데이터 카디널리티에 따른 FP-Growth 알고리즘의 효율성 분석

김진형*, 김병욱*‡
*동국대학교 경주캠퍼스 컴퓨터공학과
e-mail: usebut@naver.com, bwkim@dongguk.ac.kr

Analysis of efficiency of FP-Growth algorithm based on data cardinality

Jin-Hyung Kim*, Byoung-Wook Kim*‡
*Dept of Computer Engineering, Dongguk University-Gyeongju

요 약

서로 다른 아이템 집합의 연관성을 분석하는 것을 연관규칙분석이라 한다. 대표적인 알고리즘으로 Apriori 알고리즘이 있지만 DB스캔 횟수가 많아질 수 있고 후보 집합 생성으로 인해서 속도가 느려질 수 있다는 단점이 있다. 이를 효율적으로 개선한 FP-Growth 알고리즘을 구현하여 임의의 데이터를 이용하여 알고리즘의 속도에 대해 연구한다.

1. 서론

1.1 논문 개요

연관규칙분석이란 서로 다른 두 개 이상의 아이템 집합이 얼마나 빈번히 발생하는지를 분석한다. 즉, 어떤 집단의 구매 활동 항목들을 바탕으로 항목들 사이에서 발생한 연관성, 지지도, 신뢰도를 바탕으로 분석하는 것이다. 연관규칙분석 알고리즘의 예로 Apriori 알고리즘이 있다. Apriori 알고리즘은 후보 집합 생성 시에 아이템의 개수가 많아지면 계산 복잡도가 증가하게 된다. 또한, 패턴을 찾기 위해서 DB를 스캔하는 횟수가 최대로 가장 긴 트랜잭션의 아이템의 수만큼 발생할 수 있다. 이러한 문제들을 해결하기 위해서 DB 스캔 횟수를 줄이고 후보 집합을 생성시키지 않도록 만들어진 알고리즘이 FP-Growth이다.

2. FP-Growth 알고리즘

2.1 FP-Growth의 정의

FP-Growth 알고리즘은 Apriori 알고리즘에서 단점으로 생각되었던 DB 스캔 횟수와 후보 집합 생성을 효율적으로 개선한 알고리즘이다. 각 항목들의 지지도를 계산하여 FP-Growth 알고리즘만의 독특한 방식으로 헤더테이블과 트리 및 연결리스트 구조를 생성한다.

2.2 FP-Growth 알고리즘 진행

FP-Growth 알고리즘을 진행하기 위해서 입력으로 파일이 전달되어야 한다. 파일의 내용은 각 항목들의 내용은 공백을 기준으로 나열되어야 하고, 항목을 이루는 트랜잭션은 개행을 기준으로 작성되어야 한다.

파일이 알고리즘의 입력으로 주어지면 트랜잭션의 수를 기준으로 최소 임계치(Minimum support)를 설정한다. 본 논문에서는 최소 임계치를 트랜잭션 수의 40%로 설정하고 트랜잭션과 아이템은 표-1과 같이 설정한다.

<표 1> 데이터베이스에서 실행중인 트랜잭션의 예

Transaction ID	Itemset
100	f, a, c, d, g, i, m, p
200	a, b, c, f, l, m, p
300	b, f, h, j, o
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

예시와 같이 입력이 주어지면 트랜잭션의 수가 5개이고 최소 임계치는 2가 된다. 각 항목별로 빈발도(support)를 계산한다. 빈발도가 최소 임계치보다 작다면 제거하고 남은 항목들은 빈발도를 기준으로 내림차순으로 정렬한다. 정렬된 테이블을 통해서 기존 데이터베이스 테이블도 최소 임계치보다 높은 항목들만 <표 2>의 순서에 맞게 정렬한다. 정렬된 결과는 <표 3>과 같다.

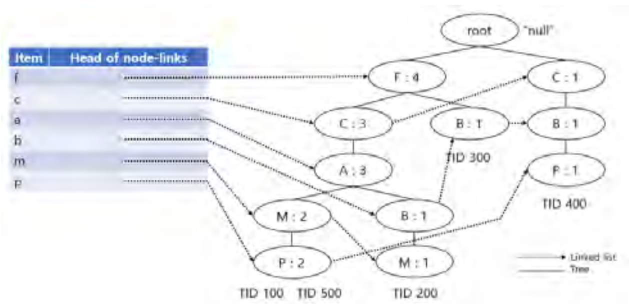
<표 2> 각 항목들의 빈발도 계산

Itemset	Support
f	4
c	4
a	3
b	3
m	3
p	3

<표 3> 정렬된 데이터베이스

Transaction ID	Ordered Frequent Itemset
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

정렬된 테이블을 기준으로 FP Tree를 구성한다. 트리의 루트노드는 “null”의 상태를 가진다. 트랜잭션의 순서대로 트리에 노드를 삽입하는데, 삽입될 노드와 항목이 일치한다면 해당 노드의 카운트를 증가시키고 그렇지 않다면 새로운 노드를 생성하여 트리를 구성한다. 이때, 각 항목의 노드들은 헤더 테이블을 통해서 연결리스트로 이루어진다.



(그림 1) FP Tree

3. FP-Grwoth 구현

3.1 FP-Growth 구조

main method에서 알고리즘의 입력이 되는 파일과 출력이 될 파일과 최소 임계치의 비율을 설정하고 알고리즘의 핵심이 되는 FP Grwoth 클래스의 run 메소드에 인자로 전달하여 호출한다. 그림 2의 경우, 입력이 되는 파일은 TestData.txt이고, 출력파일은 해당 경로 내의 TestData_FP_OUTPUT.txt에 저장된다. 최소 임계치에 해당하는 minsup은 Minimum Support로서, 이후에 트랜잭션의 수에 해당하는 transactionCount와 곱하여 최소 임계치의 수치를 정하게 된다.

```
String input = "TestData.txt";
String output = ".//TestData_FP_OUTPUT.txt";

double minsup = 0.4; // Minimum Support
AlgoFPGrowth algo = new AlgoFPGrowth();

algo.runAlgorithm(input, output, minsup);
algo.printStats();
```

(그림 2) 메인 클래스의 구조

그림 3의 경우 FP Tree를 구성할 클래스의 객체를 생성하고, 입력받은 파일로부터 각 항목들의 빈발도를 계산하여 최소 임계치보다 높은 항목들은 transaction 객체에 해당 item의 정보를 저장한다. Collections의 sort를 통해서 저장된 transaction의 항목들을 정렬한다. 정렬된 transaction 객체를 인자로 전달하여 addTransaction 메소드를 호출해서 트리 구조를 생성한다.

```
for(String itemString : lineSplited){
    Integer item = Integer.parseInt(itemString);
    // only add items that have the minimum support
    if(mapSupport.get(item) >= minSupportRelative){
        //minSupportRelative = minsup * transactionCount
        transaction.add(item);
    }
}
```

(그림 3) transaction 객체 생성 과정

```
Collections.sort(transaction, new Comparator<Integer>(){
    public int compare(Integer item1, Integer item2){
        int compare = mapSupport.get(item2) - mapSupport.get(item1);
        if(compare == 0){
            return (item1 - item2);
        }
        return compare;
    }
});
tree.addTransaction(transaction);
```

(그림 4) transaction 정렬 및 Tree 객체 노드 추가 과정

fpgrowth 메소드를 호출하여 그림 5와 같이 각 트리의 경로를 계산하여 해당 경로가 singlePath인지 판별한다. 해당 경로가 singlePath인 경우 output 파일에 경로를 출력하게 된다.

```
boolean singlePath = true;
int position = 0;
if(tree.root.childs.size() > 1) {
    singlePath = false;
}else {
    FPNode currentNode = tree.root.childs.get(0);
    while(true){
        if(currentNode.childs.size() > 1) {
            singlePath = false;
            break;
        }
        fpNodeTempBuffer[position] = currentNode;
        position++;
        if(currentNode.childs.size() == 0) {
            break;
        }
        currentNode = currentNode.childs.get(0);
    }
}
```

(그림 5) singlePath 판별 과정

트리의 경로가 singlePath가 아닌 경우, 그림 6과 같이 treeBeta를 구성하여 fpgrowth 메소드를 재귀호출하여 경로를 다시 계산한다.

```
FPTree treeBeta = new FPTree();
for(List<FPNode> prefixPath : prefixPaths){
    treeBeta.addPrefixPath(prefixPath, mapSupportBeta, minSupportRelative);
}
if(treeBeta.root.childs.size() > 0){
    treeBeta.createHeaderList(mapSupportBeta);
    fpgrowth(treeBeta, prefix, prefixLength+1, betaSupport, mapSupportBeta);
}
```

(그림 6) fpgrowth 구조

3.2 FP-Growth 실행결과

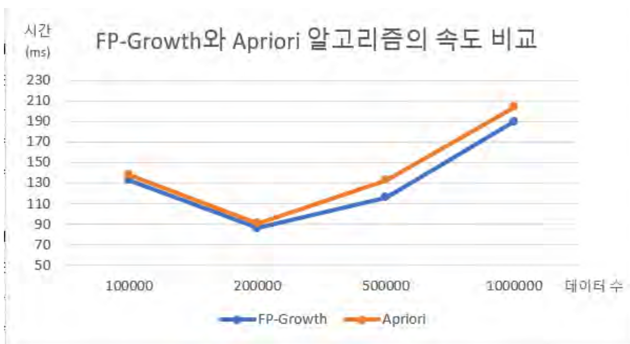
데이터의 개수를 10만개, 20만개, 50만개, 100만개로 설정하여 FP-Growth와 Apriori 알고리즘에 각각 적용하여 실행하였다. Apriori 알고리즘의 경우 평균 141.5ms의 실행시간을 보였다. 하지만, FP-Growth 알고리즘의 경우 평균 131.25ms의 실행시간을 보였다. 데이터 개수가 가장 적은 10만개의 실험을 기준으로 FP-Growth 알고리즘은 Apriori 알고리즘과 5ms의 시간이 차이가 났기 때문에 데이터 개수가 많아질수록 속도의 차이는 더욱 커질 것으로 예상된다. 다음의 표 4와 표 5는 각 알고리즘의 실행결과 표이고, 그림 7은 결과를 바탕으로 그린 그래프이다.

<표 4> FP-Grwoth 알고리즘의 실행 결과

Data	100,000	200,000	500,000	1,000,000
Time (ms)	133	86	116	190
Memory(mb)	15.88	67.38	69.38	47.45
Frequent itemset	6	3	4	4
Transaction count	23,582	48,913	123,704	249,104

<표 5> Apriori 알고리즘의 실행 결과

Data	100,000	200,000	500,000	1,000,000
Time (ms)	138	91	133	204
Memory(mb)	3.23	31.73	32.99	49.51
Frequent itemset	6	3	4	4
Transaction count	21	6	10	10



(그림 7) 속도 비교 그래프

4. 결론

본 논문에서는 연관규칙분석에 대한 알고리즘인 Apriori 알고리즘의 단점이었던 데이터베이스 스캔 횟수와 후보 집합 생성으로 인한 속도저하를 해결하는 FP-Growth 알고리즘에 대해서 다뤘다. 트리와 연결리스트 자료구조를 사용하여 데이터베이스 스캔 횟수를 줄이고 후보 집합을 생성하지 않는다는 점에서 Apriori 알고리즘을 효율적으로 개선한 알고리즘이라고 판단된다.

참고문헌

[1] Jiawei Han, Jian Pei, Yiwen Yin, Runyung Mao "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach" April 2001.

[2] Rakesh Agrawal, Ramakrishnan Srikant "Fast Algorithms for Mining Association Rules"

[3] Ke Wang, Liu Tang, Jiawei Han, Junqiang Liu "Top Down FP-Growth for Association Rule Mining"