< Apriori 알고리즘>

우선 첫 번째 단계에서는 최소 지지도 설정 값에 따라 빈도수가 높은 항목의 집합들을 찾아내고 그 다음 단계에서는 이들 집합들로부터 신뢰도 설정 값을 모두 뽑아냅니다.

Apriori 알고리즘에서 사용하는 중요한 법칙은 빈도수가 높은 항목 집합의 모든 부분 집합도 다 빈도수가 높다는 사실입니다. 예를 들어 데이터에 {빵, 버터, 우유}가 최소 지지도에 의해 빈도수가 높다면 당연히 {빵, 버터}만을 봐도 빈도수가 높고, {버터, 우유}을 봐도 빈도수가 높습니다.  다시 말해 어떤 집합이 주어졌을 때 새로운 항목을 더해주면 지지도는 절대로 전보다 증가할 수 없습니다.

Apriori 알고리즘은 우선 사이즈 한 개의 빈도수가 높은 항목들을 먼저 구하고 그 다음에 이것들을 이용해 사이즈가 두 개인 빈도수가 높은 항목들의 집합을 구하는 방식으로 한 사이즈씩 차례로 수행합니다. 그렇기 때문에 데이터에 있는 사이즈가 가장 큰 빈도 높은 항목 집합의 크기가 k라면 대략적으로 데이터를 k번 스캔하게 됩니다(여기서 집합의 사이즈란 그 집합에 들어 있는 원소 개수를 말합니다). 사이즈가 k인 빈도 높은 항목들을 지금 막 구한 단계라고 합시다. 이 때 이들을 이용해 사이즈가 k+1인 후보 항목들의 집합들을 먼저 구합니다.

예를 들어 <그림 1>에서 사이즈가 2인 빈번 아이템 항목 L2에서 {B, C}와 {B, E}를 이용해 {B, C, E}라는 사이즈가 3인 후보 항목들의 집합이 만들어집니다. 이 때 이 집합의 원소로 구성된 사이즈가 2인 모든 부분 집합이 사이즈 2인 빈도 높은 항목 집합들에 다 들어있는지 체크하고 만일 하나라도 없다면 후보에서 탈락시킵니다. 여기서 {B, C, E}는 그것의 부분집합 {B, C}, {B, E}, {C, E}가 모두 빈도 높은 항목 집합들에 들어 있으므로 후보 집합에 해당됩니다. 즉 {A, B, C}나 {A, C, E}가 후보 항목에 들어갈 수 없는 이유로 {A, C}나 {A, E}가 빈도 높은 항목 집합에 들어 있지 않기 때문입니다. 이런 식으로 후보들을 만든 후에는 실제 데이터를 스캔해 후보들을 카운트하고 그런 후에 지지도를 만족하는 것들만 뽑아내 사이즈가 3인 후보 항목 집합을 만들어냅니다. 그 다음에는 다시 이들을 이용해 사이즈가 4인 후보들을 만들어 내고 더 이상 후보 집합을 만들지 못할 때까지 같은 과정을 반복합니다.

간단하게 소비자가 산 물건의 목록을 가지고 있는 데이터베이스가 있다고 생각해보자.

한 소비자마다 구매한 물건의 목록이 하나 혹은 여러개 일수 있는데, 한 소비자마다 산 물건의 목록을 가진 데이터를 하나의 트랜잭션(transaction)이라고 보면, 여러 트랜잭션이 모여서 거대한 데이터베이스가 된다. 패턴(pattern)이라는 것은 물건들의 조합으로 생각하면 쉽다.예를 들어, 사과를 산 손님은 몇명이고, 사과와 귤을 같이 산 손님은 몇 명인지 알고 싶을때 <사과> <사과,귤>이런 것들을 반복적으로 나타낼 수 있는 패턴이라고 생각하면 된다.결국 의미있고 가치있는 ‘정보’가 되기 위해서는 패턴들이 일정 수 이상 반복해서 나타나야 한다. 그래서 많이 반복되는 패턴만을 골라내기 위해 minimum support threshold(minsup)라는 용어를 사용한다.반복이 몇%나 반복되었는지 그 기준을 제시한다.(support를 반복된 횟수라고 생각하면 된다)만약 트랜잭션이 5개이고 misup이 40%라면 5\*40%=2,즉 2번이상 반복되는 패턴을 찾겠다는 말이 된다. 이런 기본적인 조건을 가지고 데이터 베이스에서 minsup이상의 반복이 나타나는 모든 길이의 패턴을 전부 찾는 알고리즘이 Apriori알고리즘이다.

1. 사과 토마토
2. 사과 귤 오렌지
3. 사과 오렌지

이제부터 기본적인 알고리즘을 글로 설명해보면, a,b,c는 트랜잭션(여기선 소비자)이 되고 {사과,토마토,귤,오렌지}각각은 아이템이 된다.Apriori에서는 길이가 1인 패턴부터 길이를 점차 늘려가며 반복해서 Database를 읽는다.이 예제에서는 minsup을 66.666..%로 잡아보았다.(그래야 트랜잭션 수와 곱했을때 2가 나오니까)그러면 DB에서 두 번이상 반복되는 패턴들을 길이에 따라 모두 찾겠다는 것을 의미하게 된다.

우선 길이가 1인 패턴을 찾아보자.<사과:3><토마토:1><귤:1><오렌지:2> 이렇게 패턴마다 DB에서 나타나는 횟수(빈발도=support)를 구해보면 이렇게 나온다.여기서 아까 minsup에서 말한 것처럼 2번이상 반복되는 패턴만 찾겠다고 했다. 그러므로 토마토와 귤은 의미있는 패턴에서 탈락하고, 사과와 오렌지만 남는다. 이렇거 찾아낸 패턴들을 조합해서 다음 길이가 2인 패턴을 찾는다. 만약 여기서 의미있는 패턴이 사과 하나만 나왔다고 가정하면 길이가 2짜리인 패턴 후보를 만들 수 없으므로 알고리즘은 종료가 된다. 일단 지금은 조합할 수 있으니 사과와 오렌지를 조합해서 <사과,오렌지>라는 패턴 후보를 만들 수 있다.이렇게 만든 패턴 후보를 candidate라고 논문에서 정의한다.

사과와 오렌지를 이용해서 만들 수 있는 길이가 2짜리인 패턴이 더이상 없으니 다시 DB를 읽으며 <사과,오렌지>패턴이 몇 번 나오는지 세보면 된다. 트랜잭션b,c<사과,오렌지:2> 이렇게 빈발도를 구할 수 있다. 이제 이 패턴이 의미가 있나 확인해보니Minsup보다 높거나 같으므로 찾고자 하는 패턴이 맞다. 다음단계로 이전단계에서 나온 패턴들을 조합해서 길이가 3인 패턴후보를 뽑으려고 하니 사과와 오렌지로 더이상 조합을 할 수가 없다. 이시 점에서 알고리즘이 종료가 되고,모든 단계에서 구했던 minsup이상의 패턴과 빈발도는 <사과:3><오렌지:2><사과,오렌지:2>이렇게 3가지가 나오는 것을 확인할 수 있다. 이예제에서는 DB를 2번밖에 읽지 않았지만, 아이템이 많아지고 트랜젝션이 점점 많아지면 DB를 읽는 횟수는 거의 비례해서 증가한다. 한 번 읽을 때마다 모든 DB를 읽어야하므로 DB가 커질수록 실행 시간이 점점 길어질 것이다.

이렇게 구한 패턴을 어떻게 이용할까? 마트라고 생각해보면 위 예에서 사과와 오렌지를 같이 구입하는 손님이 여러 번 등장 했으므로 두 상품이 유사하거나 연관이 많다고 생각할 수 있다. 그래서 두 상품을 같이 진열해서 손쉽게 구입할 수 있게 해서 매출을 늘릴 수 있다.

Support(X=>Y) = P(X U Y) : X와 Y를 모두 포함한 거래, 거래 DB에서 주어진 룰을 만족시키는 거래비율

Confidence(X=>Y) = P(Y | X) : 연관성의 확실한 정도를 평가하는 값