전역적 범주화를 이용한 대용량 데이터를 위한 순차적 결정 트리 분류기

한경식[0] 이수원
숭실대학교 컴퓨터학과 데이터마이닝 연구실
hanks@valentine.ssu.ac.kr[0], swlee@computing.ssu.ac.kr

Incremental Decision Tree Classifier Using Global Discretization For Large DataSet

Kyong-Sik Han[0] Soo-Won Lee
Dept. of Computing, Soongsil University

요 약

최근 들어, 대용량의 데이터를 처리할 수 있는 결정 트리 생성 방법에 많은 관심이 집중되고 있다. 그러나, 대용량 데이터를 위한 대부분의 알고리즘은 일괄처리 방식으로 데이터를 처리하기 때문에 새로운 예제가 추가되면 이 예제를 반영한 결정 트리를 생성하기 위해 처음부터 다시 재생성해야 한다. 이러한 재생성에 따른 비용문제에 보다 효율적인 접근 방법은 결정 트리를 순차적으로 생성하는 접근 방법이다. 대표적인 알고리즘으로 BOAT와 ITI를 들 수 있다. BOAT는 대용량 데이터를 지원하는 순차적 알고리즘이지만 분할 포인트가 노드에서 유지하는 신뢰구간을 넘어서는 경우와 분할 변수가 변경되면 그에 영향을 받는 부분은 다시 생성해야 한다는 문제점을 안고 있고, 이에 반해 ITI는 분할 포인트 변경과 분할 변수 변경을 효율적으로 처리하지만 대용량 데이터를 처리하지 못해 오늘날의 순차적인 트리 생성 기법으로 적합하지 못하다. 본 논문은 ITI의 기본적인 트리 재구조화 알고리즘을 기반으로 하여 대용량 데이터를 처리하지 못하는 ITI의 한계점을 극복하기 위해 전역적 범주화 기법을 이용한 접근방법을 제안한다.

1. 서론

최근 들어, 대용량의 데이터를 처리할 수 있는 트리 생성 방법에 많은 관심이 집중되고 있다. 그 이유는 기존의 메모리 기반 알고리즘은 오늘날의 대용량 데이터를 처리하지 못한다는 한계점과 더 많은 데이터를 이용하여 생성된 결정 트리가 더 좋은 정확도를 보이고 있기 때문이다[1]. 그러나, 대용량 데이터를 위한 대부분의 알고리즘은 일괄처리 방식(Batch Mode)으로 데이터를 처리하기 때문에 새로운 예제가 추가되면 이 예제를 반영한 결정 트리를 생성하기 위해 처음부터 다시 재생성해야 하다.

이러한 트리 재생성이 필요한 환경에 보다 효율적인 접근 방법은 결정 트리를 순차적으로 생성하는 접근 방법(Incremental Mode)이다[2][3]. 새로운 데이터가 추가될 때, 결정 트리를 처음부터 재생성 하는 것 보다 기존의 결정 트리를 유지하고 기존의 트리가 이 데이터를 반영할 수 있도록 갱신하는 것이 비용이 훨씬 적게 들기 때문이다. 결정 트리를 순차적으로 생성하는 대표적인 알고리즘으로 BOAT[2]와 ITI[3] 등을 들 수 있다. BOAT는 부트스트랩 기법(Bootstrapping)을 이용한 순차적 알고리즘으로서 대용량 데이터를 지원한다. BOAT는 수치형 분할 변수(Split Variable)를 위해 분할 포인트(Split Point)가 존재할 수 있는 구간을 유지하고 있기 때문에 그 범위 안에서 발생하는 분할 포인트 변경에 대해서는 효율적으로 처리하지만 분할 포인트가 그 구간을 넘어서는 경우와 분할 변수가 변경되면 그에 영향을 받는 부분은 다시 생성해야 한다는 문제점을 안고 있다. 이에 반해, ITI는 분할 포인트 변경과 분할 변수 변경을 효율적으로 처리하지만 대용량 데이터를 처리하지 못해 오늘날의 순차적인 트리 생성 기법으로 적합하지 못하다.

본 논문은 ITI의 기본적인 트리 재구조화(Tree Restructuring) 기법을 기반으로 하여 대용량 데이터를 처리하지 못하는 ITI의 한계점을 극복하기 위해 전역적 범주화 기법(Global Discretization Method)을 이용한 접근방법을 제안한다. 본 논문은 기존의 메모리 기반 전역적 범주화 알고리즘을 디스크 기반으로 확장하였고 새로운 데이터 추가로 발생하는 범주의 변화에 따른 문제를 트리 재구조화를 이용하여 처리하였으며 기존의 메모리 기반 ITI를 디스크 기반으로 확장하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 순차적 결정 트리 알고리즘들을 분석하고 3장에서는 대용량 데이터 처리를 위한 접근 방법으로서 전역적 범주화 기법을 이용한 순차적 트리 생성 방법에 대해 설명한다. 4장에서는 제안하는 접근방법에 대한 대용량 처리 가능성에 대해 실험하고 5장은 결론 및 향후 연구를 제시한다.

2. 관련 연구

2.1 BOAT

BOAT는 트리를 순차적으로 생성하기 위해 각 노드 n에 범주형 변수에 대한 클래스 분포와 모든 수치형 변수 X에 대한 버켓 경계(Bucket Boundary) x의 집합 및 각 버켓경계에 대한 클래스 C의 분포 n^i_x를 저장한다($n^i_x = |\{t \in F_n : t.X \le x \wedge t.C=i\}|$이며 i=dom(C), F_n은 노드 n에 소속되는 예제의 모임이며 t.X는 예제 t의 변수 X의 값이다). 특히, 노드 n에 대해 분할 변수 X_n이 수치형일 경우, 신뢰 구간(Confidence Interval) $[i^L_n, j^R_n]$구간에 대해 $i^L_n \le Xn \le j^R_n$를 만족하는 예제는 노드에 저장하고 있으며 신뢰 구간 경계에 대한 클래스 분포, $\theta^L_{n,Xn,x,i}$를 유지한다($\theta^L_{n,Xn,x,i} = |\{t \in F_n : t.X_n \le i^L_n \wedge t.C=i\}|/|F_n|$). 새로운 데이터가 추가되면 각 노드에서 유지하는 분포 정보는 갱신되며 수치형 분할 변수를 갖는 노드일 경우 신뢰구간 검사하여 이를 만족하는 예제를 노드에 저장한다.

필요한 분포가 모두 갱신된 후 트리의 검증은 시작되며 이는 범주형 변수의 클래스 분포 정보와 n^i_x, $\theta^L_{n,Xn,x,i}$등을 이용하여 다음과 같이 수행된다.

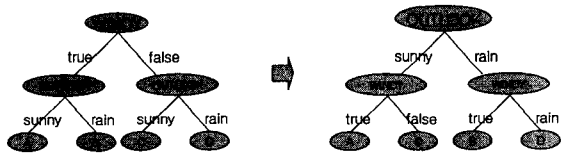
범주형 변수에 대한 평가 함수 값은 클래스 분포 정보를 이용하여 계산하며 수치형 변수에 대해서는 버켓 경계에 대한 분포 정보를 이용하여 상향 임계값을 구한다. 분할 변수가 범주형이면 다른 비분할 변수와 비교하여 현재 분할 변수가 가장 우수하면 각 서브 트리에 대해 검증을 계속 수행하고, 그렇지 않으며 현재 노드에서 트리를 재생성한다. 분할 변수가 수치형이면 이 변수의 평가 및 최종 분할 포인트는 $\theta^{L}_{n,Xn,x,i}$와 저장된 예제를 이용하여 계산되고 다른 변수와 비교된다. 분할 변수가 수치형이면, 이외에 분할 포인트가 신뢰구간의 경계를 넘어서는지도 검사한다. 만약, 상향 임계값이 계산된 포인트가 신뢰구간에 포함된다면 이는 최종 분할 포인트가 신뢰구간 내에 존재한다는 것으로 BOAT는 노드에 저장된 예제를 최종 분할 포인트를 이용하여 서브 트리에 전달하고 각 서브 트리에 대해 검증을 계속 수행하며 그렇지 않으면 BOAT는 현재 노드에서 트리를 재생성한다.

BOAT는 수치형 분할 변수에 대해 신뢰 구간과 이 구간에 포함되는 데이터를 유지하고 있기 때문에 신뢰구간 내에서의 분할 포인트 변경은 효율적으로 처리 할 수 있다. 그러나 분할 변수와 최종 분할 변수가 다른 경우와 최종 분할 포인트가 신뢰 구간을 넘어서는 경우에는 그 노드에서 트리를 다시 생성해야한다. 만약, 트리의 루트 노드의 분할 변수가 변경된다면 트리를 처음부터 다시 생성해야한다.

2.2 ITI

ITI는 최초의 메모리 기반 순차적 트리 생성 기법으로 트리 재구조화 기법을 이용하여 BOAT와는 달리 분할 포인트 변경이 필요한 상황뿐만 아니라 분할 변수 변경이 필요한 상황에서도 효율적으로 대처할 수 있다. 이를 위해 ITI는 결정 노드에 범주형 변수에 대해서는 변수가 갖는 각 값에 대한 클래스 빈도를 저장하고 수치형 변수일 경우에는, 각 값에 대한 클래스 빈도를 유지하고 값들을 정렬된 형태로 저장한다.

ITI는 새로운 예제를 기존 트리에 전달하고 노드 정보를 갱신한 후 새로운 예제를 반영하기 위한 트리 재구조화를 시작한다. 트리 재구조화는 트리의 루트로부터 시작되며 각 결정 노드의 분할 변수의 변경여부 혹은 분할 포인트의 변경여부를 체크한다. 트리 재구조화는 재귀적 호출에 의해 이루어지며 분할 변수 변경과 분할 포인트 변경은 유사한 방식으로 처리된다. [그림 1]은 루트 노드의 분할 변수를 WINDY에서 OUTLOOK으로 변경되어야할 상황이며 이 때 트리 재구조화는 다음과 같이 수행된다.



[그림 1 : 분할 변수 변경]

[그림 1]은 분할 변수를 WINDY에서 OUTLOOK으로 변경하려할 때 WINDY의 아들 노드의 분할 변수가 OUTLOOK인 상황이며 이 경우 약간의 포인터 변경으로 새로운 트리를 생성할 수 있다. 이와 같은 변경에서 루트 노드와 A, B, C, D 서브 트리는 이들이 이전에 의존한 예제들이 변경되지 않았으므로 노드 정보를 변경시킬 필요가 없다. 그러나 아들 노드의 경우, 사용한 예제들이 변경되었기 때문에 노드 정보를 변경해야한다. 즉, 예전의 왼쪽 서브 트리는 WINDY가 true인 예제들을 이용하여 트리를 구성되었으나 이제는 OUTLOOK이 sunny인 예제를 이용하여 트리를 구성해야하기 때문에 이에 맞게 루트 노드 정보를 갱신해야한다. 노드 정보 갱신은 손자 노드의 노드 정보를 이용하여 이루어진다.

3. 전역적 범주화 기법을 이용한 순차적 트리 생성

대용량 데이터에 대한 ITI의 한계점은 노드에 저장해야하는 정보의 양이 너무 많기 때문에 발생한다. 범주형 변수에 대해서는 클래스 분포 정보가 저장되어 크게 문제가 되지 않는다. 그러나 수치형 변수에 대해서는 클래스 정보가 연결된 수치 그 자체가 저장된다. 수치형 변수의 개수가 n개이고 트리의 깊이가 d이며 특정 수치형 변수가 k개의 값을 가질 때 n*d*k의 공간이 필요하다. 본 논문은 이러한 수치형 변수에 따른 문제점을 해결하기 위해 전역적 범주화 기법을 이용한 접근 방법을 제안한다. 또한 메모리에 저장하는 예제를 디스크에 저장함으로써 필요한 메모리 양을 축소한다.

3.1 디스크에 기반한 연속적 변수의 전역적 범주화

본 논문에서는 전역적 범주화 기법으로서 제타 기반 범주화 기법(Discretization Based on Zeta)[4]을 이용하였다. 제타를 사용한 이유는 다른 기법과 비교하여 정확도는 유사하지만 수행 속도 면에서 더 빠르기 때문이다. 제타를 이용한 범주화 기법은 특정 연속형 변수 A에 대해 A가 가지는 전체의 값을 하나의 범주로 고려하며 시작된다. 현 범주에서 가장 많은 빈도수를 갖는 클래스 변수의 값을 모달 클래스 값(Modal Class Value)이라 하고 모달 클래스 값이 가지는 빈도 수를 현 범주의 제타 값으로 정의한다. 범주화는 변수 A의 각각의 값으로 나누어 생성되는 서브 범주의 제타 값의 합이 가장 큰 값을 분할 포인트로 선택함으로써 이루어진다. 이러한 과정은 각각의 서브 범주에도 계속 수행되며 더 이상의 제타 값의 향상이 없을 때까지 반복된다.

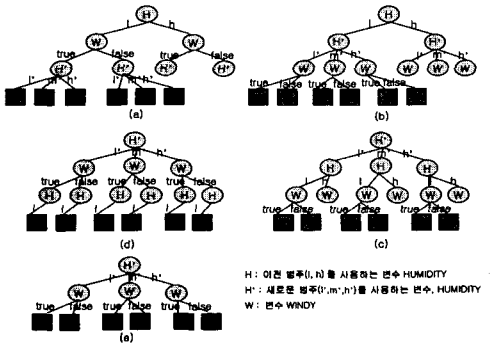
위와 같이 제타 기반 범주화 기법은 범주화를 수행하기 위해 연속형 변수가 갖는 값에 대해 클래스 분포 리스트를 필요로 한다. 그러나 해당 연속형 변수의 범위가 매우 크고, 데이터가 대용량일 경우, 이러한 분포 리스트가 매우 커지기 때문에 메모리에 모두 적재할 수 없다. 본 논문에서는 이를 위해 임의 접근 파일에서 범주화를 수행하는 방법을 제안한다.

임의 접근 파일의 장점 중의 하나는 파일내의 특정 위치에 접근하기 위해 그 이전 값을 모두 메모리에 적재 할 필요 없이 파일 안에서의 위치를 이용하여 직접 접근 할 수 있다는 점이다. 클래스 분포 리스트를 저장한 파일 안에서의 위치는 다음과 같이 계산된다. 값 자체를 저장하기 위해 m 바이트가 필요하고 클래스 빈도를 저장하기 위해 k 바이트가 필요하다면 하나의 값과 n개의 클래스 값을 위해 m+n*k 바이트가 필요하다. 따라서 첫 번째 값은 0*(m+n*k)에 위치하고 두 번째 값은 m+n*k에 위치한다. 이를 일반화하면 i 번째 값은 (i-1)*(m+n*k)에 위치하며 값의 순서는 배열의 인덱스처럼 사용할 수 있다.

3.2 범주 변화에 따른 트리 재구조화

트리 생성에 전역적 범주화를 이용하고 새로운 예제 추가에 따라 트리를 순차적으로 생성하기 위해서는 새로운 예제를 추가함으로써 발생하는 범주의 변화를 고려해야 한다. [그림 2]는 범주 변화에 따른 트리 갱신 과정을 나타내며 이는 트리 재구조화에 기반을 두고 있다. 지면상 트리의 잎 노드는 왼쪽 서브 트리만 나타내었다.

범주 변화에 따른 트리의 갱신은 [그림 2](a)와 같이 잎 노드에 존재하는 예제를 새로운 범주 H'로 나누면서 시작한다. 예제를 새로운 범주에 적용한 후에 해당 변수를 원래의 위치로 이동 시켜야한다. 즉, 초기 트리에서 H는 W의 자식 노드가 아닌 루트 노드의 분할 변수이기 때문에 그 위치로 이동 시켜야 한다. 이는 [그림 1]에서 분할 변수를 변경하는 방식과 같다. [그림 2](b)는 새로운 범주를 사용하는 H'를 W의 위치로 끌어올린 것을 나타내고, [그림 2](c)는 새로운 범주를 사용하는 H'를 이전 범주를 사용하는 H 위치로 끌어올림을 나타낸다.



[그림 2 : 범주 변화에 따른 트리 갱신 과정]

[그림 2](c)에서 루트에서 잎 노드까지 특정 경로에서 H는 분할 변수로서 두 번 사용된다. 그러나 일반적으로 모든 변수가 범주형이거나 모든 수치형을 범주화하여 사용할 경우, 특정 변수는 분할 변수로서 루트 노드에서 잎 노드까지 경로에 오직 한번만 사용된다. 따라서 [그림 2](c)에서 이전 범주를 사용하는 H를 모두 끌어내려야 한다. 이는 그 노드에서 분할 변수를 계산하고 선택하여 선택된 분할 변수를 끌어올림으로써 이전 범주를 사용하는 변수를 끌어내릴 수 있다. [그림 2](d)에서는 분할 변수로서 W가 선택되어 이전 범주를 사용하는 H가 내려갔음을 나타낸다. [그림 2](d)에서 이전 범주를 사용하는 H의 모든 서브 트리는 잎 노드이다. 이는 이전 범주를 사용하는 변수가 트리의 가장 아래 부분까지 내려왔음을 나타내며, 이전 범주를 사용한 변수를 분할 변수로 사용하는 것은 의미가 없으므로 이 부분을 잎 노드로 변환하였다. [그림 2](e)는 갱신된 최종 트리를 나타낸다.

3.3 분할 변수 변경에 따른 트리 재구조화

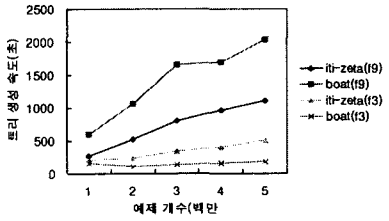
이제 트리의 구조는 새로운 예제의 추가에 따른 범주의 변화에 맞게 변경되었다. 그러나 아직 새로운 예제의 추가에 따른 분할 변수 변경과 이에 따른 트리 구조 변경은 아직 수행되지 않았다. 본 논문에서는 이를 위해 ITI를 기본 알고리즘으로 사용하였지만 대용량 데이터 처리를 위해 실행도중 메모리를 체크하여 한계에 도달하면 예제를 디스크에 저장하고 저장된 예제를 관리할 수 있도록 확장하였다.

4. 실험 및 성능 평가

본 논문의 접근 방법은 및 BOAT는 자바 및 JBuilder4.0을 이용하여 구현하였다. 메모리 기반 트리 알고리즘과 기본 자료구조를 위해 공개된 데이터 마이닝 패키지인 WEKA를 이용하였으며 정렬방법으로서 카운팅 정렬(Counting Sort)방법과 퀵정렬(Quick Sort)을 이용하여 속도를 향상시켰다. 모든 실험은 Pentium-III 1G, WINDOW 2000에서 수행되었으며 전체 256MB 메인 메모리 중 100MB만 설정하여 사용하였다.

본 논문에서는 대용량 데이터에 대한 성능을 평가하기 위해 Agrawal이 [5]에서 제안한 분석적 데이터를 이용하였다. 이 분석적 데이터는 9개의 변수와 한 개의 클래스 변수로 구성되어 있으며 10개의 분류 함수를 포함하고 있어서 다양한 복잡도에 따른 데이터를 생성할 수 있다. 본 실험에서는 분류 함수 3, 9를 사용하였다. 분류 함수 3은 두 개의 변수가 클래스 분류에 결정적인 역할을 수행하며 함수 9에서는 클래스 값이 다른 3개의 변수의 선형적 조합에 의해 의존한다. 실험은 예제의 개수를 1백만 레코드에서 5백만 레코드까지 1백만씩 순차적으로 추가해가며 트리 생성 속도를 비교하였다.

[그림 3]에서 함수-3에 대해서는 BOAT의 트리 생성 속도가 본 논문에서 제안하는 접근 방법보다 우수하였다. 그 이유는 저장하고 있는 이전 트리와 새로운 데이터를 추가한 후 생성된 트리의 형태가 같아서 서브 트리에서의 트리 재생성 작업이 필요하지 않았기 때문이다. 이러한 경우, BOAT에서는 데이터를 트리에 추가하고 필요한 빈도를 갱신하여 트리 재생성 여부를 결정하는 작업만 필요하다. 이에 비해, 제안하는 방법은 모든 데이터를 고려한 범주를 구하기 위해 연속형 변수를 정렬하고 디스크에 저장하는 오버헤드가 필요하였고 새로운 범주의 경계가 변경되어 기존의 트리를 갱신하였다. 그러나 함수-9와 같이 복잡도가 증가한 문제의 경우에는 BOAT의 경우, 분할 포인트가 신뢰구간을 넘어서는 경우가 종종 발생하였으며 이 노드에서 트리를 재생성하였다.



[그림 3 : 함수-3, 함수-9에 대한 실행속도]

5. 결론 및 향후 연구

본 논문에서는 기존의 대용량 데이터를 처리하지 못하는 ITI의 한계점을 극복하기 위해 전역적 범주화 기법을 이용한 접근 방법을 제안하였으며, BOAT와 비교할 만한 성능을 나타냄을 실험을 통해 보였다.

향후 연구로는 첫째, 범주화를 위한 다른 전역적 범주화 기법과 비교 실험이다. 본 논문에서 제시한 디스크 기반 범주화 기법은 임의 접근 파일을 이용하는 방법으로서 이 방법은 범주화를 위해 각각의 값에 대한 클래스 분포를 이용하는 모든 범주화 기법에 적용될 수 있을 것으로 기대된다. 둘째, 본 논문에서 사용한 순차적 범주 생성과 범주 변화 처리 기법을 범주화를 사용하는 다른 알고리즘에 적용할 수 있도록 확장하는 것이다. 본 논문에서는 순차적 트리 생성에 적용했지만, 이를 다른 순차적 알고리즘을 사용하는 베이시안 네트웍 및 군집화 알고리즘에 적용할 수 있을 것이다.

6. 참고문헌

[1] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. *Proceedings of the Fifth Int'l Conference on Extending Database Technology(EDBT)*, 1996.

[2] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat-optimistic decision tree construction. *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1999.

[3] P. E Utgoff, N. C. Berkman, and J. A Clouse. Decision tree inductiion based on efficient tree restructuring. *Machine Learning*, 29:5-44, 1997.

[4] K. M. Ho, and P. D. Scott. An Efficient Global Discretization Method. *Technical Report CSM-296*, Department of Computer Science, University of Essex, England.

[5] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914-952, 1993