연관 규칙 탐사를 위한 효율적인 자료 구조

권경희, 정균락
홍익대학교 컴퓨터공학과
{khkwon, chong}@cs.hongik.ac.kr

Efficient Data Structure for Mining Association Rules

Kyung-Hee Kwon Kyun-Rak Chong
Dept. of Computer Engineering, Hongik University

요 약

정보화 시대에 정보의 양이 폭발적으로 증가함에 따라 데이터 마이닝(Data Mining) 또는 데이터베이스에서의 지식 발견이라 불리는 분야가 새로운 정보기술의 활용방법으로 대두되었다. 데이터 마이닝의 한 기법인 연관 규칙 탐사를 위한 자료 구조로 그 동안 해쉬 트리, prefix 트리, 이진 트리 구조 등이 제안되었다. 본 논문에서는 연관 규칙 탐사를 위한 효율적인 자료 구조를 제안하고 실험을 통해 해쉬 트리보다 그 성능이 우수함을 보였다.

1. 서론

연관 규칙(association rule) 탐사 문제는 일반적으로 두 단계로 빈발 항목 집합(large itemset)을 찾는 것과 빈발 항목집합을 사용하여 데이터베이스로부터 최소 신뢰도(confidence)를 만족하는 연관 규칙을 생성하는 것이다. 빈발 항목 집합을 구하기 위해서는 후보 항목 집합(candidate itemset)을 먼저 구하고 데이터베이스를 스캔하면서 빈도수를 구하고 지지도(support) 이상인 항목들만을 선택하게 된다. 이 과정에서 후보 항목 집합은 탐색 연산이 많게 되고 효율적인 자료 구조에 저장되어야 하는데 [2]에서는 해쉬 트리를 사용하였다. A. Kitada는 [4]에서 이진 트리 구조를 제안하였고, L Shen은 [5]에서 prefix 트리를 제안하였다. 해쉬 트리나 prefix 트리의 문제점은 각 노드가 자식노드를 가리키기 위한 포인터를 가지고 있는 데 포인터의 수가 아이템의 수에 비례하게 되어 필요한 메모리의 양이 증가하게 된다. 이진 트리 구조[4]는 공간의 낭비는 줄였지만 노드 탐색시간이 늘어나는 단점을 가지고 있다.

본 논문에서는 연관 규칙 탐사를 위한 효율적인 자료 구조를 제안하였고 실험을 통하여 해쉬 트리 구조와 성능을 비교하여 우수함을 보였다.

2. 연관 규칙의 정의

연관 규칙이란 '어떤 사건이 일어나면 다른 사건이 일어난다'와 같은 연관성을 말하며, X->Y의 형식을 가진다.

$I = \{i_1, i_2, ..., i_k\}$를 항목집합이라 하자. 트랜잭션들로 이루어진 데이터베이스 D가 존재할 때, 각 트랜잭션은 고유한 트랜잭션 번호(TID)가 부여된다. X->Y 형식에서 $X \subset I$, $Y \subset I$이고 $X \cap Y = \emptyset$이다. 연관 규칙은 지지도와 신뢰도를 바탕으로 트랜잭션 데이터베이스에서 각 항목간의 연관성을 찾는 것을 의미한다. 지지도는 전체 트랜잭션에 대한 X와 Y를 포함하는 트랜잭션의 비율을 말하고 신뢰도는 X를 포함하는 트랜잭션에 대한 Y를 포함하는 트랜잭션의 비율을 말한다. 사용자가 정한 최소 지지도를 만족하는 항목집합을 빈발 항목집합이라 한다. 연관 규칙 탐사는 주어진 지지도와 신뢰도를 만족하는 연관 규칙을 모두 생성하는 것을 말한다.

3. 해쉬 트리 구성

연관 규칙 탐사를 위한 대표적인 알고리즘으로는 Apriori 방법이 있다 [2]. 이 방법에서 빈발 항목집합으로 구성된 해쉬 트리는 해쉬 테이블과 항목집합을 포함하는 내부노드(internal node)와 잎노드(leaf node)로 구성된다. 내부노드의 해쉬 테이블의 각 버켓은 자식 노드를 가리키는 데, 깊이 d의 내부 노드는 깊이 d+1의 노드를 가리킨다. 후보 항목집합을 해쉬 트리에 추가할 때 루트로부터 잎노드에 이르기까지 깊이를 1씩 증가시키면서 트리에 저장한다. 깊이 d의 내부 노드에서 후보 항목집합의 d번째 항목을 해쉬 함수에 적용

하여 자식 노드를 결정한다. 해쉬 트리의 단점은 각 노드가 자식노드를 가리키기 위한 포인터를 가지고 있는 데 포인터의 수가 아이템의 수에 비례하게 되어 필요한 메모리의 양이 증가하게 된다. 또 다른 단점으로는 예를 들어 두 항목 AB와 AC를 해쉬 트리에 저장할 때, 루트에서 A는 같은 노드를 찾아가지만 B와 C는 해쉬 함수에 따라 각각 다른 잎 노드에 저장될 수 있다. 따라서 AB와 AC는 항목의 특성상 인접하면서도 다른 저장 장소에 위치하게 되고, 캐쉬 메모리를 효율적으로 사용할 수 없게 되어 실행시간에 영향을 미치게 된다.

4.. 제안된 Prefix 배열 구조와 탐색 방법

본 논문에서는 개선된 탐색 트리인 prefix 배열 구조를 제안하고, 탐색 방법으로 선형 근사법(linear approximation)과 이진 탐색법을 제안하였다.

n개의 항목들로 구성되어진 n-후보 항목집합을 C_n : $\{I_0, I_1, I_2, \ldots I_{n-1}\}$라고 정의할 때, 이들 각 항목 I_i은 중복되지 않은 항목이며 각 n-후보 항목집합의 저장순서는 0번째 항목 I_0부터 n-1 번째 항목 I_{n-1}까지 순차적으로 이루어진다.

탐색 트리는 n-패스의 n-후보 항목집합에 대한 레벨의 수가 n개이고, n개 항목에 대해 각 레벨에 저장이 되며 루트레벨에서 하위레벨로 하향하면서 구성되어진다. 루트 레벨을 L_0, L_i의 부모레벨을 L_{i-1}이 된다. 구성은 N개의 n-후보 항목집합들에 대한 각 i번째 항목 I_i은 서로 연속된 공간상에서 순차적으로 저장되고 i+1번째 항목 I_{i+1}의 인덱스를 가리키고 있다. 이때 n-후보 항목집합들의 항목 I_i은 항목 I_{i-1}이 같은 경우에만 연속적으로 저장이 이루어지고, 그렇지 않은 경우에는 항목 I_{i-1}이 다르다는 것을 표현하기 위해서 항목 I_{i-1}이 바뀌기 전의 후보 항목집합들의 i번째 항목 I_i들이 저장되어 있는 인덱스 뒤에 null로 지정한 후 인덱스를 증가시킨 다음에 바뀌어진 후보 항목집합의 항목 I_i을 저장하게 된다. 트리 구성시 선행하는 후보 항목집합과 비교하여 I_i가 바뀐다면 세 가지 작업을 거치는데 첫 번째 L_{i+1}부터 L_{n-1}까지 현재의 인덱스를 상응하는 각 레벨의 부모레벨 L_i부터 L_{n-2}에 저장을 하여 마지막 인덱스를 가리키게 한다. 다음으로 L_{i+1}에서 L_{n-1}까지의 인덱스를 1씩 증가하여 null을 저장한다. 마지막으로 저장해야할 새로운 후보 항목집합의 각 항목들을 L_0부터 L_{n-1}에까지 저장하게 된다. 그림 1에 다음의 4-후보 항목집합에 대한 트리 구성의 예가 나타나 있다.

항목의 집합 I = {A, B, C, D, E, F, G, H, I, J}

4-후보 항목집합 C_4 = { {A,B,C,D}, {A,B,C,E}, {A,B,C,F}, {A,B,D,E}, {A,B,E,H}, {B,D,E,G}, {B,D,E,H}, {B,D,E,J} }

C_n^i를 i 번째 n-후보 항목집합이라 하자. 먼저 4-후보 항목집합 C_4^0 인 {A, B, C, D}를 탐색 트리에 저장한다면, 탐색 트리의 L_0에 A의 인덱스가 B를 가리키도록 하고 L_1의 시작 인덱스에 B를 저장한다. 이와 같이 C, D도 각각 L_2, L_3의 시작 인덱스에 저장하며 앞의 항목이 다음 항목을 가리키도록 구성한다. C_4^1, C_4^2는 C_4^0와 A, B, C가 동일하므로 L_3에 저장되어 있는 D에 대해 연속적으로 E와 F를 저장하게 된다. C_4^3인 {A, B, D, E}는 C_4^2인 {A, B, C, F}와 비교했을 때 C가 D로 바뀌어지므로 L_3의 인덱스를 1 증가시켜 널로 지정한 후, L_2, L_3에 각각 D와 E를 저장한다. 따라서 제안된 탐색 트리의 구성을 살펴보면 n-후보 항목집합의 각 항목들은 서로 다른 레벨에 존재하게 되며 항상 다음 레벨의 인덱스를 가지고 있으므로 후보 항목집합의 빈도수를 계산하는 과정이나 후보 항목집합을 생성하는 작업에서 이러한 인덱스에 관한 정보를 가지고 수행하게 된다.

L 0

	0	1	2	3	4	5	6	7	8	9
	(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)	(J)
항 목	A		C							
시작 번호	0	-	2	-	-	-	-	-	-	-1
끝 번호	0		2							

L_1

	0	1	2	3	4	5
항 목	B	-	D	-		
시작번호	0		4			
끝 번호	2		4			

L_2

	0	1	2	3	4	5
항 목	C	D	E	-	E	-
시작 번호	0	4	7		10	
끝 번호	2	5	8		12	

L_3

	0	1	2	3	4	5	6	7	8	9	10	11	12	1
항목	D	E	F	-	E	F	-	G	H	-	G	H	J	-
빈도	0	0	0		0	0		0	0		0	0	0	

그림 1. prefix 배열 구성의 예

그림 2에 탐색 트리 구성 알고리즘이 나타나 있다.

탐색 트리의 접근 방법으로는 각 항목 I_i들은 같은 레벨 L_i에 순차적으로 저장되므로 이진 탐색법을 사용하면 된다

다른 방법으로는 후보들간의 간격을 고려하여 찾고자 하는 항목 x가 저장되어있는 인덱스를 추정하고, 그 장소로부터 순차적으로 찾아가는 방법이다. I_a 가 a가 저장되어 있는 인덱스이고 I_b 가 b가 저장되어 있는 인덱스일 때, 구간 (a, b)에서 x가 저장되어 있는 장소의 인덱스는 (1)식과 같이 추정될 수 있다.

$$I_x = \frac{I_b - I_a}{b-a}(x-a) + I_a \qquad (1)$$

```
/*    Cn = {I0, I1, I2, ... In-1}
C n^m : m번째 n-후보 항목집합   (단, n은 item의 개수)
  Li : 레벨 i,  Li 는 Ln-1의 부모 레벨
  Index(Li,Lj) : Li에서 Lj까지의 index들
  Compare( C n^m-1 , C n^m ) : m-1번째 후보항목집합과
                  m번째 후보 항목집합과 item 비교 */
1) insert each item of C n^0 from L0 to Ln-1 at Search Tree;
2) forall n-candidates Cn ∈ C do begin
      i = Compare( C n^m-1 , C n^m );
      if(i != n-1)  then
          insert Index(Li+1, Ln-1) from Li+1 to Ln-2;
          Index(Li+1, Ln-1)++;
          insert Null to Index(Li+1, Ln-1);
          insert C n^m to Index(L0, Ln-1);
      else
          Index(Li+1, Ln-1)++;
          insert C n^m to Index(L0, Ln-1);
        endif
      end
3) insert Index(L1, Ln-1) from L0 to Ln-2;
   Index(L1, Ln-1)++;
   insert Null to Index(L1, Ln-1);
```

그림 2. 탐색 트리 구성 알고리즘

5. 성능 측정 및 결과

본 논문에서 제안하는 prefix 배열 구조와 해쉬 트리는 512MB 주기억 장치를 가진 SUN 엔터프라이즈 3000에서 구현되어 성능을 비교하였다. 데이터는 마이닝에서 많이 사용되는 합성(synthetic) 데이터로 표1에 나타나 있다. 여기서 |D|는 데이터 베이스의 크기이고, |T|는 트랜잭션에서 항목의 평균 수, |I|는 최대 빈발 항목집합의 평균크기이다.

성능 측정은 표 1의 다섯 개의 실험 데이터베이스에 대해 최소 지지도를 2%에서 0.25%로 변화하면서 해쉬 트리와 탐색 트리에 대한 수행 시간 및 메모리 할당량을 측정하였다. 해쉬 트리는 각 레벨의 해쉬 테이블의 크기 변화에 따라 수행

데이타베이스이름	|T|	|I|	|D|	크기
T5I4D100	5	4	100,000	2.4MB
T10I4D100	10	4	100,000	4.4MB
T10I6D100	10	6	100,000	
T20I4D100	20	4	100,000	8.4MB
T20I6D100	20	6	100,000	

[표 1] 실험 데이터베이스 및 매개 변수 값

시간과 메모리 사용율이 크게 달라진다. 그림 4에 T20I6D100에 대한 수행시간이 나타나 있는데 선형근사법을 사용한 prefix 배열 구조가 가장 좋은 결과를 나타냈고, 그 다음에 이진 탐색법을 사용한 prefix 배열 구조로 해쉬 트리 보다 그 결과가 우수하였다.

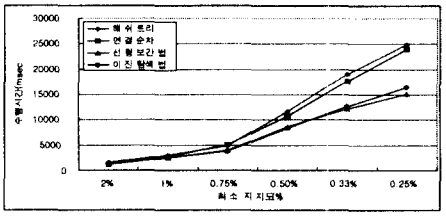

그림3 T20I6D100의 수행 시간

6. 결론

데이터 마이닝(Data Mining) 또는 데이터베이스에서의 지식 발견이 최근에 새로운 정보기술로 많이 활용되고 있다. 본 논문에서는 데이터 마이닝의 한 기법인 연관 규칙 탐사를 위한 자료 구조로 prefix 배열 구조를 제안하고 실험을 통해 해쉬 트리보다 그 성능이 우수함을 보였다.

7. 참고문헌

[1] M.-S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a Database Perspective", IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 866-883, Dec. 1996.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Database", Proceedings of the 20th International Conference on Very Large Databases, 1994.

[3] R. Agrawal and et al, "Programs Generating Test Data in data Mining", http://www.almaden.ibm.com/cs/quest, 1997.

[4] A. Kitada, etc. ,"A Data Structures for Dynamic Data Mining," 1999 Third Intl. Conf on Knowledge-Based Intelligent Info. Engineering System, PP530-533, 1999.

[5] L. Shen, H. Sha and L. Cheng, "New Algorithms for Efficient Mining of Association Rules, " the 7th symposium on the Frontiers of Massively Parallel Computation, pp 234-241, 1999