比 FindFPOF 快了 11.02 秒，減少了 13.6%的時間。在平均交易長度為 9 的資料集中，RfpFPOF 的執行時間比 FindFPOF 快了 13.67 秒，減少了 13.8%的時間。在平均交易長度為 10 的資料集中，RfpFPOF 的執行時間比 FindFPOF 快了 14.48 秒，減少了 11.9%的時間。在平均交易長度為 11 的資料集中，RfpFPOF 的執行時間比 FindFPOF 快了 24.41 秒，減少了 15.7%的時間。在平均交易長度為 12 的資料集中，RfpFPOF 的執行時間比 FindFPOF 快了 33.59 秒，減少了 16.7%的時間。

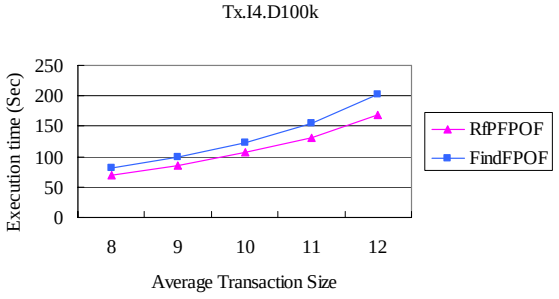

圖 8 不同平均長度的模擬資料集下執行時間之比較

5. 結論與未來研究方向

5.1. 結論

傳統異常探勘方法並無有效的使用在多維環境中類別屬性資料。Z. He 等人提出之 FindFPOF 方法處理了多維度空間與類別屬性的資料的問題。但是，FindFPOF 方法會因為頻繁項目集(frequent itemsets)的數量越多，所執行的時間會愈長。所以本研究提出了 RfpFPOF 方法，減少頻繁項目集數量來解決 FindFPOF 方法會因為頻繁項目集的數量增加而執行時間變長。實驗結果顯示，RfpFPOF 方法能找出較少的頻繁項目集以及找出異常交易，因此在整體時間較 FindFPOF 方法勝出。

5.2. 未來研究方向

未來可行的研究方向在找出更多可以刪除的項目及加速計算的方式，而不會影響到方法的準確度。因此，總結未來研究發展方向如下：

1. 由於本論文僅移除每一階段頻繁項的支持度最大者不去組下一階段的候選項目集；未來能考慮刪除更多不需要用到的頻繁項目，讓整個執行的時間可以更短的方法。
2. 在本論文的研究實驗中，只考慮到類別屬性的資料，在未來可以考慮使用混合屬性的資料，在分割數值屬性的資料時，不要以傳統的方式去切割，而是考慮到每個數值間的距離有其關連性，並且可依每個階段的不同其數值間有不同的關連，動態的切割出適合的間距。

參考文獻

[1] Aggarwal, C. C., & Yu, P. S. (2001). Outlier Detection for High Dimensional Data. ACM-SIGMOD 2001 International Conference on Management of Data, 37-46.

[2] Aggarwal, C. C., & Yu, P. S. (2005). An Effective and Efficient Algorithm for High-Dimensional Outlier Detection. The VLDB Journal, 14(2), 211-221.

[3] Barnett, V., & Lewis, T. (1994). Outliers in Statistical Data. John Whiley, Sons and Chichester.

[4] Breunig, M. M., Kriegel, H. P., Ng, R. T, & Sander, J. (2000). LOF: Identifying Density-based Local Outliers. ACM-SIGMOD 2000 International Conference on Management of Data, 93-104.

[5] Chen, M. S., Han, J., & Yu, P. S. (1996). Data Mining: An Overview from Database Perspective. IEEE Trans. Knowledge and Data Eng., 8(6), 866-883.

[6] Han, J., & Kamber, M. (2000). Data Mining: Concepts and Techniques. Morgan Kaufmann.

[7] Harkins, S., He, H., Williams, G. J., & Baxter, R. A. (2002). Outlier Detection Using Replicator Neural Networks. 4th International Conference DaWaK 2002, 170-180.

[8] He, Z., Xu, X., & Deng, S. (2003). Discovering Cluster-based Local Outliers. Pattern Recognition Letters, 24(9-10), 1641-1650.

[9] He, Z., Xu, X., Huang, J. Z., & Demg, S. (2004). A Frequent Pattern Discovery Method for Outlier Detection. 5th International Conference WAIM 2004, 726-732.

[10] He, Z., Xu, X., Huang, J. Z., & Deng, S. (2005). FP-Outlier: Frequent Pattern Based Outlier Detection. Computer Science and Information System, 2(1).

[11] He, Z., Xu, X., & Deng, S. (2005). A Unified Subspase Outlier Ensemble Framework for Outlier Detection in High Dimensional Spaces. 6th International Conference Web-Age Information Management, 632-637.

[12] Jiang, M. F., Tseng, S. S, & Su, C. M. (2001). Two-Phase Clustering Process for Outliers Detection. Pattern Recognition Letters, 22(6-7), 691-700.

[13] Knorr, E. M., & Ng, R. T. (1998). Algorithms for Mining Distance-based Outliers in Large Data Sets. 24th International Conference on Very Large Data Bases, 392-403.

[14] Knorr, E. M., Ng, R. T., & Tucakov, V. (2000). Distance-based Outliers: Algorithms and Applications. The VLDB Journal, 8(3-4), 237–253.

[15] Otey, M. R., Ghoting, A., & Parthasarathy, A. (2004). Fast Distributed Outlier Detection in Maxed-Attribute Data Sets. Data Mining and Knowledge Discovery, 12(2-3), 203-228.

[16] Ramaswamy, S., Rastogi, R., & Kyuseok, S. (2000). Efficient Algorithms for Mining Outliers from Large Data Sets. ACM-SIGMOD 2000 International Conference on Management of Data, 93-104.

[17] Willams, G. J., Baxter, R. A., He, H., Hawkins, S., & Gu, L. (2002). A Comparative Study of RNN for Outlier Detection in Data Mining. IEEE ICDM 2002, 709-712.

[18] Wei, L., Qian, W., Zhou, A., Jin, W., & Yu, J. X. (2003). HOT: Hypergraph-based Outlier Test for Categorical Data. 7th Pacific-Asia Conference PAKDD 2003, 399-410.

[19] Yu, D., Sheikholeslami, G., & Zang, A. (2002). Findout: Finding Outliers in Very Large Datasets. Knowledge and Information Systems, 4(3), 387-412.