データマイニングにおける追加データの処理方式

2P－4

大森 匡　Zhang Ji　星 守
電気通信大学 大学院 情報システム学研究科*

1 はじめに

最近、データベースを用いた連想型ルールのマイニングが注目されており、Apriori （文献1）などの高速な算法が提案されている。しかし、これらの算法はいずれも静的な（変更されない）データベースを対象としている。一方で、実際のデータベースでは データが増加していくことが普通である。本稿では、こうした「増加していくデータベース」において Apriori 法を効率良く適用する方式を提案する。

2 ルール発見問題

はじめに連想型ルール発見問題と Apriori 法を図1を用いて説明する。

今、データベース DB のスキーマを [A1, A2, ..., An] として 各属性がある事象の発生の有無を示すブール値としよう。すなわち、DB のタプルは「命題 A1, A2, ..., An のうち どれが同時に成立したか」を示すログレコードであり、DB は 大規模なログデータベースである。（図1では タプルは 「同時に成立した属性のリスト」で表現されている。例えば、タプル ABCER は「事象 A,B,C,E,R が同時に発生した」という記録を表す。）また、以下では DB における属性組 Ax Ay ... Az の出現回数 (サポート数) とは、「DB において属性 Ax、Ay、...、Az が全て1であるようなタプルの総数」のことである。このとき、連想ルール発見問題とは、サポート数が θ 以上となるような 属性組 (ItemSet) を DB に対し全て求め、そこから「A,B なら C である確率は α 」という形のルールを発見することである （図1）。（θ は minsupport と呼ばれ、minsupport より出現回数の大きい属性組を Large ItemSet、それ以外の属性組は Small ItemSet と呼ぶ。ルール自体は Large ItemSet から条件つき確率の式で求められる。）

一般に ルール発見問題では k 個の属性からなる属性組について そのサポート数を DB をスキャンして数え、$k = 1$ から順に 何回か DB をスキャンして全ての Large Itemset を見つける。Apriori 法では、「ある属性組 *ItemSetX* の中で、もし、一つかそれ以上の部分集合が Small Itemset であれば、*ItemSetX* 自体も Small Itemset である」という原則を使い、数えるべき属性組を限定していく。

*Mining Association Rules from Incremental Database
J.Zhang,T.Ohmori,M.Hoshi (U.Electro-Comm.)

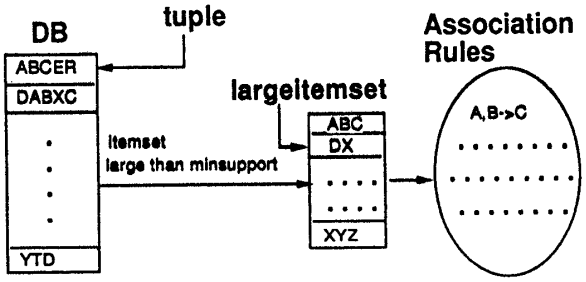

図 1: マイニングの流れ

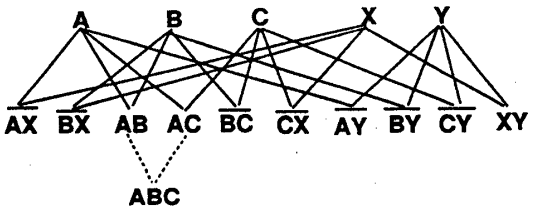

図 2: Apriori

図2に Apriori が終了した時の属性組の例を示す。図中、上に線がついている属性組は Small ItemSet であり、線がついていない属性組は Large ItemSet である。属性組の間の直線は、上位の 属性組 ItemSetX と ItemSetY を結合 (i.e. 和集合をとること) して得られた属性組 ItemSetXY について、それを DB スキャンの時に数えることを示している。点線は「上位の属性組を結合して得た属性組 $IS1$ の部分集合の中に Small ItemSet が入っているので $IS1$ は数えない」を示す。（図2では、属性組 ABC は BC が small だから 数えない。また、AX,BY などは small であるため、これ以上 結合しない。） このように Apriori では サポート数を数えた Small Itemset に対し、それを真に含む属性組は 数えられない。

3 追加データの処理方式

それでは データベース DB に対し、タプルを追加する (DBnew :=DBold+ΔDB) した場合を考えよう。このとき、DBold においては Small Itemset だった属性組が DBnew では Large Itemset になりうるため、これを見つけなければならない。特に、Apriori では 一旦 Small ItemSet と判断すると そこから後の属性組のサポート数

を数えないわけで、追加データ処理のときは こうしたDBoldにおけるSmall ItemSet $X1X2..Xn$ を DBnewにおいて数え直す必要がある。このとき、DBoldを全スキャンするのは 増分が非常に微量であれば 不利である。すなわち、元のデータベースを前処理しておき、新しいLarge Itemsetを見つける時にはスキャンするデータベース領域を限定すべきである。 以下では 数え直しのアクセスパタンに応じてDBoldを分解してアクセス量を減らす方法を提案する。これはDBoldへの前処理方法 (3.1節の圧縮とDB分割) とデータ追加時の処理(3.2節) とにわけられる。

3.1 前処理 – DBの圧縮と分割

前処理1 – DBの圧縮と出現回数の記憶

最初にAprioriをDBに適用した際、次の情報を記憶しておく。

1：次のような {Large Itemset LX, Counter} の組を記録しておき、DBから余分なタプルを削除する。すなわち、CounterとはDBoldにおいてLarge Itemset LX に含まれる属性にのみ1が立っているようなタプルの数である。この Counterが記録されると、これらのタプルは DBoldから除いても問題ない。 DBoldにおいて残ったタプル集合をRDB (Remaining DB) としておく。

2：この他に、DBoldにおいて サポート数を数えた結果Small Itemsetと判断された属性組についても [属性組, サポート数] を記憶しておく。

前処理2 – Signatureによる DB分割

次にRDBを いくつかのグループに分割し、追加データ処理のときにRDBのアクセス領域を限定する。一般に追加データ ΔDBが生じた時、ΔDBを一度スキャンして前処理1の情報を使うと 新しいLarge ItemSetが見つかる場合がある。問題となるのは、この新しい Large Itemset $NewLx$ がDBoldではSmall ItemSetであったため、$NewLx$ を真に含む属性組 $X1...Xn$ をDBoldで数えていないことである。実際、$X1...Xn$ のサポート数はDBoldで数え直さないといけない。そこで、RDBへのアクセス量を削減するためにRDB を シグニチャを使って分割しておく。すなわち、

1: タプル t で 1が立つ属性を Y1...Yn としたとき、そのシグニチャを sig(t) = h(Y1) + ... + h(Yn) とする。DBの属性数をNとすると、tはN-bit長レコードであり、signatureは k-bit長で生成する。 (h はk-bit長レコードを生成するハッシュ関数で、+は ビット単位のOR演算。)

2: 次に、RDBを sig(t) の値に応じて 2^k 個のクラスタへ分割しておく。 (各クラスタは同じsig(t) 値を持つタプル集合。)

こうすると、追加データ処理時に 属性組 $X1...Xn$ のサポート数を RDBで数えるときには QueryMask($X1...Xn$) = h($X1$)+...+h(Xn) を計算し、 QueryMaskにヒットするシグニチャを持つクラスタ (QueryMaskで1が立つビット位置に 1が立っているようなクラスタ) だけをアクセスすれば良い。

この利点は、RDBを互いに素なクラスタへ分割しておけることである。特に 属性間で相関度が強いため、シグニチャの生成方法をLarge ItemSetの出方に応じて決めるとうまくいきそうである。

極めて単純な例をあげると、図2において DBの全属性が [A,B,C,X,Y] に対し極大なLarge ItemSetは AB、AC、XYであるから sig(t) を [tがABを含むか, ACは?, XYは?] の3-bit長シグニチャとして、 RDBを [000] から [111] までの 8クラスタへ分解しておくことが考えられる。こうすると、属性組 ABCの数え直しに際しては クラスタ [110] だけをアクセスすれば良い。

一般に データベースにおいては 属性の総数Nが 10^5 程度と非常に大きいが、 各タプルの1が立つビット数は10程度と小さいため シグニチャ化するには都合が良い。本節で提案した方式ではシグニチャのビット数 kとして 2^k 個に DBをクラスタ分割するが、$k = 10$ としてクラスタ総数が1000個でヒット率1%程度なら見合うというものである。

3.2 追加データに対する処理

以上に述べてきた前処理の後、 追加データ ΔDBに対し 次の手順で処理を行なう：

手順1： ΔDBをスキャンし、前の (DBoldにおける) Large Itemset に対しサポート数の増分を計算しDBnewでのサポート数を得る。

手順2： DBoldにおいてサポート数を数えたSmall Itemsetに対し、ΔDB をスキャンして そのサポート数を更新する。これにより新しい Large Itemsetが見つかる。

手順3: 手順2で見つかった新しいLarge ItemSet を各々 NewLx1, ..., NewLxNとする。これら属性組から始めて RDB + ΔDBに対しAprioriを実行する。このとき、ある属性組 $X1...Xn$ を DBoldで数えていなかった場合、前処理2で作成したRDBのクラスタに対しQueryMask($X1...Xn$) を作成して これにヒットしたクラスタを使ってAprioriを実行する。

手順4: 追加データに対する処理を完了した後、次回処理のために $DBold + \Delta DB$ に前処理1、2を行なう。

4 まとめ

本稿では 連想型ルール発見において追加データを扱う算法を提案した。具体的には、シグニチャを使って元のデータベースを分割することでAprioriの再計算負荷を削減した。シグニチャの作り方自体は Large ItemSetの出方、つまり データベースの分布に応じた設計項目であり、現在 試験的な評価を行なっている。

参考文献 1： Agrawal他. *"Fast Algorithms for Mining Assocaition Rules"*. Proc. 20th VLDB 94.