1

순차 패턴

데이터 마이닝 분야 중 하나인 순차 패턴(sequential pattern)은 데이터에 공통으로 나타나는 순차적인 패턴을 찾아내는 것이다. 순차 패턴은 CRM이나 웹 사이트 내비게이션 디자인 등에 응용되고 있다.

임종화 KAIST 전산학과 igkst@kaist.ac.kr

데이터 마이닝 분야 중 하나인 순차 패턴(sequential pattern)은 데이터에 공통으로 나타나는 순차적인 패턴을 찾는 문제다. 예를 들어 대형 슈퍼마켓에서 고객들의 물품 구매 행태를 데이터베이스에 저장한다고 하자. 고객 A는 처음에는 담배와 술을, 다음날에는 담배와 신문을, 그 다음날에는 음료수와 과자를 구매했다고 하면 이러한 구매 패턴을 각각 '{담배, 술}' 이라는 트랜잭션(한 번에 구매한 물품들의 집합), '{담배, 신문}' 이라는 트랜잭션, '{음료수, 과자}' 라는 트랜잭션으로 나타낼 수 있다. 이들 트랜잭션의 시간적 순서를 고려하면 '({담배, 술}, {담배, 신문}, {음료수, 과자})' 와 같은 트랜잭션들의 시퀀스(이하, ()로 나타냄)로 나타낼 수 있을 것이다. 그 외에 다른 고객의 구매 패턴도 이런 식의 시퀀스들로 데이터베이스에 저장되어 있을 것이다. 이 시퀀스를 사용자 시퀀스라고 한다. 이 때, 모든 사용자 시퀀스 중 몇 % 이상 공통으로 나타내는 시퀀스를 찾는 것이 순차 패턴 마이닝이다.

이런 순차 패턴 마이닝을 어디에 어떻게 사용할 수 있을까. 대표적으로 앞의 예와 같은 사용자 구매 시퀀스의 패턴을 조사해, 고객이 다음 방문시 구매할 것으로 예상되는 물품에 대한 선전물이나 쿠폰을 집으로 보내주는 등의 서비스를 할 수 있을 것이다. 이러한 서비스를 웹에 적용해 생각해 보면 전자상거래 사이트에서 고객별로 데이터베이스에 구매 패턴을 저장해 놨다면, 어떤 고객이 향후 방문시 구매할 것으로 예상되는 물품에 대한 광고나 배너(banner), 또는 하이퍼링크를 화면에 나타내 줌으로써 매출을 올리거나 CRM(Customer Relationship Management) 등에 사용할 수 있다. 또 다른 적용 분야로서, 웹 구조 디자인에 사용할 수 있다. 웹 내비게이션 패턴 역시 순차 패턴이기 때문에, 내비게이션 패턴을 분석해, 사용자들이 좀더 효율적인 내비게이션을 하도록 웹 링크의 구조를 바꿀 수 있다. 이런 순차 패턴을 찾는 알고리즘은 현재까지 크게 두 가지로 분류할 수 있다. IBM 연구소에서 개발한 GSP(Generalized Sequential Pattern) 알고리즘과 캐나다의 Simon Fraser 대학에서 개발한 PrefixSpan 알고리즘이 그것이다. GSP 알고리즘을 중심으로 이들 알고리즘들을 차례대로 살펴보자.

GSP 알고리즘

순차 패턴 마이닝에서는 사용자가 지지도(support)라는 값을 정해 준다. 지지도란 어떤 패턴이 전체 데이터베이스의 몇 개(또는 몇 %)의 사용자 시퀀스에서 등장하는가를 뜻하는 것이다. 순차 패턴 알고리즘은 데이터베이스에서 사용자가 정해준 지지도 이상 나타나는 패턴을 찾아준다. 이렇게 사용자가 정해준 지지도보다 많은 사용자 시퀀스를 포함하고 있는 패턴을 빈번한 패턴(frequent pattern)이라고 말한다. 앞으로 길이가 k인 빈번한 패턴들의 집합을 F_k라고 말할 것이다. 빈번한 패턴과 함께 등장하는 개념은 후보 패턴(candidate pattern)이다. 후보 패턴은 빈번한 패턴이 될 가능성이 있는 패턴으로, 알고리즘은 후보 패턴들의 실제 등장 빈도를 데이터베이스에서 카운트해 빈번한 패턴을 찾게 되는 것이다. 빈번한 패턴의 경우와 마찬가지로 앞으로는 길이가 k인 후보 패턴들의 집합을 C_k라고 말할 것이다.

GSP에서는 후보 패턴을 찾아낸 뒤, 데이터베이스를 조사하면서 그 후보 패턴이 실제로 데이터베이스에 몇 번 나오는지를 세어본 후, 사용자가 정해준 최소 지지도 이상인 패턴들만을 모아 빈번한 패턴으로 찾아준다. 그러면, 후보 패턴은 어떻게 만들어지는 것일까. 이들 후보 패턴들은 다음과 같은 사실을 이용해 만들 수 있다.

어떤 패턴이 빈번하면 그 패턴의 일부분도 모두 빈번하다.

즉, 예를 들어 ({신문}, {우유}, {초콜릿})이 빈번한 패턴이라면 일부분인 ({신문}), ({우유}), ({초콜릿}), ({신문}, {우유}), ({신문}, {초콜릿}), ({우유}, {초콜릿}), ({신문}, {우유}, {초콜릿})도 모두 빈번한 패턴이어야 한다는 뜻이다. 이 중에서 하나라도 빈번한 패턴이 아니라면 ({신문}, {우유}, {초콜릿})은 빈번한 패턴이 될 수가 없다. 이 사실을 이용해 GSP에서는 크기가 1인 후보 집합부터 시작해 빈번한 집합을 찾아간다. 즉, $C_1 \rightarrow F_1 \rightarrow C_2 \rightarrow F_2 \rightarrow C_3 \rightarrow F_3 \rightarrow \cdots$의 순서로 점점 길이가 긴 빈번한 패턴을 찾아가는 것이다. C_1은 F_0이 존재하지 않으므로, 데이터베이스에 등장하는 모든 아이템이 될 것이며 C_{k+1}은 F_k로부터 만들어 낼 수 있다(만들어 내는 과정은 관계형 데이터베이스의 조인(join) 연산자를 통한 후보 패턴 발견 과정과 발견된 후보 패턴 중의 일부분이 빈번하지 않을 경우 후보 패턴에서 삭제하는 삭제(pruning) 과정의 두 과정으로 이루어져 있다). 그리고 어느 순간 F_k가 하나도 발견되지 않거나 F_k로부터 C_{k+1}을 하나도 만들어낼 수 없을 때, GSP 알고리즘을 끝내게 된다. 전체적인 알고리즘은 **〈리스트 1〉**과 같다.

〈리스트 1〉 GSP 알고리즘

```
F1 = 빈번한 아이템들
for(k=2;Fk-1≠?;k++) do
{
   Ck = Fk-1로부터 만들어낸 길이 k인 후보 패턴들
   for each DB에 저장된 사용자 시퀀스 c do
      Ck 중 c의 부분 시퀀스(subsequence)인
      것들의 카운트를 1씩 증가시킨다.
   Fk = Ck 중에서 카운트가 최소 지지도 이상인 것들
}
결과 = F1 ∪ F2 ∪ F3 ∪ ... ∪ Fk
```

예를 통해 실제로 GSP 알고리즘이 어떤 과정을 거쳐 결과를 얻어 내는지 살펴보도록 하자. **〈표 1〉**은 GSP 알고리즘의 입력으로 들어온 데이터베이스의 내용이며, **〈표 2〉**는 지지도를 100%(네 명)로 했을 때, GSP 알고리즘이 얻는 단계별 결과이다.

〈표 2〉에서 볼 수 있듯이 C_1은 모든 아이템들이 후보가 될 수 있을 것이며, 그 중에서 데이터베이스에 네 명(100%) 이상의 시퀀스에 등장하는 아이템들이 F_1이 된다. 이들 F_1로부터 C_2를 만든다. 그리고 다시 데이터베이스를 살피면서 카운트해 사용자가 정해준 지지도 이상인 빈번한 패턴들인 F_2를 찾는다. F_2를 이용해 C_3를 만들어 보면 아무것도 만들어지지 않으므로 알고리즘을 끝내게 된다.

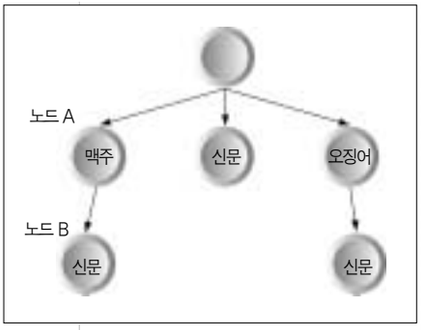
〈표 1〉 데이터베이스에 저장된 사용자 시퀀스들

고객 번호	구매 기록
1	({맥주, 땅콩}, {맥주, 오징어}, {신문})
2	({신문}, {오징어}, {소주}, {맥주}, {신문, 땅콩})
3	({맥주, 신문}, {오징어}, {오징어, 땅콩}, {신문})
4	({맥주, 오징어}, {신문, 양주}, {신문, 오징어})

〈표 2〉 지지도 100%일 때의 단계별 결과

패턴의 길이	후보 패턴	빈번한 패턴
1	({맥주}), ({신문}), ({오징어}), ({땅콩}), ({소주}), ({양주})	({맥주}), ({신문}), ({오징어})
2	({맥주}, {맥주}), ({맥주, 신문}), ({맥주}, {신문}), ({맥주, 오징어}), ({맥주}, {오징어}), ({신문}, {맥주}), ({신무}, {신문}), ({신문, 오징어}), ({신문}, {오징어}), ({오징어}, {맥주}), ({오징어}, {신문}), ({오징어}, {오징어})	({맥주}, {신문}), ({오징어}, {신문})
3	없음	없음

〈그림 1〉 PrefixSpan 트리



PrefixSpan 알고리즘

PrefixSpan은 기존의 GSP 방법이 후보 패턴을 만들고, 그 후보 패턴이 데이터베이스에 몇 번 나오는가 세느라 시간이 걸리는 단점을 없애기 위해, 후보 패턴을 만들지 않으면서 빈번한 패턴을 찾는 방법이다. PrefixSpan에서는 PrefixSpan 트리를 만들어 가면서 빈번한 패턴을 찾게 된다. **〈그림 1〉**의 트리는 {({맥주}), ({신문}), ({오징어}), ({맥주}, {신문}), ({오징어}, {신문})}이 빈번한 패턴의 집합임을 의미한다. **〈그림 1〉**을 보면, 노드 A는 빈번한 패턴 '({맥주})'를 나타내고 있고, 노드 B는 빈번한 패턴 '({맥주},{신문})'을 나타낸다. 앞으로는 '({맥주}) 노드'와 '({맥주}, {신문}) 노드'와 같은 식으로 부르기로 한다.

그러면 PrefixSpan 트리를 만들어 가는 과정을 살펴보자. 루트 노드에서 시작해 깊이 우선 검색(Depth First Search) 순서로 노드를 확장해 가면서 트리를 만든다. 노드를 확장할 때에는 노드 확장 이외에 projected

〈그림 2〉 ({맥주})로 확장된 PrefixSpan 트리

고객 번호	구매 기록
1	({맥주, 땅콩}, {맥주, 오징어}, {신문})
2	({신문}, {오징어}, {소주}, {맥주}, {신문, 땅콩})
3	({맥주, 신문}, {오징어}, {오징어, 땅콩}, {신문})
4	({맥주, 오징어}, {신문, 양주}, {신문, 오징어})

고객 번호	구매 기록
1	({_땅콩}, {맥주, 오징어}, {신문})
2	({신문, 땅콩})
3	({_신문}, {오징어}, {오징어, 땅콩}, {신문})
4	({_오징어}, {신문, 양주}, {신문, 오징어})

〈({맥주})-projected DB〉

〈그림 3〉 ({맥주}, {신문})으로 확장된 PrefixSpan 트리

고객 번호	구매 기록
1	Null
2	({_땅콩})
3	Null
4	({_양주}, {신문, 오징어})

〈({맥주}, {신문})-projected DB〉

DB도 만들게 된다. projected DB란 전체 데이터베이스의 사용자 시퀀스 중에서 그 노드가 나타내는 빈번한 시퀀스를 포함하고 있는 사용자 시퀀스만을 모은 후 노드가 나타내는 시퀀스 이후 부분만을 저장해 놓은 데이터베이스를 말한다. 그리고 N-projected DB란 'N 노드'에서의 projected DB를 말한다.

예를 들면 **〈표 1〉**의 데이터베이스에서 '({맥주}) 노드'의 projected DB인 ({맥주})-projected DB는 고객번호 1의 구매 시퀀스인 ({맥주, 땅콩}, {맥주, 오징어}, {신문})으로부터 ({맥주})의 뒷부분인 ({_땅콩}, {맥주, 오징어}, {신문})을 만들고(트랜잭션 내부의 '_'는 '맥주'가 같은 트랜잭션 안에 있다는 것을 의미한다), 고객번호 2의 ({신문}, {오징어}, {소주}, {맥주}, {신문, 땅콩})으로부터 ({신문, 땅콩})을 만든다. 이런 식으로 고객번호 3의 ({맥주, 신문}, {오징어}, {오징어, 땅콩}, {신문})으로부터 ({_신문}, {오징어}, {오징어, 땅콩}, {신문})을, 고객번호 4의 ({맥주, 오징어}, {신문, 양주}, {신문, 오징어})로부터 ({_오징어}, {신문, 양주}, {신문, 오징어})를 만들면 **〈그림 2〉**의 projected DB를 얻게 된다. PrefixSpan 트리의 루트 노드는 널 시퀀스(null sequence-({}), 즉 0개의 아이템으로 이루어진 시퀀스)를 나타내게 되고, 모든 사용자 시퀀스는 널 시퀀스를 포함하므로 루트 노드의 projected DB는 전체 데이터베이스가 된다. PrefixSpan 트리에서는 각 노드에서의 projected DB에서 F_1을 찾음으로써, 다음 노드로 확장을 하는 것이다.

〈표 1〉의 사용자 시퀀스 예제에서 GSP에서와 같이 F_1을 찾아보면 {({맥주}), ({신문}), ({오징어})}가 된다. 이 중에서 먼저 '맥주'부터 깊이 우선 검색 순서로 트리의 노드를 확장해 보면 **〈그림 2〉**와 같은 트리와 ({맥주})-projected DB를 갖게 된다.

'({맥주}) 노드'에서의 projected DB에서 지지도 100%(네 명)를 만족시키는 아이템은 '신문'밖에 없으므로 ({신문})으로 다시 노드를 확장하면 **〈그림 3〉**과 같이 된다.

'({맥주}, {신문}) 노드'에서는 더 이상 빈번한 아이템이 없으므로 다시 깊이 우선 검색 순서에 따라 루트 노드에서 다른 아이템들로 노드를 확장하게 된다('({{맥주})-projected DB'에서 빈번한 아이템은 '신문'밖에 없었다). 결과적으로 **〈그림 1〉**의 트리를 만들어 내고 알고리즘이 끝나게 된다.

그 밖의 다른 알고리즘들

앞의 알고리즘들을 적용해 나오는 결과 중에 특정 시퀀스를 포함하는 패턴들에게만 특별히 관심이 있는 경우가 있다. 이런 경우에는 그 외의 패턴들은 전혀 불필요한 정보일 뿐이며, 오히려 계산시에 불필요한 계산의 양만 늘어나게 된다. 이러한 문제를 해결하는 방법으로 후보 패턴을 만들 때 사용자가 관심 있는 시퀀스를 정규식(regular expression) 형태로 나타내면 그 조건을 만족하는 패턴들만을 고려하는 SPIRIT이라는 알고리즘도 있다.

순차 패턴에 대한 소개를 마치면서

순차 패턴은 실생활에 있는 많은 순차적인 흐름 속에서 유용한 정보를 찾아내는 것이다. 시간의 흐름과 같은 순차적인 흐름은 어쩌면 인간 생활 내부에 근본적으로 내재된 것이므로 앞으로 더 많은 이용분야가 생겨나리라 믿는다.

앞에서 언급한 알고리즘에 대해 간단히 설명했지만 실제로는 방대한 데이터를 다루어야 하는 시간이 오래 걸리는 작업이므로, CPU 사용을 줄이기 위해 다양한 자료 구조와 방법을 사용하고 있다. 그에 대한 자세한 내용이 알고 싶으면 참고문헌들을 찾아보기 바란다.

정리 : 송우일 wooil@sbmedia.co.kr

참고자료

❶ R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements", the 5th International Conference on Extending Database Technology (EDBT), Avignon, France, March 1996

❷ Jian Pei, Jiawei Han, Behzad Mortazavi-Asi and Helen Pinto, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", the 17th International Conference on Data Engineering (ICDE' 01), Heidelberg, Germany, April 2001

❸ Minos N. Garofalakis, Rajeev Rastogi and Kyuseok Shim, "SPIRIT: Sequential Pattern Mining with Regular Expression Constraints", the 25th International Conference on VLDB, Edinburgh, Scotland, UK, 1999