一個適用於連續探勘的關聯規則演算法

An efficient algorithm for continuously mining of association rules

呂永和
國立台灣科技大學資訊管理系
yhl@cs.ntust.edu.tw

徐雅琪
國立台灣科技大學資訊管理系

摘要

目前有許多關聯規則演算將交易資料庫的資料，轉成儲存於主記憶體內部的資料結構，再對這個資料結構進行探勘的動作，由於資料掃瞄的動作是在主記憶體中進行；因此，其速度均比傳統的 Apriori 演算法還要快很多。但這些方法所面臨的共同難題，就是當交易資料庫太大時，這些方法必須將部份交易資料，存入磁碟機中，降低了資料探勘的速度。而且，一般的資料探勘工作，都需要對同一資料以不同參數連續探勘，方能找出有用的資訊。目前大多數演算法都只考量單次的資料探勘，並未考慮多次探勘時演算法的效率。

本論文提出一個以變動長度編碼(Run-length Encoding)為主的演算法稱為 CM 演算法，使用變動長度編碼技術，將交易資料庫壓縮成少量的資料，然後直接對主記憶體中的壓縮資料，進行連續的資料探勘，不需要將交易資料解壓縮。 經實驗證明，在多次探勘的情況下，CM 演算法優於目前的演算法。

關鍵詞：資料探勘、關聯規則、連續探勘、資料壓縮

Abstract

Some algorithms store transactional databases in the main memory to reduce the time needed in scanning databases. Since the main memory is much faster than the disk. These algorithms gain significant performance improvements over the traditional Apriori algorithm. However, some databases may be too big to fit in the main memory. As such, part of these databases will be stored in a disk, which deteriorates the performance of the mining process. Besides, data mining is a continuous process, which requires several rounds before useful information is found. Most of the existing algorithms concern only one-time-mining. The performance of data mining in several rounds is seldom considered.

In this paper, we propose an algorithm, called the CM algorithm, to mine association rules from compressed databases. The advantage of the CM algorithm is that it needs not decompress the data structure before performing its mining process. Through experiments, we show that the CM algorithm significantly reduces the time required in continuously mining of the transactional database.

Key words: Data Mining, Association Rules, Continuously mining, Data Compression

一、研究背景與動機

資料探勘的技術，可依資料庫的型態與應用的領域來加以分門別類，一般可分為關聯規則(Association Rules)、分類(Classification)、群組化(Clustering)以及序列型樣(Sequential Patterns)等，這些方法為滿足不同的需求，處理資料的方式以及分析出的結果皆不相同。在這些方法中，關聯規則的探勘是近幾年來最被廣泛研究的議題，關聯規則已被運用於實際的企業應用中，所得到的成效非常的顯著。目前有許多關聯規則演算，將交易資料庫的資料轉成儲存於主記憶體內部的資料結構，再對這個資料結構進行探勘的動作，由於資料掃瞄的動作是在主記憶體中進行；因此，其速度均比傳統的 Apriori 演算法還要快很多。但這些方法所面臨的共同難題，就是當交易資料庫太大時，這些方法必須將部份交易資料，存入磁碟機中，降低了資料探勘的速度。而且，一般的資

料探勘工作，都需要對同一資料以不同參數連續探勘，方能找出有用的資訊。目前大多數演算法都只考量單次的資料探勘，並未考慮多次探勘時演算法的效率。

二、相關研究

關聯規則探勘方法，由 Agrawal 等，於 1994 年首先提出，所提出的 Apriori 演算法[2][3]，利用簡單且循序漸進的方式，找出關聯規則。藉由讀取資料庫中的所有交易，來找出出現次數頻繁的項目組，稱為高頻項目組(Frequent Itemsets)，再利用這些高頻項目組產生關聯規則(Association Rules)。

(一) 類似 Apriori 的演算法(Apriori-Like Algorithms)

所謂類似 Apriori 演算法，是以 Apriori 為基礎，改進其中花費時間最多的部分：產生候選項目組(Candidate Generation)和檢查候選項目組(Test Candidates)。著名幾個類似 Apriori 的演算法包括 Apriori[2][3]、DLG[4]、RARM[5]、及 VIPER[6]等。其中 Apriori 是利用反覆讀取整個資料庫的方式，由短的項目組產生長的項目組，直到找出所有的高頻項目組，並利用所找出的高頻項目，尋找關聯規則。DLG[4]演算法是使用 Bit-Vector 來紀錄交易資料，以方便計算支持度(Support)，這樣的紀錄方式可以減少磁碟存取時間，並且採用圖的資料結構來產生候選項目集合。RARM[5]演算法是將原始的資料庫經由前置處理，存成一個 Trie 的資料結構，這個資料結構依 Support Count 由大到小儲存所有的 1-項目組(1-itemsets)及 2-項目組(2-itemsets)；之後就在這個 Trie 上面進行尋找高頻-1 項目組(Frequent-1 itemsets)與高頻-2 項目組(Frequent–2 itemsets)的操作，尋找其它長度的高頻項目組，仍須掃瞄交易資料庫。VIPER[6]演算法主要是提出一套壓縮的方式，希望能減少資料儲存的空間，但是在產生高頻項目組時，每次均要將前一個高頻項目組解壓縮，才能產生候選項目組，並將結果再以壓縮的方式儲存在磁碟中。這一壓一解的動作，降低了執行的效率。

(二) 非類似 Apriori 的演算法 (Non-Apriori-Like Algorithms)

這類演算法改進 Apriori-Like 需反覆處理候選項目組的缺失。如 FP-Growth[7]、H-Mine[8]等。FP-Growth 將資料庫先掃描一次，找到高頻-1 項目組(Frequent-1 itemsets)，接著進行第二次的掃描資料庫，以建立 FP-tree 結構，之後透過遞迴的方式利用 FP-tree 結構，找出所有的高頻項目組。H-Mine[8]為掃描資料庫一次後，找出所有的高頻-1 項目組，並計算高頻-1 項目組的次數，之後掃描資料庫第二次，將高頻-1 項目組建立成 Header Table 結構，再利用此 Table 搭配深度優先法則並加上 Link-list，以找出所有的高頻項目組。DLG 利用 Bit-Vector 的紀錄方式來紀錄交易資料，每個項目的位元向量的長度，就是交易資料的筆數。因此，當交易資料庫中的交易筆數很多，或交易資料庫中的項目很多時，DLG 就需要花費相當大的記憶體空間來儲存 Bit-Vector，主記憶體的空間會不敷使用，相對的減低了整體的運算速度。

RARM 演算法提出一個前置處理的機制，但是若遇到使用者想要尋找高頻 3-項目組(Frequent 3-itemsets)或更長的高頻項目組時，還是必須去重新掃描資料庫，且為結構中只儲存 1-項目組與 2-項目組，無法將完整的交易資料載入。FP-tree 演算法可以有效的解決 Apriori-Like 演算法所面臨的問題，但是在[5]中作者特別提到，當使用者設定的最小支持度小於 1.5%後，則 FP-tree 運算的速度與傳統 Apriori 不相上下，並且在[1]中，作者提到當設定很小的最小支持度時，一個 FP-tree 會比比原始資料庫佔據更多的記憶體空間。針對這些相關演算法所面臨到的問題，我們提出一套新的關聯規則探勘演算法。

三、CM 關聯規則演算法

我們發展的方法是將原始的交易資料庫，透過前置處理將交易資料庫儲存於一個矩陣，此矩陣稱為 CM(Count Matrix)矩陣，此 CM 矩陣可以完整的紀錄所有的交易資料，當使用者在作資料探勘時，只需要讀取 CM 矩陣，不需去掃描原始資料庫。另外，當交易資料庫發生異動時，我們只需調整 CM，不需要將整個 CM 進行重建。

對於我們所提出的資料探勘演算法，我們將分成三個部分來進行說明，第一部分為前置處理，主要是說明如何建置 CM；第二部分為 AND 運算的演算法，為本演算法的核心，我們將說明如何對 CM 中的交易記錄進行"AND"運算以求算支持度；第三部分為我們所提出的資料探勘演算法，我們會說明如何運用 CM 來進行資料探勘。

(一) 前置處理

我們所提出的關聯規則演算法，稱之為 CM(Count Matrix)演算法，在利用 CM 進行資料探勘之前，首先建立一個矩陣，稱為 CM 矩

陣(Count Matrix)，這個矩陣包含了三個部分，分別為：

- CMI(CountMatrix Item)：這個部分包含整個交易資料庫所有出現的項目種類，即所有的 1-項目組。
- CMRLE(CountMatrix Run-Length Encoding)：以我們發展的一套壓縮方法，來記錄每個項目的交易紀錄。
- CMCn(CountMatrix Combination for n-itemset)：這個部分是紀錄包含特定項目的 n-itemset 的個數。

我們以一個例子來進行說明，表 1 為一個原始的交易資料庫，我們以此交易資料庫來建立一個 CM 矩陣(Count Matrix)，如表 2 所示。

表 1：原始交易資料庫

Tid	Item set
T10	AB
T20	ABC
T30	ABC
T40	BCD
T50	BCD
T60	BCDF
T70	CDF
T80	CDF

表 2：CM 矩陣

CMI 項目集	CMRLE 交易紀錄	CMC1(1-項目組的個數)	CMC2(2-項目組的個數)	CMC3(3-項目組的個數)	CMC4 (4-項目組的個數)
A	1*3*5	3	5	2	0
B	1*6*2	6	12	7	1
C	2*7	7	15	9	1
D	4*5	5	11	7	1
F	6*3	3	7	5	1

CM 矩陣包括了所有的交易資訊，其主要分成三部份，CMI(項目集)、CMRLE(壓縮之交易紀錄)、CMCn(n-項目組的個數)，底下我們將詳細說明此三部份的內容、意義與作法。

(1) CMI（Count Matrix Item; 項目集）

CMI 代表目前在交易資料庫中，所有的項目種類。如表 2 紀錄的 A、B、C、D、F。

(2) CMRLE(Count Matrix Run-Length Encoding; 壓縮之交易紀錄）

CMRLE 代表目前在交易資料庫中各個項目的所有交易紀錄。在 CMRLE 中以"*"區隔編碼之後的奇數位置與偶數位置(以所產生的碼的第一個數的位置為奇數位置，依此類推)。奇數位置的數值所代表的意思是"沒有包含此項目"的連續交易筆數，偶數位置的數值則代表"有包含此項目"的連續交易筆數。

為了讓奇數位置表示沒有包含此項目的連續交易筆數，我們在編碼之前，在每一個項目的 Bit-Vector 之前都補一個"0"。在解讀 CMRLE 時，將第一個數減 1。

舉例來說，從表 1 中，項目 A 的 Bit-Vector 為 11100000，將其前面補 0 後為 011100000，其 CMRLE 編碼如表 2 所示為 1*3*5，將其中第一個數值"1"減 1 等於 0，表示一開始的交易就包含項目"A"，其後的 3 和 5 分別表示，第 1 至 3 筆交易包含"A"，第 4 到 8 筆交易不含項目"A"。又項目"C"的 Bit-Vector 為 01111111，將其前面補 0 之後為 001111111 其 CMRLE 編碼為 2*7，將第一個數值 2 減 1 等於 1，代表在原始交易資料庫中，第一筆交易"沒有出現項目 C"，第二個數值"7"代表第二筆到第八筆交易中"有出現項目 C"。注意，在解釋 CMRLE 時，我們以 Bit-Vector 解釋，但實際上，我們並不需透過 Bit-Vector 即可進行 CMRLE 編碼及運算。

(3) CMCn(Count Matrix Combination for n-itemset; n-項目組的個數)

這個部分紀錄包含某一特定項目的 n-itemset 個數。

舉例而言，從表 2 中我們可以看到第一列(與 A 相關)的 CMC2 欄位值為 5，代表項目"A"搭配其它一個不同的項目，出現在所有的交易紀錄中共 5 次。也就是說包含 A 的 2-itemset

有 5 個。我們紀錄 CMCn 的目的是希望能加快尋找高頻項目組的速度。

(二)利用 CM 矩陣中的 CMRLE 作"AND"運算

我們以 AND 運算的求得項目的支持度，如圖 1 所示，欲求項目組{AB}的支持度，我們取項目組 A 的 CMRLE 為 3*6，項目 B 的 CMRLE 為 3*3*3，進行 AND 運算，我們首先要計算出各個項目在交易中"有包含此項目"的位置。故依據 CMRLE 之定義得知，在交易中有包含項目組 A 的位置為 4 至 9，而在交易中有包含項目組 B 的位置為 4 至 6，接著檢查將項目組 A 與項目組 B 所找出的位置，是否擁有相同的位置，經由檢查知道項目組 A 與項目組 B 所擁有相同的位置為 4 至 6，表示 4 到 6 的位置共同出現項目組 A 與項目組 B。

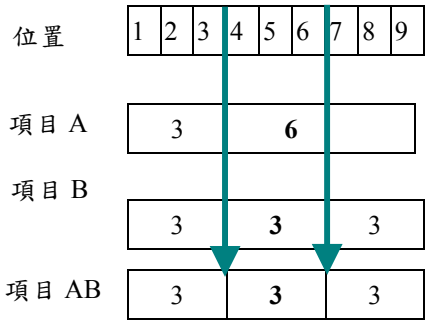

圖 1：CMRLE 做 AND 運算

經由上述之運算我們可以產生一個新的項目組 AB，其 CMRLE 為 3*3*3。並由 CMRLE 的定義可知偶數位數的數值表示在交易資料庫中" 包含此項目"的連續交易個數，故項目組 AB 在第三筆至第五筆共出現 3 次，表示 AB 的支持度為 3。

(三) 利用 CM 矩陣尋找高頻項目組

我們所提出的 CM 演算法，主要目的為找出所有高頻項目組，詳細的演算步驟如圖 2 所示。在本演算法中共包含了四個模組，各個模組的功能解釋如下：

(1) apriori_gen 模組：

本模組的功能主要為產生候選項目組，產生的方法為依據文獻[3]中所提的方法。

(2) checkCMC 模組：

本模組的功能為檢查在候選項目組中的各個 1-項目組的 CMCn 欄位值是否皆大於最小支持度，當任何一個 1-項目組的 CMCn 小於最小支持度時，則不對此候選項目組不可能滿足最小支持度，因此不進行 AND 運算。演算法如圖 3 所示。藉由本模組可以大量減少需要檢查的候選項目組的個數。

(3) AND 模組：

本模組的功能為對候選項目組中的所有 1-項目的 CMRLE 作 AND 運算，以求得此候選項目組的支持度。

```
輸入：CountMatrix；最小支持度，min_sup.
// Ck：候選 k-項目組集合，cand：候選項目組
// Lk：高頻 k-項目組集合，輸出：所有高頻項目組 L
L=φ;
L1=Find_frequent_1-itemsets(CM);
L=L ∪ L1
For(k=2; Ck-1 ≠ φ; k ++){
  Lk=φ;
  Ck=apriori_gen(Lk-1);
      For each cand ∈ Ck
       If checkCMC(CM, cand, min_sup) Then
            count = AND (CM,cand)
            updateCM(CM, cand, count)
            If count>min_sup Then
            Lk=Lk ∪ cand
            End If
        End If
      End For
  L=L ∪ Lk
End For
}
return L
```

圖 2：CM 找尋所有高頻項目組

(4) updateCM 模組：

本模組的功能為將 AND 運算後所產生的支持度，對候選項目組中的各個 1-項目之 CMCn 欄位值加以更新。其演算法如圖 4 所示。我們以一個候選項目組(AB)為例，利用圖 2 之演算法，檢驗其是否為高頻項目組，假設最小支持度為 3，首先先檢查項目 A、B 的

CMC2 欄位值是否皆大於最小支持度，從表 2 中可得知項目 A 的 CMC2 欄位為 5，項目 B 的 CMC2 欄位為 12，

```
//CM:CountMatrix,，cand:候選項目組
//I∈cand,即 I 爲 cand 中的一個 1-項目組
// CMCn(I)爲 I 項目組的 CMCn 的值
Boolean CheckCMC(CM, cand, min_sup)
{
  For each 1-item I in cand
      If CMCn (I) <min_sup Then
          return FALSE
          End IF
  End For
```

圖 3：檢查 CMCn 欄位值是否大於最小支持

```
//CM:CountMatrix,，cand:候選項
//目組
//count:支持度， I∈cand
updateCM(CM, cand, count){
  For each 1-item I in cand
     CMCn(I)= CMCn(I) -count
  End For
}
```

圖 4：更新 CMCn 欄位值

經圖 3 之演算法可以得知項目 A、B 皆大於最小支持度，故將 A、B 的 CMRLE 作 AND 運算求得項目組(AB)的支持度為 3，得知其符合最小支持度，故為高頻 2-項目組。在進行 AND 運算完後，我們需對項目 A、B 的 CMC2 欄位值進行更新。利用圖 4 之演算法進行更新後，項目 A 的 CMC2 欄位變為 2，項目 B 的 CMC2 欄位變為 9。

四、效能分析

在這裡我們將 CM 演算法的執行效率與 FP-Tree 演算法的執行效率做比較，以顯示 CM 演算法的特點，會以 FP-Tree 為比較對象是因為 FP-Tree 演算法是目前公認最快的演算法。

(一) 實驗設計

在實驗環境之建置部份，我們主要分成兩個部分來說明，第一部分為硬體環境，第二部分為測試資料庫之建置。

(1) 硬體環境

我們所有的實驗都在同一台機器上進行。本研究利用 Microsoft Visual C++ 6.0 發展 CM 演算法，硬體平台的規格為，一顆 Pentium 4 2G 中央處理器、1024mb 主記憶體、60GB SCSI 硬碟。測試平台之作業系統為 Windows XP 專業版。對於作業系統中不必要的常駐程式在測試之前都已關閉。所有的應用程式和服務程式也都結束，只留下內建之命令處理器(Command Processor)以供演算法程式開啟之用。

(2) 測試資料庫

在建立測試資料方面，均採用 IBM Almaden Research Center[9]所開發的人工資料產生器所產生的資料作為測試資料。我們依據 Apriori[2]文獻中所提到的方法來產生人造交易資料庫。這類的資料庫經常被用來當作測試資料用，除了比較有公信力之外，其測試的結果也能夠與前人的實驗作比較。人造資料在產生時有幾個參數可供設定，我們將產生資料庫的參數列於表 4。

表 4：人造資料庫

Ntrans	Tlen	Patlen	nitems	記作
|D|	|T|	|I|	N	Ta.Ib.Nc.Dd
原始資料庫中交易的數目	每筆交易中的平均長度	可能是高頻項目組集合數量的平均值	項目的數目	例如 T10I4N10kD1000k，表示此資料庫有 100 萬筆的資料，其中每筆交易為 10 筆，高頻項目組平均為 4 個，項目種類為 1 萬個。

(二) **實驗結果**

(1) 不同支持度的影響

圖 5 為資料量 90 萬筆下不同的支持度對演算法的影響，我們發現當支持度增加時，CM 演算法的效率比 FP-Tree 的方法顯著的好，原因是當支持度增加時，CM 演算法中的 checkCMC 模組發揮了大量減少候選項目組的效果，尤其支持度在 0.5%以上，CM 演算法幾乎以接近 0 秒的時間就可以找出所要的資訊。然而，必須指出的是 CM 的前置處理時間（建立相關矩陣的時間）佔了 150 秒，但一次前置處理的結果可供多次探勘之用，因此若以多次探勘而言，CM 演算法較 FP-Tree 有效。

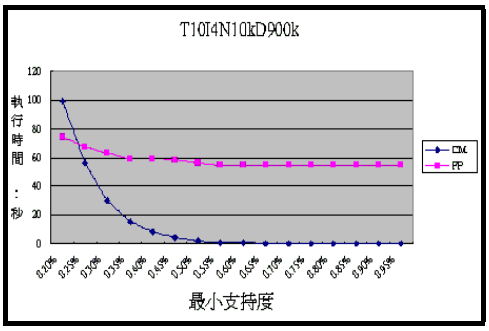

圖 5：資料量為 90 萬筆時之 CM 與 Fp-tree 演算法之效能比較

(2) 連續探勘效能比較

在這個比較中我們加入 Apriori 演算法，此 Apriori 演算法是採用 Christian Borgelt 之實作。我們藉由設定各種不同的最小支持度在各個演算法中作多次探勘，並將執行時間累加。圖 6 為 90 萬筆資料下的比較，圖 6 可以很清楚的看出利用我們提出的 CM 演算法在多次探勘的情況下，產生完整高頻項目組所需要的時間，比其它演算法更短。其中的前置處理時間是在尚未執行資料探勘時，建置 CM 時所需花費的時間，其它演算法沒有前置處理時間。

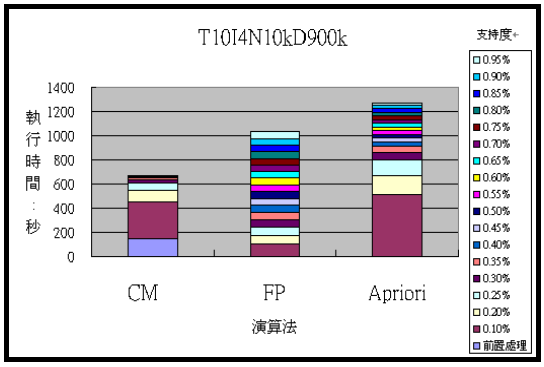

圖 6：在連續探勘下資料量為 90 萬筆之演算法效能比較

圖 7 為 140 萬筆資料下的比較，可以看出不管資料庫的大小，我們的 CM 演算法在多次探勘的情況下，比其它演算法更有效率。

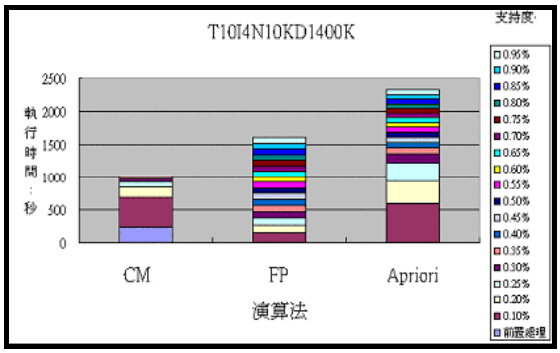

圖 7：在連續探勘下資料量為 140 萬筆之演算法效能比較

五、結論

在本論文中，我們提出一個新的資料探勘演算法 CM，此方法主要是利用變動長度編碼方法，將交易資料庫壓縮成少量的資料，然後直接在主記憶體中對壓縮後的交易資料進行資料探勘，而且任何時候都不需要再將交易資料解壓縮，解決了許多演算法在遇到資料庫太大時，必須將部份交易資料，存入磁碟機之中，而造成資料探勘速度的快速降低的問題。另外，利用掃描完一次資料庫將 CM 建置完成；此後，不管進行幾次不同支持度的資料探勘，都不需要再重新建置 CM，也就是說，多次資料探勘時，前置處理只需作一次。經過分析與實驗證實，我們的方法可以大量減少所佔的主記憶體空間，並且在多次探勘的情況下，優於目前的演算法。

誌謝: 本論文由國科會計畫編號 NSC 92-2213-E-011-076 支持，特此致謝

六、參考文獻

[1] 邱士軍(2002)，”關聯規則演算法之實作和效能評估”，國立台灣科技大學資訊管理研究所碩士論文。

[2] R. Agrawal, T. Imielinski, and A. Swami. “Mining Association Rules between Sets of Items in Large Databases,” Proc. of ACM SIGMOD, pages 207-216,May 1993.

[3] R. Agrawal, and R. Srikant, “Fast algorithms for mining association rules in large database,” Proceedings of the International Conference on Very Large Data Bases, pages 487-499, September 1994.

[4] S. J. Yen and A. L. P. Chen. “An Efficient Approach to Discovering Knowledge from Large Databases,” Proc. IEEE/ACM International Conference on Parallel and Distributed Information Systems (PDIS), 1996.

[5] Amitabha Das , Wee-Keong Ng , Yew-Kwong Woon, “Association Rule Mining: Rapid association rule mining,” Proceedings of the tenth international conference on Information and knowledge management October 2001.

[6] Pradeep Shenoy , Jayant R. Haritsa , S. Sundarshan , Gaurav Bhalotia , Mayank Bawa , Devavrat Shah, “Turbo-charging Vertical Mining of Large Databases,” ACM SIGMOD Record , Proceedings of the 2000 ACM SIGMOD international conference on Management of data, May 2000

[7] Han, J., Pei J. and Yin Y., “Mining Frequent Patterns without Candidate Generation,” Proceedings of ACM-SIGMOD International Conference on Management of Data, pages 1-12, May 2000.

[8] J. Pei, J. Han. H. Lu, S. Nishio, S. Tang, and D. Yang. “H-Mine: Hyper-structure Mining of Frequent Patterns in Large Databases,” Proc. The 2001 IEEE International Conference On Data Mining (ICDM’01), San Jose, California, November 29-December 2, 2001.

[9] IBM Data Mining Web Site: http://www.almaden.ibm.com/cs/quest/index.html