KIPS Tr. Software and Data Eng.
Vol.2, No.12 pp.855~864 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.12.855

A Study on the Implementation of SQL Primitives for Decision Tree Classification

An Hyoung Geun† · Koh Jae Jin††

ABSTRACT

Decision tree classification is one of the important problems in data mining fields and data minings have been important tasks in the fields of large database technologies. Therefore the coupling efforts of data mining systems and database systems have led the developments of database primitives supporting data mining functions such as decision tree classification. These primitives consist of the special database operations which support the SQL implementation of decision tree classification algorithms. These primitives have become the consisting modules of database systems for the implementations of the specific algorithms. There are two aspects in the developments of database primitives which support the data mining functions. The first is the identification of database common primitives which support data mining functions by analysis. The other is the provision of the extended mechanism for the implementations of these primitives as an interface of database systems. In data mining, some primitives want be stored in DBMS is one of the difficult problems.

In this paper, to solve of the problem, we describe the database primitives which construct and apply the optimized decision tree classifiers. Then we identify the useful operations for various classification algorithms and discuss the implementations of these primitives on the commercial DBMS. We implement these primitives on the commercial DBMS and present experimental results demonstrating the performance comparisons.

Keywords : Data Mining, Primitive, Decision Tree, Classification

판단 트리 분류를 위한 SQL 기초 기능의 구현에 관한 연구

안 형 근† · 고 재 진††

요 약

판단 트리 분류는 데이터 마이닝의 중요한 문제의 하나이고, 데이터 마이닝은 대형 데이터베이스 기술의 중요한 과제가 되고 있다. 그러므로 데이터베이스와 데이터 마이닝 시스템의 결합 노력은 판단 트리 분류와 같은 데이터 마이닝 기능을 지원하는 데이터베이스 기초 기능의 개발로 이어지고 있다. 이런 기초 기능은 분류 알고리즘의 SQL 구현을 지원하는 특수한 데이터베이스 연산들로 구현되며, 특정 알고리즘을 구현하여 데이터베이스 시스템의 구성 모듈로 사용하고 있다. 데이터마이닝 기능을 제공하는 데이터베이스 기초 기능의 개발에는 두 가지 관점이 있다. 하나는 데이터 마이닝 기능을 분석해서 그런 기능들을 제공하는 데이터베이스 공통 기초 기능을 확인하는 것, 다른 하나는 데이터베이스 시스템의 인터페이스의 한 부분으로 이런 기초 기능의 구현을 위한 확장된 메커니즘을 제공하는 것이다. 데이터마이닝에서 어떤 기초 기능들을 DBMS에 저장할 것인가는 어려운 문제 중에 하나이다.

따라서 본 논문에서는 이러한 문제를 해결하기 위하여, 최적화된 판단 트리 분류기를 만들고 데이터베이스 기초 기능에 대해서 기술한다. 판단 트리 분류 알고리즘의 유용한 연산들을 확인하고, 상업적 DBMS에서 이러한 기초 기능의 구현에 대해서 기술하고, 성능 비교를 위한 실험 결과를 제시한다.

키워드 : 데이터마이닝, 기초기능, 판단트리, 분류

1. 서 론

판단 트리 분류는 데이터 마이닝의 중요한 문제의 하나이고, 데이터 마이닝은 대형 데이터베이스 기술에서 중요한 위치를 차지하고 있다. 데이터 마이닝하기 위한 데이터는 대부분 데이터베이스 시스템(이하 DBMS)에 저장되고, 이 DBMS는 데이터 접근(access), 필터링(filtering), 인덱싱

※ 이 논문은 2010년 울산대학교 연구비에 의하여 연구되었음.
† 정 회 원 : 울산대학교 LINC사업단 연구교수
†† 정 회 원 : 울산대학교 전기공학부 교수
논문접수 : 2013년 5월 28일
수 정 일 : 1차 2013년 8월 19일, 2차 2013년 9월 23일
심사완료 : 2013년 10월 8일
* Corresponding Author : Koh Jae Jin(jjkoh@ulsan.ac.kr)

(indexing)하는 구현 기능들을 갖고 있다. 데이터 마이닝의 SQL 활용 기법은 대용량 데이터 처리, 병렬화, 필터링, 집계 기능 등과 같은 DBMS 기술을 주로 활용하고 데이터 자체뿐만 아니라 질의어 처리 결과를 마이닝 하는 것이 특징이다[1,11]. 그러나, 처리 성능이 낮아 조인, 그룹핑, 집계 같은 SQL 연산만으로 데이터 마이닝 기능을 수행하기에는 충분하지 않은 문제점이 있어 SQL 연산의 최적화를 위한 인덱싱 기법을 사용하기도 하고, 또한, 효율적인 구현을 위해서 데이터 마이닝 기능들이 DBMS에서 연산이나 접근 패턴 및 접근 경로 등의 지식을 활용하기도 한다. 이러한 상황에서 데이터 마이닝의 어떤 기능들이 DBMS로 저장될 것인지가 가장 어려운 문제로 되고 있다[12].

기존 연구들은 주로 판단 트리 분류를 이용하여 데이터 마이닝 기능들을 확인하고 DBMS의 구현 기능들을 이용하였으며, DBMS에서는 데이터 마이닝 기능들을 지원할 몇 가지 기술들을 서술하고 있다[2, 3]. 첫째는 연관 규칙에 대한 새로운 언어 구성을 SQL에 추가하는 것, 둘째는 데이터 마이닝을 위한 OLE DB 같은 특수한 API를 사용하거나 사용자 정의 타입과 메소드를 사용해서 데이터 마이닝 기능을 내부적으로 구현, 셋째는 DBMS가 데이터 마이닝에 유용한 특수한 연산자나 기초 기능을 제공하는 것 등이 있다. 이 모든 방법들이 데이터 마이닝 기능들에 유용하지만 본 논문에서는 상기 내용에서 기술한 문제점 해결을 위하여 특수한 연산자나 기초 기능의 세 번째 기술 관점에서 연구가 진행되었다[11].

따라서, 본 논문에서는 판단 트리 분류를 위한 기초 기능에 대해서 서술하고, AVC 그룹이나 CC 테이블 컴퓨팅 등을 상업적 DBMS의 SQL 연산자로 구현한다. 이 과정에서 최적화된 판단 트리 구축을 위한 노드 통계 질의어들의 평가는 부분 대응 질의어를 가속화하는 인덱스 기법들을 대상으로 하였다. 또한, 새로운 데이터에 대한 유도 분류 모델을 적용하는 연산자인 prediction join(예측 조인)을 구현하고, 데이터 마이닝을 위한 SQL 기초 기능에 기반한 추가적 연산인 분류 측도 계산 등을 기술한다. 본 논문에서는 채무 불이행자를 예상하기 위한 주제를 제시하고, 과정들의 이해를 돕기 위하여 Table 1의 훈련 집합(Training Set)과 Table 2의 시험 집합(Test Set)을 예시 데이터를 사용한다[4]. 채무 불이행자(defaults)를 예상하기 위하여 주택의 소

Table 1. Training Set

id	hh	mrg	inclev	defaults
1	y	unm	125,000(m)	n
2	n	mrg	100,000(m)	n
3	n	unm	70,000(l)	n
4	y	mrg	120,000(m)	n
5	n	div	95,000(m)	y
6	n	mrg	60,000(l)	n
7	y	div	220,000(h)	n
8	n	unm	85,000(l)	y
9	n	mrg	75,000(l)	n
10	n	unm	90,000(m)	y

Table 2. Test Set

id	hh	mrg	inclev	defaults
11	n	unm	55,000(l)	
12	y	mrg	80,000(l)	
13	y	div	110,000(m)	
14	n	unm	95,000(m)	
15	n	div	67,000(l)	

유 여부(hh)의 yes(y), no(n), 결혼 상태(mrg)의 미혼(unm), 기혼(mrg), 이혼(div), 연수입(inclev)의 경우 9만불 미만이면 하위(l), 9만불 이상 15만불 이하이면 중위(m), 15만불 이상이면 상위(h)로 하여서, 수치 데이터를 카테고리 데이터(category data)로 변환한 속성들로 구성하였다.

1장 서론에 이어 본 논문은 다음과 같이 전개된다. 2장에서 판단 트리 분류 개념과 기초 기능에 대해서 서술하고, 3장에서 판단 트리 기초 기능의 PL/SQL 구현, 4장에서는 질의 성능을 비교평가를 하고, 5장에서 결론을 내린다.

2. 판단 트리 분류 개념과 기초 기능

2.1 판단트리 분류 개념

분류(Classification)는 데이터 마이닝의 중요한 문제의 하나로서 수년간 연구되어 왔으며, 베이시안(Bayesian) 분류, 신경망(neural network), 회귀 트리(regression tree), 판단 트리(decision tree) 등과 같이 몇 개의 모델이 제시되었다. 그 중에서 판단 트리 분류가 단순하면서 이해하기 쉬워 가장 선호하는 모델이며, 판단 트리를 구축하는 알고리즘으로는 ID3, C4.5, SPRINT, SLIQ, PUBLIC 등이 있다. 대부분의 판단 트리 알고리즘은 greedy 접근 방법을 사용하며[5, 6, 12], 본 알고리즘은 트리 성장 단계(tree-growing phase)에서 근 노드의 전체 데이터 세트를 가지고 시작한다. 해당 데이터 세트는 분류 기준에 의해서 부분 집합으로 나누어지며, 이런 과정은 각 부분 집합이 같은 클래스(class)에 속하는 멤버(member)들만 가지거나 충분히 작은 부분 집합으로 될 때까지 각 부분 집합에 대해서 반복적으로 수행된다. 트리 가지치기 단계(Tree pruning phase)에서 구축된 트리는 과적합(over-fitting)을 방지하거나, 트리의 정확성을 높이기 위해서 부분적으로 잘려지게 된다. 트리 가지치기에 대한 중요한 접근 방법의 하나는 MDL(Minimum Description Length) 원칙에 기반 한다[7].

트리 성장 단계에서 분류 기준은 노드 파티션의 샘플들을 개별 클래스들로 가장 잘 분류하는 속성을 선택함으로써 결정되며, 이 속성이 노드의 판단 속성이 된다. 분류 속성이 A라면 판단 기준은 다음과 같다. A가 수치 속성이면 $A\ \Theta\ v(v \in dom(A)$, Θ는 비교 연산자)이고, A가 카테고리 속성이라면, $A \in V(V \subseteq dom(A))$이다. 가장 좋은 분류점을 선택하기 위해서 최적화된 측도로 ID3과 C4.5는 분류된 파티션들의 information entropy를 최소화하는 분류점을 선택하고, SLIQ와 SPRINT는 분류된 파티션들의 gini index를 최

소화하는 분류점을 선택한다. n개의 레코드들을 갖는 데이터 세트 S에 대해서 information entropy(E(S))는 $E(S) = -\sum p_i \log p_i$($p_i$: class i의 상대 빈도수)이다. 그리고 데이터 세트 S를 부분 집합 S1과 S2로 나누는 분류에 대해서 entropy E(S1, S2) = (n1/n)E(S1) + (n2/n)E(S2)(여기서, n1은 S1의 레코드 수, n2는 S2의 레코드 수이다.), 데이터 세트 S에 대한 gini index $Gini(S) = 1 - \sum p_i^2$(p_i : class, i : 상대 빈도수)이다. 데이터 세트 S의 S1과 S2로의 나눔에 대한 분류 gini index S는 다음과 같이 Gini-split(S) = (n1/n)gini(S1) + (n2/n)gini(S2)로 정의한다. 한 노드에서 판단 속성은 그 노드의 자식 노드에서는 고려되지 않는다.

판단 트리를 위한 알고리즘의 treegrowing procedure는 다음과 같다.

```
procedure treegrowing(dataset S)
  if all records in S belong to the same class
      return
  foreach attribute Ai
      Evaluate splits on attribute Ai
  use best split found to partition S into S1 and S2
  treegrowing(S1)
  treegrowing(S2)
```

판단 트리 구축에 있어서 많은 시간을 소비하는 부분은 분류점 선택이다. 구축된 활동 노드는 논리곱(and) 분류 조건을 만족하는 데이터의 부분집합(파티션, partition)과 그 노드의 부모 노드들이 구성되어야 하고, 각 노드의 속성은 분류 가능한 평가가 이루어져야 한다. 판단 트리에서 직접적인 데이터 접근은 앞서 서술된 측도에 기초한 가장 좋은 분류점 선택이 필요한 것은 아니며, 속성과 클래스의 결합이 일어나는 파티션의 레코드 개수에 대한 통계치는 필요하다. 이러한 통계치 정보는 속성이름(attribute name), 속성값(attribute value), 클래스라벨(class-label), 횟수(count)로 구성되는 단순 테이블로부터 얻을 수 있다. 구조는 CC 테이블 및 AVC group(Attribute-Value-Class)형태로 기술되며, 표현은 아래와 같이 SQL 질의어를 통해서 만들 수 있다[8, 9].

```
select 'att1' as attname, att1 as attvalue, cl as
class, count(*)
from S
group by att1, cl
union
select 'att2' as attname, att2 as attvalue, cl as
class, count(*)
from S
group by att2, cl
union
….
```

2.2 판단트리 분류 기초기능

DBMS에서 단일 스캔으로 통계 테이블을 만드는 것은 분류 기초 기능을 위한 좋은 후보가 된다. 분류 기준을 선택하는데 사용하는 측도들이 다르기 때문에 필요한 통계치를 계산하는 기초 기능이 다양한 알고리즘들을 지원할 수 있다. 한 노드의 파티션에 속하는 데이터들을 선택하는 질의어는 대부분 partial-match 질의어이며 조건들은 다음과 같다. $cond_1$ and $cond_2$ and… and $cond_m$, 여기서 $cond_i$는 A_j Θ v(Θ는 비교연산자, $A_j \in V$, $m \leq n$, n은 속성의 개수) 형태의 프레디키트(predicate)이며, 예로 C_4 클래스에 대한 partial-match 질의어는 A_1=2 and A_3=1로 표현할 수 있다. 대형 데이터 세트의 효율적인 평가를 위해서는 알맞은 접근 경로를 제공하는 속성들의 집합으로 구성된 인덱스가 필요하다. 따라서 특수한 인덱스 구조에 기초한 partial-match 질의어를 구현함으로써 한 노드의 파티션을 얻는 필터링 연산이 분류를 위한 기초 기능의 후보가 될 수 있고, 이를 통해 통계 기초 기능의 결과를 얻을 수 있다.

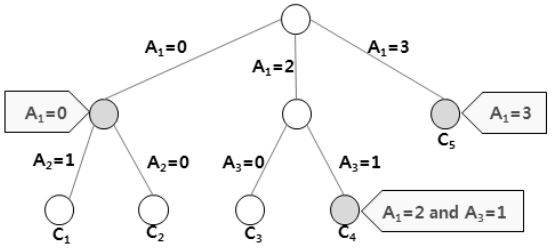

Fig. 1. Decision Tree example

판단 트리에서 MDL 가지치기는 트리를 상향식(bottom-up)으로 탐색하면서 수행되며, 만약에 노드 N에 기초한 최소 비용 부트리의 비용이 노드 N에서 직접적으로 레코드들을 엔코딩(encoding)하는 비용보다 같거나 크면, 노드 N의 자식들을 가지치기한다. 부트리의 비용은 재귀적으로 계산된다. 가지치기 단계에서 가장 비용이 많이 드는 연산은 각 노드에서 레코드들의 클래스들을 엔코딩(encoding)하는 비용이며, 각 파티션의 개별 클래스들에 속하는 레코드들의 개수에 대한 정보가 필요하다. 분류 후 기술되는 것은 새로운 데이터에 대해 유도된 마이닝 모델을 적용하는 예측(prediction)이며, 이런 적용을 prediction join 연산이라 한다. 새로운 소스 데이터(source data)의 속성 값들은 유도된 마이닝 모델로부터 제시된 사례(cases)들과 매칭(matching)이 된다. 잎 노드에 대한 클래스 라벨 배정은 훈련 데이터로부터 얻어진 통계적 기대값에 기초한다. 그러므로 주어진 사례들에 대한 예측된 클래스는 훈련 데이터로부터 유도된 확률과 같은 추가적인 통계치에 의해서 표현되어진다. 대부분의 경우 예측은 단일 값이 아니고 클래스들과 확률을 포함한다.

Prediction join의 구현은 모델 표현에 크게 의존한다. 예를 들면, 판단 트리는 연관된 조건-클래스와 노드-간선으로 표현되거나 예측된 클래스 라벨과 더불어 노드에 있는 분류점 또는 속성 값들의 개별 조합을 구체화함으로써 표현될

수 있다. 특정한 모델 표현이 주어진다면 prediction join 연산자는 중요한 분류 기초 기능이 될 수 있다.

판단트리 분류를 위한 기초 기능에는 다음과 같은 것들이 있다.

(1) 데이터 마이닝의 전처리 단계의 데이터 준비를 하는 기초 기능[10] : 만약 관련이 없거나 중복되거나 또는 잡음성의 데이터이거나 신뢰할 수 없는 데이터가 있다면 데이터 마이닝은 어렵게 된다. 데이터 전처리에는 데이터 필터링, 데이터 정규화, 변환, 추출과 선택 등이 포함된다.

(2) AVC group and CC table generation primitives(노드 통계표 작성 기초 기능) : 분류기에 공통되는 CC table 등을 작성한다. 판단 트리 알고리즘에서 각 노드는 그 노드의 데이터에 관련된 개수 통계를 얻어서, 모든 가능한 분할들을 평가해서 가장 좋은 분할을 선택한다. 각 속성에 대해서 그 속성 값과 클래스 값의 각 조합에 대해서 발생하는 튜플들의 개수가 필요하다. 분할 판단 기준은 개수 통계 테이블로부터 계산할 수 있으며, 4 개의 칼럼(속성 이름, 속성 값, 클래스, 개수)을 갖는 테이블이다. 이 테이블을 CC 테이블[8]이라 하고, 한 번 만들어지면 더 이상 데이터 자체를 참조할 필요가 없다.

(3) partial-match query의 구현에 의한 노드의 partition을 얻기 위한 filtering operation primitives.

(4) gini index, information entropy 같은 분류 척도를 계산하는 기초 기능 : 노드 통계 테이블에 기초해서 구현될 수 있다.

(5) prediction join operator 기초 기능.

3. 판단트리 기초기능의 PL/SQL 구현

본 절에서는 판단 트리 분류 기초 기능에 대해서 서술하고, partial-match 질의어를 지원하는 필터링 기초 기능에 대해서 서술한다. 판단 트리 T에서 노드를 $N_i(0 \le i \le n)$라면, N_0는 근 노드이다. 각 노드 N_i에 대해서 $A_i \ \theta \ v$ 또는 $A_i \in V$ 이고, 프레디키트 P_{Ni}에 연관되는 분류 조건과 각 노드에 따른 클래스 라벨 C_{Ni}를 배정한다. 클래스 라벨은 해당 노드의 파티션에 있는 가장 빈번한 클래스를 선택함으로써 결정되며, 시퀀스 $N_0\ N_1 \cdots\ N_k (k \le n)$을 decision path라 지정한다. 판단 트리 T에서, $i \ge 1$에 대해서 N_i는 N_{i-1}의 직접적인 자식 노드이다. 노드 N_r에 대한 decision path는 and 연결 조건으로 선택되어지며, $cond_{Nr}=P_{N1}$ and P_{N2} and$\cdots$ and P_{Nr}을 의미한다. 각 속성은 프레디키트에서 최대 한 번만 나타난다. 따라서 필터 기초 기능의 목적을 다음과 같이 기술한다.

- select 연산자 : 필터기초 기능
- 입력 : 데이터 세트 S, 노드 Nr에 대한 decision path의 분류 조건 $cond_{Nr}$
- 출력 : 파티션 $Sr \subseteq S$, $Sr = \sigma\ cond_{Nr}(S)$

기본적으로 필터 기초 기능을 구현하는 여러 가지 접근 방법으로 KDB 트리와 같은 다차원 해싱(MDH, Multi Dimensional Hashing), 그리드 파일, 비트맵 인덱스 등이 있으며, 그 중 본 논문에서는 기본 인덱스 또는 비트맵 인덱스로 구현한다. Table 2의 예시 테이블에서 속성 mrg(결혼 상태) 평가에 따른 hh(주택 소유 여부), inclev(연수입)에 대한 노드 통계를 산출하기 위한 판단트리(Fig. 2)와 N_2에 대한 파티션(Table 3) 및 노드 통계(Table 4)를 살펴보면 다음과 같다.

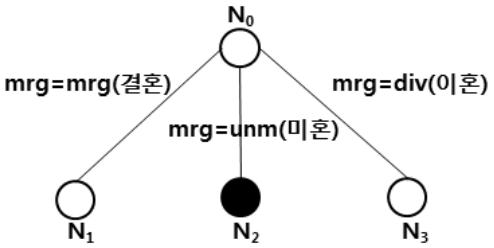

Fig. 2. Decision Tree for Node statistics

Table 3. Partition for Node N2

hh	inclev	defaults(class)
y	m	n
n	l	n
n	l	y
n	m	y

Table 4. Node statistics for Node N2

attribute name	attribute value	class label	count
hh	y	n	1
hh	n	n	1
hh	n	y	2
inclev	m	n	1
inclev	l	n	1
inclev	l	y	1
inclev	m	y	1

판단 트리의 노드와 연관된 파티션을 필터링하는 것은 분류 속성과 분류 조건을 결정하는 첫 단계이다. 속성 값과 클래스 라벨의 각 조합에 대한 튜플들의 개수를 요약하는 테이블로부터 information entropy와 gini index를 계산할 수 있다. 분류 속성에 의해 이미 사용하지 않은 데이터 세트의 모든 속성들이 조사된다. Fig. 2와 같이 부분적으로 성장한 트리와 노드 N_2에서 스키마 $R(A_1, A_2, \cdots, A_n,$ 클래스)에 대해서 속성 $A_2, \cdots, A_n$ 고려된다. Table 3과 같이 노드 N_2에서 데이터 세트의 나머지 파티션이 주어졌다고 한다면, 결과 테이블은 Table 4와 같은 정보를 갖는다. $Sp \subseteq S$에서 S_p는 데이터 세트 S의 한 파티션이라 하고, 클래스 Sp를 S_p에 나타나는 클래스 라벨의 집합이라 한다. $A=\{A_1, A_2, \cdots, A_m\}$을 속성들의 집합, $V=U_i dom(A_i)$를 값들의 집합이라 하고, 하나

의 레코드 r ∈ S_p에 대해서 r(A_i)를 레코드 r의 속성 A_i의 값이라 한다면, 노드 통계 계산 기초 기능은 다음과 같다.

- 입력 : 파티션 S_p
  속성들의 집합 A = {A_1, A_2, …, A_m}
  클래스 라벨을 나타내는 속성 '클래스Sp'
- 출력 : 릴레이션 S통계 ⊆ A x V x 클래스Sp x N(횟수)

(속성이름, 속성 값, 클래스 라벨, 횟수) ∈ S통계는 다음 사항과 equivalent(동등) 하다.

⇔ 속성 이름 ∈ A and 속성 값 ∈ V and 클래스 ∈ 클래스Sp
and 횟수 = |{ r | r ∈ S_p and r(A_i)=속성 값 and r(클래스 라벨) = 클래스}|

앞의 테이블을 만드는 방법으로 SQL 질의어를 사용하며, 예시로 근노드에서 노드 통계 테이블을 만드는 SQL 질의어는 다음과 같은 것이다.

```
insert into ndsttab  /* ndsttab : 근노드 노드 통계테이블, trtab : 훈련 레코드 테이블 */
  (select 'hh', hh, defaults, count(*) from trtab
  group by hh, defaults)  /* hh : 주택소유 여부 속성 */
union
  (select 'mrg', mrg, defaults, count(*) from trtab
  group by mrg, defaults)  /* mrg : 결혼 상태 속성 */
union
  (select 'inclev' inclev, defaults, count(*) from trtab
  group by inclev, defaults);  /* inclev : 소득 수준, default : 채무불이행 클래스 라벨 */
```

다음은 해당 노드의 통계를 계산하는 알고리즘이다.

```
procedure nodestatscomp(query q, attribute set A, class label attribute cl)
  array count = initialize
  execute query q
  foreach tuple t = (a1, a2, …, am, cl)
      foreach attribute A1, …, An∈ A
          count[Ai][ai][cl] += 1
  foreach attribute Ai
      foreach value v ∈ dom(Ai)
          foreach class label c ∈ dom(cl)
              if count[Ai][v][cl]>0
                  produce tuple (Ai, v, clb, count[Ai][v][cl])
```

다음은 분류 속성을 결정하기 위한 측도로 활용되는 기대정보 information entropy 계산식을 기술한다. 데이터 세트 S가 클래스 P에 속하는 p개의 개체와 클래스 N에 속하는 n개의 개체를 갖는다면, 데이터 세트 S에 대한 information entropy(I)는 다음과 같다.

- I(p, n) = -((p/(p+n)) * log(2, (p/(p+n)))) - (n/(p+n)) * log(2, (n/(p+n))))

A = a_i의 조건에 의해서 근노드의 데이터세트 S를 S_1, S_2, …, S_n분할 세트로 나누고, 각 분할 세트 S_i에서 속성 A는 속성 값 a_i를 갖는다. 분할 세트 S_i가 클래스 P에 속하는 p_i와 클래스 N에 속하는 n_i를 갖는다면, 분할 집합 S_i에 대한 부트리에 필요한 정보는 I(p_i, n_i)가 된다. 근노드에서 분류 속성으로 A를 택했을 때 기대 정보는 다음과 같은 균형 평균 기대정보 E(A) = ∑((p_i+n_i)/(p+n)) * I(p_i, n_i)이며, 근노드에서 속성 A를 분류 속성으로 분할했을 때 얻어지는 획득 정보 gain(A) = I(p, n) - E(A)와 같다. 이와 같은 계산 결과 가장 큰 획득 정보를 얻을 수 있는 속성으로 분류하는 것이 좋은 기준이 될 수 있다. ID3[6]는 모든 후보 속성에 대해서 gain(속성)을 계산을 해서 최대인 것을 분류 속성으로 정한다. 이런 과정을 각 부트리의 근노드에서도 수행을 해서 판단트리를 만든다. 판단 트리가 개체를 분류하기 위해서 사용될 때 예상되는 클래스의 반환은 근노드에서 시작하여 'y' 또는 'n' 결과 클래스 개수 3개와 7개를 반환하는 것으로 생각할 수 있다. Table 1의 결과 클래스의 기대 정보는 다음과 같다.

- I(y, n) = -(3/(3+7)) * log(2,3/(3+7)) - (7/(3+7)) * log(2,7/(3+7)) = 0.88

어떤 속성 A에 의해서 근노드에서 분류했을 때, 각 부트리에서 'y' 또는 'n' 메시지를 나타내는데 필요한 기대 정보는 I(y_i, n_i)이고, 분류했을 때 근노드에서의 기대 정보 E(A)는 가중치 평균으로 각 속성에 적용하여 계산하면 다음과 같다.

- E(mrg) = 0.4 * (-(0/(0+4) * log(2,0)) - 1 * log(2,1)) + 0.4 * (-0.5 * log(2,0.5) - 0.5 * log(2,0.5)) + 0.2 * (-0.5 * log(2,0.5) - 0.5 * log(2,0.5)) = 0.6
- E(hh) = 0.3 * (-1 * log(2,1)) + 0.7 * (-(3/7) * log(2,(3/7)) - (4/7) * log(2,(4/7))) = 0.69
- E(inclev) = 0.5 * (-(2/5) * log(2,(2/5)) - (3/5) *log(2,(3/5))) + 0.4(-(1/4) * log(2,(1/4)) - (3/4) * log(2,(3/4))) + 0.1 * (-1 * log(2,1)) = 0.8

따라서 속성 A에 의해서 분류했을 때 얻어지는 획득 정보 gain(A)을 각 속성에 적용하여 계산하면 다음과 같다.

- gain(mrg)=I(y,n) - E(mrg)=0.88 - 0.6 = 0.28
- gain(hh) = I(y,n) - E(hh) = 0.88 - 0.69 = 0.19
- gain(inclev) = I(y,n) - E(inclev) = 0.88 - 0.8= 0.08

위 결과, 근노드에서 각 속성에 대한 정보 획득을 계산했을 때 mrg 속성이 가장 정보 획득이 컸다. 따라서 근노드에서 분류 속성으로 mrg를 선택하고 분류했을 때, 세 개의 부트리가 생성된다. 'mrg=mrg'인 프레디키트에 의해서 분류된 부트리는 클래스 라벨이 전부 'n'이기 때문에 더 이상 분류가 불필요하다. 'mrg=unm'인 프레디키트에 의해서 분류된 부트리에서의 각종 기대 정보는 다음과 같다.

- I(y,n)= -0.5*log(2,0.5)-0.5*log(2,0.5)=1.0
- E(hh)= (1/4)*(-log(2,1))+(3/4)*(-(2/3)*log(2,(2/3))-(1/3)*log(2,(1/3)))=0.69
- E(inclev)=0.5*(-0.5*log(2,0.5)-0.5*log(2,0.5))+0.5*(-0.5*log(2,0.5)-0.5*log(2,0.5))=1

미혼(unm) 노드에서 각 속성별 정보 획득을 계산하면 아래와 같은 결과를 얻을 수 있다.

- gain(hh)=0.31, gain(inclev)=0.0

이 결과, 미혼 노드에서는 hh 속성의 값이 크기 때문에 선택하고 분류한다. 다음은 이혼(div) 노드에서 기대 정보를 계산한다.

- I(y,n) = -0.5*log(2,0.5)-0.5*log(2,0.5)=1
- E(hh) = 0.5*(-log(2,1))+0.5*(-log(2,1))=1
- E(inclev) = 0.5*(-log(2,1))+0.5*(-log(2,1))=1

이혼 노드에서 속성 hh와 inclev에 의한 정보 획득은 동일하며(hh를 분류 속성으로 임의로 정하였음), 이렇게 만들어진 판단트리는 Fig. 3과 같이 나타낼 수 있다. 판단 트리가 만들어진 후 유도된 모델은 클래스 속성이 없는 새로운 데이터 레코드의 클래스를 예측할 수 있고 이러한 연산을 prediction join이라 한다. 이 연산은 각 노드의 연관된 분류 조건이 평가되는 트리의 경로를 따라서 모델이 해석되어지며 다음과 같이 정의할 수 있다.

(prediction-join)

- 입력 : 판단 트리 T : 노드 N_0, … N_n으로 구성
  소스 릴레이션 $R(A_1, \cdots, A_m)$

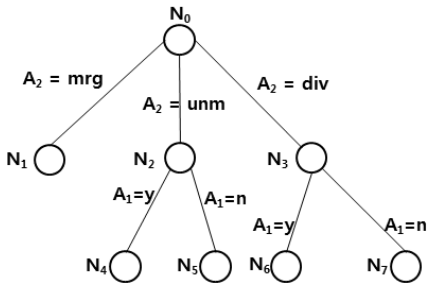

Fig. 3. Decision Tree Model

- 출력 : prediction 릴레이션 $Rp(A_1, \cdots, A_m, 클래스속성)$
  path $N_0, \cdots, N_k$, $\forall_t \in R$, $\exists t_p \in Rp : k \leq m \wedge k$ maximal
  $\wedge \forall_i\ i = 1, \cdots, k : t(A_i)=t_p(A_i)$
  $\wedge cond_{Np}(t_p)$=true and $t_p(cl)=cl_{Ni}$

prediction join의 구현은 모델 표현에 의존하기 때문에 적당한 자료 구조가 필요하며, 트리를 RDBMS에 저장하기 위해서 flat table 구조를 이용한다. 그런 예로 Fig. 3의 판단 트리를 테이블로 표현하면 Table 5와 같다.

Table 5. Decision Tree Table of Fig. 3

node	parents node	attribute	attribute value	class label	probability
N_0	-	-	-	n	0.70
N_1	N_0	A_2	mrg	n	1.0
N_2	N_0	A_2	unm	y	0.5
N_3	N_0	A_2	div	y	0.5
N_4	N_2	A_1	y	n	1.0
N_5	N_2	A_1	n	y	0.67
N_6	N_3	A_1	y	n	1.0
N_7	N_3	A_1	n	y	1.0

판단 트리 테이블의 각 튜플은 각 노드에 해당되며, N_1노드에 해당하는 튜플은 다음과 같은 pl/sql 코드에 의해서 생성된다.

```
declare
v_ncnt number;
v_ycnt number;
clslb varchar2(5);
v_prob number;
begin
  select count(*) into v_ncnt        /* psmset : 훈련 데
  이터 테이블 */
  from (select * from psmset where mrg='mrg') g
  where defaults='n';
 v_ycnt :=0;
 if v_ncnt >= v_ycnt then
   clslb:='n';
   select v_ncnt / (v_ncnt + v_ycnt) into v_prob
   from dual;
 else
   clslb:='y';
   select v_ycnt / (v_ncnt + v_ycnt) into v_prob
   from dual;
 end if;                              /* ndttab : 판단 트리
 테이블 */
   insert into ndttab(node, parent, att, atval, cls, prob)
   values(1, 0, 'mrg', 'mrg', clslb,v_prob);
end;
```

판단 트리의 테이블 표현의 각 튜플은 부모 노드로부터 현재 노드까지를 경로로 하는 트리의 각 간선(edge)으로 나타낸다. 간선은 조건과 연관되어 있으며, 속성 이름은 속성 필드에 저장되고, 분류를 위한 도메인은 속성값 필드의 값들에 의해서 표현된다. 클래스 라벨 필드는 그 클래스에 속할 확률과 더불어 간선의 노드에 연관된 파티션에 가장 많이 나타나는 클래스의 라벨을 갖는다.

prediction join의 의사 코드는 다음과 같다.

```
procedure predictionjoin(source table S, model table M)
  foreach tuple tS=(a1, …, am)∈ S
     execute query q(tS)
     fetch tuple tM=(n, p, c, prob)
     node := n
     classlabel := c
     finished := false
     do
       do
          fetch tuple tM=(n, p, c, prob)
          if tuple = not found
            create result tuple (a1, …, am, c)
            finished := true
       while  p ≠ node
       node := n
       classlabel = c
     while not finished
```

소스 릴레이션의 각 튜플 $t_S=(a_1,\cdots, a_m)$에 대해서 튜플의 속성 값에 의해서 만족되는 조건을 갖는 노드들이 선택된다. 다음은 Table 2의 시험 데이터 집합의 11번 튜플에 대해서 판단 트리 테이블로 대응하여, 그 결과를 임시 판단 트리에 넣은 후 임시 판단 트리 테이블을 노드 번호 순으로 정렬하는 pl/sql 코드이다.

```
set serveroutput on
begin
 delete from tdttab; /* tdttab은 임시 판단 트리 테이블 */
 insert into tdttab
    select * from ndttab /* ndttab은 훈련 데이터에 대
    한 판단 트리 테이블 */
    where (att='hh' and atval='n') or
          (att='mrg' and atval='unm') or
          (att='inclev' and atval='l');
 delete from stdttab;  /* stdttab은 임시 판단 트리 테이
 블을 노드 번호순으로 정렬 */
 insert into stdttab
    select * from tdttab
    order by node asc;
 end;
```

위의 질의어의 원형은 다음과 같다.

```
select *
from model
where  (atname = 'A1'  and  A1=a1)or
       (atname = 'A2'  and A2=a2)or
       …
       …
       (atname = 'Am' and Am=am)
```

이 후보 노드들은 노드-id(노드 번호) 순으로 정렬되며 근 노드로부터 시작해서 현재 노드-id가 부모-id인 다음 노드를 얻는다. 이런 경우 클래스 라벨과 확률이 활동 소스 튜플(시험 데이터 튜플)에 배정된다. 이런 절차를 수행하는 시험 데이터 세트의 11번 튜플을 처리하는 pl/sql 코드는 다음과 같다.

```
 declare
  frnode number; /* current node */
  nxnode number; /* next node */
  tcls varchar2(5); /* class label */
  tprob number; /* probability */
 procedure calcsetcls(strowid in varchar2) /* calculate
and set the class label and probability */
 is
 begin
  select node into frnode from stdttab /* sorted
  temparay decision table */
  where parent = 0;
  loop
  select node into nxnode from stdttab
  where parent=frnode;
 if nxnode != 0 then
   frnode := nxnode;
  end if;
  end loop;
 exception
  when NO_DATA_FOUND then
   select cls into tcls from stdttab
   where node = frnode;
   select prob into tprob from stdttab
   where node = frnode;
  update pstset /* update the test set tuple */
    set defaults=tcls
    where id=strowid;
  update pstset
    set prob=tprob
    where id=strowid;
 end calcsetcls;
```

```
begin
calcsetcls('11'); /* '11' tuple of the test set table */
end;
```

Table 2의 시험 집합을 판단 트리의 테이블에 적용한 결과는 Table 6과 같다.

Table 6. Decision Tree Table of Table 2. Test set

id	hh	mrg	inclev	defaults	prob
11	n	unm	l	y	0.67
12	y	mrg	l	n	1.0
13	y	div	m	n	1.0
14	n	unm	m	y	0.67
15	n	div	l	y	1.0

4. 성능 평가

본 논문의 기초 기능에 대한 성능을 비교 평가하기 위해서 sun workstation 시스템에 solaris OS를 구축하고, 대부분 데이터베이스 환경에서 실험이 가능하나 본 논문에서는 이들 제품들 중에서 Oracle 9i 버전의 pl/sql 프로그래밍 언어를 사용하여 실험을 진행 하였다. 합성된 실험 데이터 세트는 10개의 속성을 갖는 테이블로 구성하고 적용된 튜플의 개수는 10만 개, 11만 개, 12만 개 등 세 가지로 분류하였다. 각 속성은 1에서 5까지의 개별 값들을 갖는다. 본 실험으로 첫 번째, 노드 통계 테이블을 만드는 질의어에 대한 몇 가지 인덱스 전략에 대한 성능 실험이며, 두 번째, partial-match 질의에 대한 전체 테이블 스캔, 비트맵 인덱스, MDH 기반 기본기능 등을 비교 평가하였다.

다음은 성능평가 실험에 이용하기 위한 훈련 집합, 노드 통계 테이블, 기본 인덱스, 비트맵 인덱스 등을 만드는 pl/sql 코드들이다.

▪ 훈련 집합 생성 pl/sql 코드

```
begin
 delete from trset;
 sttime := systimestamp;
 for i in 1 .. 120000 loop
  insert into trset(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,cs) /*
  this is the training set */
  select trunc(dbms_random.value(1,6)),
   ~ 중략 ~
        trunc(dbms_random.value(0,2))
  from dual;
  end loop;
end;
```

▪ 노드 통계 테이블 생성 pl/sql 코드

```
delete from ndsttab; /* ndsttab: node statistics table */
set serveroutput on
declare
 sttime    timestamp;
 entime    timestamp;
 tstring varchar2(200);
begin
 sttime := systimestamp;
insert into ndsttab
(select 'a1', a1, cs, count(*) from trset group by a1, cs)
union
(select 'a2', a2, cs, count(*) from trset group by a2, cs)
   ~ 중략 ~
union
(select 'a10', a10, cs, count(*) from trset group by
 a10, cs);
  entime := systimestamp;
  dbms_output.put_line(entime - sttime);
  delete from times;
  insert into times values(entime, sttime);
  select extract(hour from (tentime - tsttime)) || ':' ||
   extract(minute from (tentime - tsttime)) ||
   ':' || extract(second from (tentime - tsttime))
  into tstring from times;
  dbms_output.put_line(tstring);
end;
```

▪ 기본 인덱스 생성 pl/sql 코드

```
create table itrset12(a1 varchar2(2), a2 varchar2(2), a3
varchar2(2),
a4 varchar2(2), a5 varchar2(2), a6 varchar2(2), a7
varchar2(2),
a8 varchar2(2), a9 varchar2(2), a10 varchar2(2), cs
varchar2(2));
create index idx0112 on itrset12(a1);
create index idx0212 on itrset12(a2);
     ~ 중략 ~
create index idx1012 on itrset12(a10);
```

▪ 비트맵 인덱스 생성 pl/sql 코드

```
create  bitmap index idxb0112 on btrset12(a1);
create  bitmap index idxb0212 on btrset12(a2);
     ~ 중략 ~
create  bitmap index idxb0912 on btrset12(a9);
create  bitmap index idxb1012 on btrset12(a10);
```

첫 번째, 비교 평가로 노드 통계 테이블을 만드는 질의어에 대한 각 인덱스 전략별 실험 결과는 아래 Table 7과 같다.

Table 7의 결과를 기반으로 성능 비교한 결과, 인덱스를 사용하지 않은 질의와 기본 인덱스를 사용한 질의 결과보다 비트맵 인덱스를 사용한 질의 결과가 더 우수하다는 것을

Table 7. Specific performance test results for each index strategy (unit : second)

tuple number	count	no index	primary index	bitmap index
100,000	1	4.574092	4.155113	4.004418
	2	4.296085	4.200096	4.020136
	3	4.310769	4.199035	4.003782
110,000	1	4.747681	4.531726	4.438142
	2	4.752661	4.536967	4.457907
	3	4.746554	4.536083	4.427109
120,000	1	5.046417	4.892384	4.855071
	2	5.008943	4.889755	4.850213
	3	5.051371	4.901146	4.857553

알 수가 . 추가적으로 기본 인덱스를 사용한 테이블 생성 시간 보다 비트맵 인덱스를 사용한 테이블 생성 시간이 훨씬 작았다. 이 결과 테이블 및 질의어 수행 시간 면에서 기본 인덱스를 사용하는 것 보다 비트맵 인덱스를 사용하는 것이 우수함을 학인 할 수 있었다.

두 번째 실험으로 partial-match 질의어에 대한 서로 다른 전략으로 전체 테이블 스캔, 비트맵 인덱스, MDH 기반 기본기능 등올 비교를 하였으며 상기에서 제시된 다양한 수의 정의되지 않은 속성들을 가진 10만개 튜플의 테이블을 대상으로 하였다. 아래 Fig. 4는 전략별 100개의 질의를 적용한 평가 결과 시간이다.

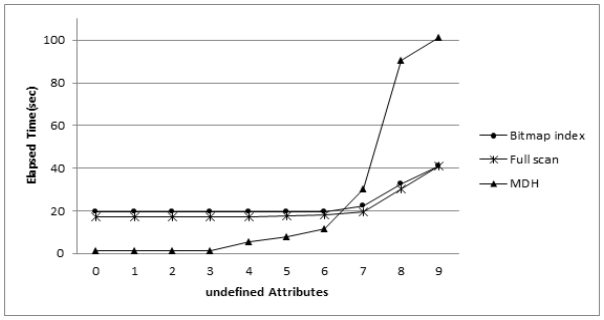

Fig. 4. partial-match query evaluation result

partial-match에서 모든 속성들이 쿼리에 주어진 경우라면 완전 검색 쿼리의 상황에서 MDH가 1.2초 필요하며, 완전 스캔은 약 17초, 비트맵의 경우 19.5초의 시간을 필요로 한다. 정의되지 않는 속성들의 수가 증가할수록 경과 시간은 완전 스캔과 비트맵 인덱스보다 많이 시간을 필요로 하지만 완전 검색 쿼리에서 6개의 속성 개수까지는 MDH 전략이 우수하다는 점을 보였다.

5. 결 론

데이터 마이닝과 데이터베이스 시스템의 결합은 대형 데이터베이스에 대한 데이터 마이닝의 효율을 향상시킨다. 특히 판단 트리 분류에 대한 기초 기능이 DBMS에 의해서 제공되는 경우에 대해서 본 논문에서 서술했고, 기초 기능을 pl/sql로 구현하는 것에 대해서 기술하였다. 첫째로 판단 트리 분류 기능을 분석해서 공통적인 기초 기능을 확인하였으며, 둘째로 기초 기능의 효율적인 구현을 위해서 데이터베이스 시스템의 pl/sql을 사용하였다. 기초 기능의 하나로 분류 측도를 계산하는 연산을 노드 통계 테이블에 대한 pl/sql의 함수로 구현했다. 예시 훈련 데이터에 대한 판단 트리 모델을 만들고 예시 시험 데이터에 적용해서 예상되는 클래스와 확률을 계산하였다. 또한, 성능 평가하기 위하여 첫 번째로 판단 트리에 대한 노드 통계 테이블을 구현하는 질의어를 구현해서, 전체 테이블 탐색, 기본 인덱스를 이용한 탐색, 비트맵 인덱스를 이용한 탐색별로 질의어를 실행해서 평가하였으며, 두 번째로 partial-match 질의어에 대한 서로 다른 전략으로 전체 테이블 스캔, 비트맵 인덱스, MDH 기반 기본기능 등을 비교하였다. 실험 결과는 이런 기초 기능의 장점을 각 전략별로 제시하였고, 데이터베이스 시스템과의 통합의 필요성을 보여주었다.

참 고 문 헌

[1] Surajit Chaudhuri, “Data Mining and Database Systems: Where is the Intersection?,” Data Engineering Bulletin, 21(1): 4-8, 1998.

[2] R. Meo, G. Psaila, and S. Ceri. A New SQL-like Operators for Mining Association Rules. VLDB’96, pp. 122-133, Mumbai, India, Sept., 3-6, 1996. R.

[3] A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad, “Integration of Data Mining with Database Technology,” Proceedings of 26the International Conference on Very Large Data Bases, September 10-14, 2000.

[4] Vipin Kumar, etc., Introduction to data mining, Addison-Wesley, May 12, 2005.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, “Classification and Regression Trees,” Chapman and Hall, 1984.

[6] M. Xu, J. Wang, and T. Chen, “Improved decision tree algorithm: ID3+,” Intelligent Computing in Signal Processing and Pattern Recognition, Vol.345, pp.141-149, 2006.

[7] M. Mehta, I. Rissanen, and R. Agrawal, “MDL-based Decision Tree Pruning,” Proc. of Intl. Conf. on Knowledge Discovery in Databases and Data Mining, Montreal, Canada, 1995.

[8] S. Chaudhuri, U. M. Fayyad, and J. Bernhardt, “Scalable Classification over SQL Databases,” ICDE-99, pp.470-479, Sydney, Australia, 1999.

[9] J. Gerhke, R. Ramakrishnan, and V. Ganti, “RainForest - A Framework for Fast Decision Tree Construction of Large Datasets,” VLDB’98, pp.416-427, New York City, New York, USA, 1999.

[10] S.B. Kotsiantis, D. Kanellopoulos and P.E. Pintelas, “Data Preprocessing for supervised learning,” International Journal of Computer Science, Vol.1, No.2, 2006.
[11] M. BenHajHmida and A. Congiusta, “Parallel, distributed, and grid-based data mining : algorithms, systems, and applications,” Handbook of Research on Computational Grid, IGI Global, pp.90-119, May, 2009.
[12] L. Zhou, Z. Zhang, and M. Xu, “Massive data mining based on item sequence set grid space,” In Proceedings of the 2nd International Asia Conference on Informatics in Control, Automation and Robotics, pp.208-211, March, 2010.

안 형 근
e-mail : hkahn@ulsan.ac.kr
2003년 울산대학교 정보통신대학원 정보통신공학과(공학석사)
2008년 울산대학교 컴퓨터정보통신공학부(공학박사)
1997년～2004년 현대오토시스템 기술지원부
2004년～2006년 (주)CFIC 기업부설연구소 연구소장
2008년～2010년 울산대학교 컴퓨터정보통신공학부 객원교수
2012년～현 재 울산대학교 LINC사업단 연구교수
관심분야 : 멀티미디어DB, DB설계/분석, ERP, BPM, Workflow

고 재 진
e-mail : jjkoh@ulsan.ac.kr
1972년 서울대학교 응용수학과(공학사)
1981년 서울대학교 계산통계학과(이학석사)
1990년 서울대학교 컴퓨터공학과(공학박사)
1975년～1979년 한국후지쯔(주) 기술개발부 사원
1979년～2010년 울산대학교 컴퓨터정보통신공학부 교수
2011년～현 재 울산대학교 전기공학부 교수
관심분야 : DB시스템, 전문가 시스템, DB설계, ERP