비트맵을 활용한 분류 구현

조용준*, 이상원*
*성균관대학교 컴퓨터공학과
e-mail : antelope@skku.edu , swlee@skku.edu

BBC : Bit-map Based Classification

Yong-Joon Cho*, Sang-Won Lee*
*Dept. of Computer Engineering, SungKyunKwan University

요 약

분류란 여러 분야에서 쌓인 정보 데이터를 분석하여, 결과값에 대한 공통속성을 찾아내어 새로운 입력 데이터에 대해 보다 보편적인 결과를 분석하거나 예측하는 기법이다. 의사 결정 트리는 이러한 분류의 한 형태로 저장된 데이터를 활용하여 선험적 지식을 취득하고, 새로운 데이터에 대한 예측을 발생시키는 데이터 분석 방법이다. 그러나, 의사 결정 트리의 여러 가지 장점에도 불구하고 트리 구성에 많은 비용이 소요되는 단점이 존재한다. 점점 대량의 데이터를 다루어야 하는 현대 사회에서는 이러한 단점이 더욱더 커질 수 밖에 없다. 본 논문에서는 이러한 문제점을 해결하고자 비트맵을 활용한 의사 결정 트리의 구현을 제안한다. 비트맵을 사용하게 되면 의사 결정 트리 생성의 가장 큰 비용인 속성값 측정에서 높은 효율을 유지할 수 있게 된다. 또한 보다 효율적이고, 확장성이 높은 의사 결정 트리를 구현할 수가 있다.

1. 서론

현대 사회에서는 각종 분야에서 컴퓨터 사용의 증가로 엄청난 량의 데이터가 수집되고 있습니다. 이렇게 쌓인 데이터들은 여러 가지 기법을 통하여 분석 및 재가공 되고 있습니다.

데이터 베이스에서의 분류란 이렇게 여러 분야에서 쌓인 정보 데이터를 분석하여, 결과값에 대한 공통 속성을 찾아낸 후, 새로운 입력 데이터에 대해 보다 보편적인 결과를 분석하거나 예측하는 데이터 마이닝 기법입니다.

각 집합에 속한 데이터들의 특성을 분석함으로써, 새로 제공되는 데이터의 속성을 유도할 수 있게 됩니다. 이러한 분류는 신청자의 여러 자격요건에 따라 발행되는 카드의 종류를 결정하거나, 은행 대출의 한도 및 가능 여부 등 여러 가지 객관적인 비지니스 의사 결정을 하는데 쓰이게 된다.

의사 결정 트리는 분류의 한가지 구현 형태로서 잎과 노드를 가진 트리 구조를 생성함으로써 새로운 데이터에 대한 검사 및 예측을 쉽게 진행하도록 한 것이다.

이러한 의사 결정 트리는 여러 가지 장점에도 불구하고, 구성 비용이 많이 드는 단점을 가지고 있습니다. 이에 이 논문은 비트맵을 활용하여 보다 빠르고, 보다 확장성이 높은 새로운 의사 결정 트리의 구현을 제안하고자 한다.

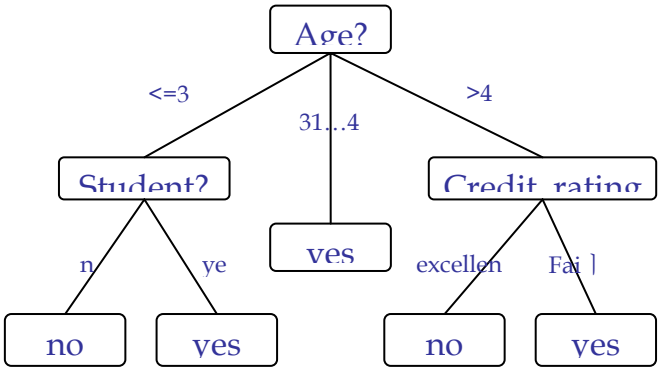
본 논문의 구성은 다음과 같다.

제 2 장에서는 의사 결정 트리의 기본 구현 알고리즘에 대해서 알아보겠다. 제 3 장에서는 제안하는 비트맵 알고리즘의 특징 및 구현 형태에 대해서 알아보겠다. 제 4 장에서는 실험을 통해 기본 구현 알고리즘과 제시된 새로운 알고리즘의 성능을 비교하겠다. 제 5 장에서는 결론 및 향후 연구 과제를 제시 하겠다.

2. 의사 결정 트리 알고리즘

2.1 정의

의사 결정 트리(그림 2)는 트리 형태로 구성되며, 중간 노드에 속성검사를 위한 분기 조건들을 제시하고, 이에 따라 입력 데이터를 검사하여 클래스 값을 도출하는 일종의 흐름도이다.



(그림 2) 의사 결정 트리

의사 결정 트리의 기본적인 알고리즘은 아래와 같다.

(1) Create a node N;
(2) If samples are all of the same class, C then
(3) Return N as a leaf node labeled with the class C;
(4) If attribute-list is empty then
(5) Return N as a leaf node labeled with the most common class in samples; //majority voting
(6) Select test-attribute, the attribute among attribute-list with the highest information gain;
(7) Label node N with test-attribute;
(8) For each known value ai of test-attribute // partition the samples
(9) Grow a branch from node N for the condition test-attribute = ai;
(10) Let si be the set of samples in samples for which test-attribute = ai; // a partition
(11) If si is empty then
(12) attach a leaf labeled with the most common class in samples;
(13) Else attach the node returned by generate_ decision_tree(si, attribute-list-test-attribute);

2.2 속성 선택 방법

각 노드에 필요한 속성의 선택은 information gain 값을 계산하여 얻을 수 있다.

가장 큰 information gain 값을 갖는 속성이 각 노드에서 분기 조건으로 작용하게 되는데, 무작위성 또는 불순도를 최소화 하도록 만들어 진다.

Information gain 을 얻는 식은 아래와 같다.

$$GAIN(A) = I(S_1, S_2, \ldots, Sm) - E(A)$$

$$I(S_1, S_2, \ldots, Sm) = - Pi \sum \log2(Pi)$$

$$E(A) = \frac{\sum S_{1j} + \ldots + Smj}{S} I(S_{1j}, \ldots, Smj)$$

$$Pij = \frac{S_{ij}}{|S_j|}$$

여기서 I(S1, S2, …, Sm)은 주어진 샘플에 대한 계산 값 이고, E(A)는 각 속성에 대한 계산 값, Pij 는 샘플이 클래스에 속하는 확률이 된다.

각 노드 마다 위와 같은 값을 계산하면서 GAIN(A) 값을 얻고, 각 속성들의 GAIN(A)값을 비교하면서 위의 알고리즘(6-7 단계)을 적용해 나가게 됩니다.

위의 연산은 대부분 클래스에 속한 값이 몇 개인지를 계산, 평가하는 수식이다. 따라서 속성에 속한 값의 카운팅은 전체 구성 속도에 많은 영향을 미치는 중요한 요소가 된다.

3. 비트맵을 활용한 알고리즘

3.1 정의

위에서 서술한 바와 같이 기본 알고리즘 연산은 대부분 클래스에 속한 값이 몇 개인지를 계산, 평가하는 수식입니다. 따라서 속성에 속한 값의 카운팅은 전체 구성 속도에 많은 영향을 미치는 중요한 요소가 된다.

제안하는 알고리즘은 이러한 부분의 속도 향상을 위해 제공되는 데이터를 비트맵으로 구성한다. 비트맵 재구성된 데이터는 각 속성들의 연산과 비트맵 카운팅에서 빠른 속도를 제공한다.

표 3 에서 보는 바와 같이 제공된 데이터를 비트맵으로 재구성한 후에 알고리즘에 필요한 연산을 하면 된다.

RID	1	2	3	4	5	6	7	8	…
Age1 (<=30)	1	1	0	0	1	0	0	0	…
Age2 (31…40)	0	0	1	0	0	1	1	0	…
Age3 (>40)	0	0	0	1	0	0	0	1	…
Income1 (low)	0	0	0	0	1	1	1	0	…
Income2 (medium)	0	0	0	1	0	0	0	1	…
Income3 (high)	1	1	1	0	0	0	0	0	…
…	…	…	…	…	…	…	…	…	…
class	0	0	1	1	1	0	1	0	…

<표 2> 비트맵 구성

표 2 와 같이 비트맵을 구성하면 의사 결정 트리를 생성하기 위한 알고리즘의 필요 값들을 쉽게 계산할 수 있게 된다. GAIN(A)값을 계산하기 위한 수식들은 대부분 S1, S2 등으로 구성 되는 데, 이것은 AND 연산과 COUNT 를 통해 쉽게 구할 수가 있게 된다.

예를 들면 S11 과 S21 을 구해야 하는 경우

S11 = COUNT(age1 AND class)
S21 = COUNT(age1 AND not(class))

과 같이 실행하면 된다.

위와 같은 연산을 행함으로써 여러 가지 속성들의 카운트 값을 보다 빠르게 얻을 수 있고, G(A), I(S1…Sm), E(A) 등의 값을 손쉽게 알아 낼 수가 있다.

3.2 구현

본 논문은 구현측면에서 여러 가지 성능 향상 기술을 적용시켰다. 또한 비트맵 연산에 SIMD 를 적용하였다.

3.2.1 AND ( ) 함수

비트맵 AND 연산은 두 메모리 공간의 데이터를 읽어와 AND 하고 그 결과를 다시 메모리 공간에 저장함을 의미한다. 이 때 비트맵 크기가 동적이라는 점과 메모리 공간에 다시 저장해야 한다는 두 가지 특징을 AND ( ) 함수를 사용하여 효과적으로 활용하여 구현 성능에 많은 영향을 가졌다.

3.2.2 카디널리티 계산

카디널리티는 비트맵 내의 참인 비트 개수, 즉 카운트 값을 의미한다. 카운트의 계산 알고리즘은 비트맵 연산의 핵심적인 부분이다. 비트의 개수를 계산하는 알고리즘은 여러 가지 이지만, 결국 8 비트 단위로 값에 따른 비트 수를 미리 계산하여 배열에 넣어두고, 참조하면서 계산하는 방식을 적용하여 성능 향상 효과를 얻었다.

3.2.3 SIMD 의 적용

SIMD(Single Instruction Multiple Data)는 한번의 명령어로 여러 개의 데이터를 처리하는 방식이다. 본 논문에서는 Intel Pentium4 – processor 에서 제공하는 SSE2(Streaming SIMD Extentions 2)를 이용하여 AND( ) 함수와 COUNT( )함수를 SIMD 화 하였고, 이를 통한 성능향상을 보았다.

4. 실험 결과 및 분석

본 장에서는 제안된 알고리즘을 기본적인 의사 결정 트리 알고리즘의 실행 시간과 비교하였다.

실험은 비트맵을 활용할 수 있는 부분을 위주로 하였다.

4.1 실험 환경 및 데이터

제안된 알고리즘의 성능을 평가하기 위한 실험 환경은 P-4 2.0GHz , 512MB, Linux Kernel 2.4 가 사용되었고, 구현은 C 언어로 되었다.

성능 평가에 사용된 데이터는 속성값이 랜덤하게 생성된 데이터이다. 생성된 데이터의 파라미터 인자는 표.4 와 같다.

D	데이터 행의 수
A	사용된 속성의 수
P	속성에 사용된 값의 수

<표 4> 파라미터 목록

4.2 실험 결과

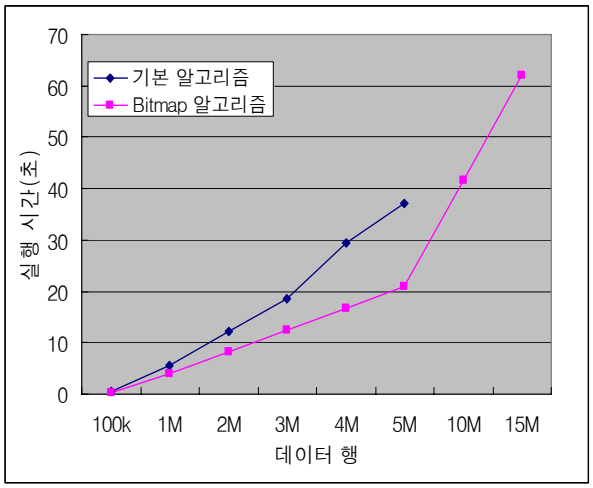
실험 1 은 데이터의 수를 증가시키면서 의사 결정 트리의 생성 속도를 기본 알고리즘과 비교 측정한다.

실험 2 는 속성에 사용된 값의 수를 증가시키면서 생성 속도를 비교 측정한다.

실험 3 은 속성의 수를 증가시키면서 생성 속도를 비교 측정한다.

실험 1 에서는 100k 에서 12M 까지 데이터 행을 증가하면서 실험을 수행한다. 실험 결과는 그림 5.1 에서 볼 수 있는 것처럼 데이터의 량이 늘어날수록 2 배 가까운 속도 향상을 볼 수가 있다. 또한 10M 이상의 데이터 행에 대해서는 기본 알고리즘이 시스템 상에서 처리하지 못함을 보여주고 있다.

기본 알고리즘은 문자열로 데이터를 처리하기 때문에 Bitmap 에 비하여 상대적으로 낮은 처리량을 보여준다. 또한 비교적 낮은 속성과 속성값이 이처럼 기능 차이을 나타내는 이유가 되었다. 이는 생성되는 노드의 수를 상대적으로 적게 발생시키기 때문인데, 이로 인한 bitmap 연산의 수가 상대적으로 적게 발생하게 된다.

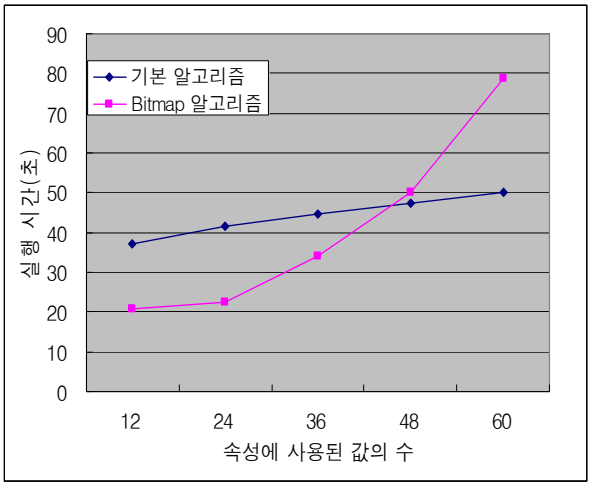


(그림 5.1) 실험 1. A5P13 의 데이터 행 변화에 따른 성능평가

실험 2 에서는 각 속성에 사용된 값의 수를 증가시키면서 평가한다.

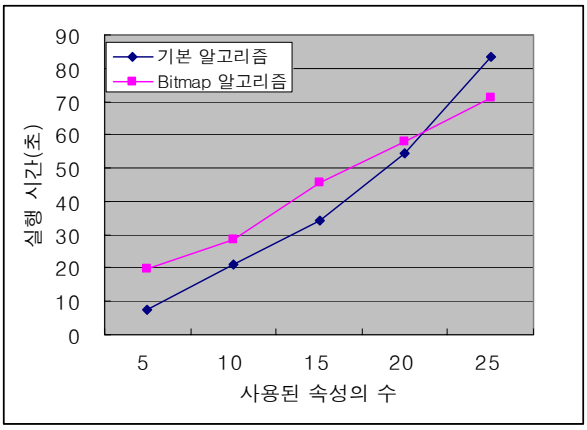
실험 결과는 그림 5.2 에서 보여지는 것처럼 속성에 사용된 값이 낮은 초기에는 Bitmap 알고리즘이 월등히 빠른 속도를 보이다가, 점차 많아짐에 따라 성능의 역전현상이 나타남을 볼 수 가 있다. 이러한 결과가 발생되는 이유는 속성의 값이 증가함에 따라 결과로 생성되는 노드의 수가 exponential 하게 증가하기 때문인데, 이에 따라 Bitmap 연산의 수가 생성된 노드의 수에 비례하여 증가하기 때문이다.

실험 3 에서는 전체 속성에 사용된 값의 수 변화 없이, 사용된 속성의 수만을 증가시키면서 평가한다. 실험 결과는 그림 5.3 에서 보여지는 것처럼 초기에는 기본 알고리즘이 좋은 성능을 보이지만, 사용된 속성의 수를 점점 증가시키고 속성에 사용된 값을 통합할수록 Bitmap 의 성능이 보다 안정적인 것을 알 수가 있다.



(그림 5.2) 실험 2. D2A9 의 속성에 사용된 값의 수 변화에 따른 성능평가

이러한 현상이 나타나는 이유는 전체 적으로 사용된 속성값의 수는 일정하게 유지되고 있기 때문이다. 전제 적인 속성값이 일정하게 유지되면, 생성되는 노드의 수 변화가 적어서 Bitmap 의 연산의 수도 변화도 적어지게 된다. 기본 알고리즘의 경우는 속성의 증가로 행 데이터의 크기가 커지기 되므로 Bitmap 에 비하여 더 많은 영향을 받게 된다.



(그림 5.2) 실험 2. D1P50 의 속성에 사용된 값의 수 변화에 따른 성능평가

5. 결론 및 향후 연구 과제

본 논문은 보다 효율적이고 빠른 의사 결정 트리를 생성하기 위하여 새로운 Bitmap 연산 알고리즘을 제안 하였다. 실험 결과에서 알 수 있듯이 단순한 형태의 데이터 집합에 대해 서는 2 배 가량의 빠른 성능 향상을 보였으며, 기본 알고리즘 보다 많은 데이터 집합에 대하여 처리가 가능하였다. 또한 데이터 집합의 확장에 대해서도 기본 알고리즘보다 낮은 실행 시간 증가율을 보여 주었다.

하지만, 사용된 속성의 수나 속성에 사용된 값이 커지는 경우는 생성되는 노드의 수가 많아져서 기본 알고리즘 보다 좋지 못한 결과를 보여주었다. 그리고 노드의 수는 속성이나 속성값의 증가에 대하여 exponential 하게 증가하는 경향이 있어 추가 개선이 시급하였다.

또한, 두 가지 알고리즘이 모두 메모리 내에서의 연산만을 수행하는데, 이 또한 추가 개선이 시급한 사항이라고 하겠다.

본 논문에서는 Bitmap 의 기본적인 적용을 통하여 기본 알고리즘의 성능향상을 제안한다. 일반적인 경우에는 대부분 기본 알고리즘에 비하여 좋은 성능과 확장성을 보였으나, 모든 경우에 좋은 성능을 나타내지는 못하였다.

실험을 통하여 몇 가지 추가 적인 개선 사항이 발견되었다. 추후 연구는 이러한 여러 문제점들의 개선을 위주로 진행되어갈 것이다.

연산된 결과를 저장함으로써 반복적인 연산을 줄이는 방식이나, 노드 생성을 제한하는 방법, 그리고 중복된 행 값들을 동시 저장하는 방법 등을 적용하여 더 빠르고, 확장성 높은 추가 연구를 진행 하고자 한다.

참고문헌

[1] M.Mehta, R.Agrawal, and J.Rissanen. SLIQ : A fast scalable classifier for data mining. In Proceedings of the 5th International Conference on Extending database Technology, Avignon, France March 1996.

[2] J.Ayres, J.E.Gehrke, T.Yiu, J.Flannick “Sequential pattern mining using bitmaps”, In Proc. 2002 ACM SIGKDD, jul.2002.

[3] J.C Shafer, R.Agrawal, and M.Mehta. SPRINT : A scalable paralle classifier for data mining. In Proceedings of the 1996 International Conference on Very Large Databases, Mumbai (Bombay), India, September 1996.

[4] R. Agrawal, S.Ghosh, T.Imielinski, B.Iyer, and A.Swami. An interval classifier for database mining application. In Proceedings of the 1992 International Conference on Very Large Databases, pages 560-573, Vancouver, Canada, August 1992.

[5] Takeshi Fukuda, Yasuhiko Morimoto, and Shinichi Morishita “Constructing Efficient Decision Trees by Using Optimized Numeric Association Rules”. In Proceedings of the 22nd VLDB Conference Mumbai(Bombay), India, 1996.

[6] J.R.Quinlan and R.L.Rivest, Inferring Decision Trees Using Minimum Description Length Principle. Information and Computation, 80:227-248,1989.