TID List를 이용한 빈발항목의 효율적인 탐색 알고리즘

고윤희, 김현철
고려대학교 컴퓨터교육과
e-mail:{unygo, hkim}@comedu.korea.ac.kr

An efficient algorithm to search frequent itemsets using TID Lists

Younhee Ko, Hyeoncheol Kim
Dept. of Computer Science Education, Korea University

요 약

연관규칙 마이닝과정에서의 빈발항목 탐색의 대표적인 방법으로 알려진 Apriori 알고리즘의 성능을 향상시키기 위한 많은 연구가 진행되어 왔다. 본 논문에서는 트랜잭션 데이터베이스(TDB)에서 생성되는 각 패스의 k-itemset들에 대해 각각 트랜잭션 ID List(TIDList)를 유지하고 이를 이용해 (k+1)-itemset을 효율적으로 찾아내는 방법을 제안한다. 이 방법은 frequent (k+1)-itemset(k>0)의 빈도수 및 TIDList를 TDB에 대한 스캔이 전혀 없이 k-itemset의 TIDList로부터 직접 구한다. 이는 빈발항목집합을 찾기 위한 탐색 complexity는 크게 줄여줄 뿐 아니라 시간 변화에 따른 빈발항목집합의 분포 정보를 제공해 준다.

1. 서 론

데이터마이닝은 대용량의 데이터로부터 유용하게 활용될 수 있는 지식을 효과적으로 찾아내는 지식 탐사의 한 연구분야이다. 이렇게 수많은 데이터에서 숨겨진 패턴을 탐사하는 연구 중 가장 대표적인 것이 데이터들간의 연관 규칙을 찾아내는 것이다. 일반적으로 연관 규칙을 생성하는 과정은 크게 빈발항목(frequent itemset)을 찾아내고, 이들로터 연관 규칙을 생성하는 과정으로 구성된다. 빈발항목 탐색의 대표적인 방법으로 알려진 Apriori 알고리즘은 다음과 같은 성질을 이용하여 빈발항목 탐색의 효율성을 높였다[3]: "빈발항목집합들의 모든 부분집합 또한 반드시 빈발항목이 된다." 그러나 이러한 Apriori 알고리즘은 각 패스에서 생성된 후보항목집합(candidate itemset)들 중에서 빈발항목집합을 찾아내기 위해 매번 계속적으로 방대한 양의 트랜잭션 데이터베이스를 스캔해야 하며, 이는 연관 규칙 생성의 전 과정에 있어서 가장 큰 비중을 차지하는 부분이다. 따라서 이를 줄이고자 하는 많은 연구가 이루어지고 있다. 이러한 연구들은 해쉬 테이블을 이용하여 후보항목집합의 크기를 줄이거나[1], transaction reduction[2], data partitioning[4], data sampling[5] 등과 같이 데이터베이스의 스캔을 최소화시킴으로써 전체적인 성능을 향상시키고자 하였다.

본 논문에서는 트랜잭션 ID 리스트(TIDList)를 이용하여 빈발항목집합을 효율적으로 찾기 위한 방법을 제안한다. 이 알고리즘은 Apriori와 같은 level-wise 탐색을 수행하지만 데이터베이스의 스캔이 필요없이 frequent k-itemsets과 TIDList만 가지고 효과적으로 frequent (k+1)-itemsets을 구한다. 2장에서는 제안하는 알고리즘을 소개하고 향상된 시간 복잡도를 보이고 3장에서 결론을 제시한다.

2. TIDList를 이용한 빈발항목 생성 알고리즘

본 논문에서 제안하는 알고리즘은 각 빈발항목에 대해, 이들이 발생한 트랜잭션의 ID로 구성되는 TIDList을 만들고 이들의 비교를 통해 반복적으로 다음 단계의 빈발항목집합을 발견한다. I= $\{i_1, i_2, \cdots, i_m\}$을 데이터 아이템들의 집합이라고 가정하고, TDB는 트랜잭션 T의 데이터베이스를 의미하며, 이는 데이터 아이템들의 집합으로 구성된다.($T \subseteq I$) 각 트랜잭션은 고유한 트랜잭션 ID인 (TID)에 의해 구별되며, TID는 각 트랜잭션이 발생한 시점에 따라 시간순서를 가지고 부여된다고 가정한다. L_k는 frequent k-itemset들의 집합을 의미하고, S_A는 A라는 항목의 빈도수를 의미하며, minsup는 사용자에 의해 주어지는 최소 빈도수를 의미한다. $TIDList_A$는 item A를 포함하는 트랜잭션 ID들의 집합으로써 ID에 의하여 순차적으로 정렬되어 있다. 이는 처음 생성되었던 TDB를 중심으로 빈발항목집합을 효율적으로 찾아내기 위해 제안하는 데이터 구조로써, 각 패스에서 생성된 빈발항목집합의 빈도수 이외에 각 항목이 나타난 TID를 하나의 리스트 구조로 유지하고 있다. 이러한 새로운 데이터 구조의 생성을 통해 앞으로 진행될 다음 패스에서는 원래의 데이터베이스에 접근할 필요 없이 새로운 데이터 구조 사이의 비교만으로 빈발항목집합을 구한다.

2.1 알고리즘 및 수행 과정

본 논문에서 제안하는 알고리즘은 아래 그림과 같이 작동한다.

```
Ck = apriori_gen(Lk-1);
for_all p ∈ Ck  p is a powerset of A, B
CompareTIDList( TIDList_A, TIDList_B)
{
   TIDList_A(i) ∈ TIDList_A (0<i≤SA);
```

```
TIDList_B(j) ∈ TIDList_B (0<j≤S_B);
cur_support = 0, count = 0;

first_A, first_B are TIDList_A(1), TIDList_B(1);
last_A, last_B are TIDList_A(S_A), TIDList_B(S_B);
front diff = | first_A - first_B | ;
last diff = | last_A - last_B |;

if (front diff <= last_diff)                    ---------- ①
   Start to compare TIDLists from the front;
else
   Start to compare TIDLists from the back;

while( TIDList_A or TIDList_B doesn't reach the end of the list)
  {
    if( min(S_A, S_B) == minsup &&
                found the different TID from two TIDList)
                remove p from C_k                          ③
    if ( last_A < first_B || last_B < first_A)
                remove p from C_k                          ②
    if( ( minsup - cur_support) < the number of TID
    remained to compare)
                remove p from C_k                          ④
    count ++;
    if( TIDList_A(i) ≠ TIDList_B(j))
                compare next TID ;
    else {
          go to next TID in TIDList_A;
          go to next TID in TIDList_B;
          cur_support++;
        }
  }
}
```

1. TDB를 스캔하여 1-itemset에 대한 support를 구함과 동시에 각 항목에 대하여 그 항목이 포함된 TID를 유지하는 TIDList를 생성한다. 새롭게 생성된 모든 TIDList의 전체 크기는 TDB의 크기보다 작게 되는데 이는 1-itemset을 생성하는 과정에서 실제 minsup를 만족시키지 못하는 항목들을 모두 제거하므로 빈발항목에 대한 정보만이 존재하게 되기 때문이다. 또한 트랜잭션 자체가 발생한 시간에 따라서 TID가 부여되므로 이를 스캔하여 생성된 TIDList는 자연스럽게 TID에 의해 오름차순으로 정렬되어 있게 된다. 예를 들어, [그림1]에서의 TDB의 경우 1-itemset {A}의 $TIDList_A$는 {*T1, T2, T5, T6, T7*}가 되며, 이 항목집합의 빈도수 S_A는 $TIDList_A$의 크기인 5가 된다.

2. 생성된 1-itemset의 TIDList의 교집합을 구해 2-itemset의 새로운 TIDList를 생성시킴과 동시에 빈도수를 구한다. 이는 기존의 Apriori 알고리즘이 모든 2-itemset ∈ C_k에 대해 전체 TDB를 스캔하여 빈도수를 구했던 것에 비해 time complexity를 상당히 줄여준다. 제안된 알고리즘은 두 TIDList의 교집합을 구하는 과정을 보여준다. 예를 들어, [그림 1]에서 itemset {A}와 {F}로부터 2-itemset {A, F}를 구할 때에, $TIDList_A$와 $TIDList_F$의 교집합을 구하기 위한 최대 비교 횟수는 다음과 같다.

$$S_A + S_F \qquad \text{---------- (1)}$$

그러나, 실제적인 비교 회수는 아래의 휴리스틱 방법을 사용하여 훨씬 더 줄어든다.

① k-itemset의 TIDList를 사용하여 (k+1)-itemset의 TIDList를 구할 때, 두 TIDList의 가장 앞의 TID값과 가장 뒤의 TID값들을 비교하여 이들의 차이가 적은 쪽부터 비교를 시작한다.

② 두 항목의 TIDList의 비교시, $last_A < first_B$, $last_B < first_A$일 경우는 두 TIDList의 교집합이 전혀 생성될 수 없는 상황이므로 여기서 생성되는 후보 항목은 C_k로부터 제거된다.(∵이들은 TID에 대해 오름차순으로 정렬되어 있으므로)

③ 두 항목 A, B에 대한 TIDList를 비교하여 교집합을 구할 때, $\min(S_A, S_B)$ = minsup이라면, 두 TIDList의 비교시 다른 TID가 발견되는 순간, 더 이상 비교하는 것을 멈추고, 후보항목집합인 {A, B}를 C_k으로부터 제거한다.(∵이는 TIDList의 TID가 다른 것이 발생되는 순간 TIDList의 교집합의 수 < minsup 이 되므로 minsup을 만족시킬 수 없으므로)

④ 두 항목의 TIDList를 비교시, 앞으로 비교해야할 TID의 수가 *minsup - cur_support*보다 작다면 더 이상 TIDList를 비교하지 않고 이로부터 생성된 후보항목을 C_k로부터 제거한다.

3. 위의 과정을 거쳐, 생성된 k-itemset의 TIDList를 중심으로 이 과정을 반복하면서 다음 패스의 (k+1)-itemset의 TIDList를 구한다.

위의 알고리즘에 의해 수행되는 구체적인 예는 아래 [그림 1]과 같다. 1-itemset를 구하기 위해 Apriori-gen을 이용하여 후보항목집합을 구하는 것은 Apriori와 동일하나 각 항목의 빈도수를 구하기 위해 TDB를 스캔함과 동시에 빈발항목집합을 구하고 이들이 발생한 TID를 저장한다. 이를 기본으로 마찬가지로 2-itemset의 후보항목집합을 구하고 이의 빈도수를 찾기위해 전체 TDB를 스캔하여 찾는 대신 1-itemset의 TIDList를 비교하여 교집합을 구함으로써 2-itemset의 빈도수는 물론이며 TIDList까지 쉽게 구해낼 수 있다. 즉, 위의 예에서 S_{AF}를 구하기 위해 우선, $TIDList_A$= {*T1, T2, T5, T6, T7*}과 $TIDList_F$ = {*T1, T3, T4*}의 교집합을 구하게 되는데, 알고리즘 2-①에 의해 (T_1-T_1=0) < (T_7-T_4=3)이므로 두 TIDList의 앞에서부터 비교하기 시작한다. 두 TIDList의 첫 번째 원소가 T_1=T_1이므로 cur_support를 증가시키

고, 다음 원소를 비교한다. $T_2<T_3$이므로 $TIDList_A$의 다음 원소와 비교한다. 즉 $T_5<T_3$이므로 이번엔 $TIDList_F$의 다음 원소를 비교하게 되고 $T_4<T_5$임과 동시에 $TIDList_F$의 마지막 원소에 도달했으며, S_{AF} = 1이므로, 더 이상 비교하는 것을 멈추고 항목집합 {A, F}를 후보 항목집합으로부터 제거한다.

minsup=3일 경우 I

TDB

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

1-itemset TList DB

frequent itemset	TID List	Support
A	T1, T2, T5, T6, T7	5
B	T2, T3, T4, T5, T6, T7	6
C	T1, T2, T3, T6, T7, T8	6
F	T1, T3, T4	3

2-itemset TList DB

frequent itemset	TID List	Support
AB	T2, T5, T6, T7	4
AC	T1, T2, T6, T7	4
BC	T2, T3, T6, T7	4

3-itemset TList DB

frequent itemset	TID List	Support
ABC	T2, T6, T7	3

[그림1] TIDList을 이용해 빈발생성항목을 생성하는 과정 (minsup = 3 일 경우)

2.2 Time Complexity

TDB에 있는 트랜잭션의 수를 m, 각 트랜잭션이 포함하고 있는 평균적 항목 수를 n 이라고 하자. 이 때 후보항목집합으로부터 빈발항목집합인 k-itemsets을 구하기 위해 Apriori의 경우 $m \cdot n \cdot k$ 만큼의 비교를 하게 된다. 특히 m의 크기가 매우 크다는 사실을 감안한다면, 실제 한 항목을 찾기 위해 전체 TDB를 스캔하는 것은 매우 비효율적이다. 또한 전체의 비교 횟수는 k의 값이 증가함에 따라 이에 비례하여 증가하게 되므로 실제적인 비교횟수는 패스가 깊어짐에 따라 함께 증가하게 된다.

본 논문에서 제안하는 알고리즘은 실제 각 후보항목집합에 존재하는 아이템들에 대해 이들이 발생한 트랜잭션들만 비교하게 되므로, 실제 비교 횟수는 Apriori보다 줄어든다. 또한 이 알고리즘은 k의 값에 독립적으로 언제나 두 k-itemset의 TIDList의 비교만을 통해 (k+1)-itemset의 TIDList을 구한다. 따라서, 두 item A, B에 대한 TIDList의 비교 회수는 최악의 경우에 S_A+S_B가 된다.(by (1)) 그러나 실제적인 비교 횟수는 위에 제안된 여러 가지 휴리스틱 방법을 이용하여 훨씬 더 줄어든다. 두 개의 k-itemset A와 B로부터 (k+1)-itemset {A, B}를 생성한다고 할 때 총 비교횟수는 다음과 같다.

$$\text{총비교횟수} < S_A+S_B \ll m \cdot n \cdot k$$

2.3 시간변화에 따른 빈발생성 항목의 분포

각 패스에서 생성된 빈발항목집합에 대한 TIDList를 유지하는 것은 다음 패스의 빈발항목집합을 효율적으로 구할 수 있도록 하며 또한 각 빈발항목이 시간에 따라 발생하는 분포 경향을 알려준다. 즉 예를 들어 특정 빈발항목의 TIDList를 살펴본 결과, TID의 분포가 특정 부분을 중심으로 집중적으로 분포되어 있거나 혹은 전반적으로 넓게 분포되어 있는지의 여부를 가지고 이러한 빈발항목의 단순한 연관 관계 뿐만 아니라 시간 변화에 따른 빈발항목이 생성 관계를 유추해 낼 수 있다. 즉, $TIDList_{BCD}$ = $\{T_1,T_2,T_3,T_{50},T_{51},T_{52},T_{100},T_{101}..\}$ 와 같다면 항목{B, C, D}는 특정 시간 주기를 중심으로 발생하는 빈발항목이라는 부수적인 정보를 제공해 준다. 즉 이 방법은 단순히 빈발항목을 발견해주는 것 이외에 각 빈발항목이 시간에 따라 발생하는 경향 즉, 시간적 정보를 알려주고 이를 유용하게 사용할 수 있다.

3. 결론 및 향후 연구과제

본 논문에서는 빈발항목집합을 발견하기 위한 효율적인 알고리즘을 제안한다. 이는 각 패스에서 발생한 k-itemset들의 TIDList를 가지고 (k+1)-itemset의 빈도수와 TIDList를 구한다. 이는 1-itemset을 생성시에 단 한번 TDB를 스캔하고 그 이후로는 오직 TIDList의 비교만으로 빈발항목집합을 구한다. TDB의 크기가 매우 크다는 사실을 고려한다면, 이에 대한 접근을 줄이는 것만으로도 빈발항목집합을 찾기 위한 time complexity는 상당히 줄어들게 된다. 또한 이는 단순히 빈발항목집합만을 제시하는 것이 아닌 각 빈발항목들이 시간에 따라 어떠한 경향을 가지고 발생하는지에 대한 분포 정보를 제공해 준다.

향후에는 실제 데이터베이스의 내용이 계속적으로 업데이트되는 환경 하에서, TIDList을 이용하여 새로이 생성되거나 기존의 빈발 항목으로부터 제거되는 항목들을 효율적으로 관리할 수 있는 방법에 대한 연구가 필요하다.

4. 참고 문헌

[1] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data, San Jose, CA, May 1995.

[2] D. W. Cheng, J. Han, V. T. Ng, and C. Y. Wong . Maintenance of discovered association rules in large databases: An incremental updating technique. In Proc. 12th Int. Conf. on Data Engineering, New Orleans, 1996.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data, 207-216, May 1993.

[4] A. Savasere, E. Omiecinski, and S. Navathe, An effective algorithm for mining association rules in large databases, In Proc. 21th Int. Conf. on Very Large Data Bases, pp. 432 - 444, Zurich, Swizerland, 1995.

[5] H. Toivonen, Sampling large database for association rules," In Proc. 22nd Int. Conf. on Very Large Data Bases, Mumbai, India, 1996.