# Apriori Algorithm

## 0. Introduction

- Lecture: 데이터사이언스(ITE4005)
- Student: 한양대학교 컴퓨터소프트웨어학부 2017030191 이현지
- Programming Assignment #1: find association rules using the Apriori algorithm

## 1. Main Concepts

### Summary

1. Main Idea

> 💡 input file로부터 transactions를 가져온 후, 초기 candidates을 구한다. 길이를 하나씩 늘려가며 각 길이의 candidates을 구하고 더이상 candidates이 만들어지지 않을 때까지 반복한다. 코드를 실행할 때 arguments로 받은 min support 이상의 support를 가진 candidates만 frequent pattern에 추가될 수 있다.

2. Support
   - transaction이 item_set과 associative_item_set을 함께 가지고 있을 확률
3. Confidence
   - item_set을 가지는 transaction이 associative_item_set도 가지고 있을 조건부 확률

### Step by Step (Main.class)

1. Set variables

```
Double minSup = Double.valueOf(args[0]);
String inputFile = args[1], outputFile = args[2];

Set<Set<Integer>> notFrequent = new HashSet<>();
```

   - minSup: 소스 코드 실행 시 인자로 받는 min support
   - inputFile/outputFiile: 소스 코드 실행 시 인자로 받는 input/output 파일의 이름
     - 경로를 명시하지 않으면 jar파일과 같은 경로에 존재
   - notFrequent: candidates을 생성할 때 min support 보다 작아서 frequent pattern이 될 수 없는 item set을 Set 타입(순서 중요 X)으로 수집
2. Get transactions from input file

```
ArrayList<Set<Integer>> transactions = getTransactions(inputFile);
```

   - transactions: 인자로 받은 input 파일 이름을 parameter로 하여 모든 transactions을 받아옴
     - 각 transaction은 같은 item set을 가지고 있을 수 있기 때문에 ArrayList로 선언
3. Get C1 from transactions

```
Map<Set<Integer>, Double> C1 = new HashMap<>();
for ( Set<Integer> transaction : transactions ) {
    for ( Integer nowItem : transaction ) {
        Set<Integer> nowItemSet = new HashSet<>();
        nowItemSet.add(nowItem);
```

```
        Set<Set<Integer>> C1KeySet = C1.keySet();
        if(!C1KeySet.contains(nowItemSet)){
            Double nowSupport = getSupport(nowItemSet, transactions);
            C1.put(nowItemSet, nowSupport);
        }
    }
}
```

- C1: 초기 candidates의 Map (길이 1)
  - 각 transactions을 순회하며 C1에 item(set)과 계산한 support 값을 추가
  - Key: item set
  - Value: support

4. Get L1 generated from C1

```
Map<Set<Integer>, Double> L1 = new HashMap<>(C1);
Iterator<Set<Integer>> L1Iter = L1.keySet().iterator();
while(L1Iter.hasNext()){
    Set<Integer> nowItemSet = L1Iter.next();
    Double nowSupport = L1.get(nowItemSet);

    if(nowSupport < minSup){
        L1Iter.remove();
        notFrequent.add(nowItemSet);
    }
}
```

- L1: 초기 frequent item set들의 Map
  - C1에서 support 값이 min support 미만인 item set을 제거한 맵

5. Get frequent item sets with loop

```
Map<Set<Integer>, Double> C = new HashMap<>(C1);
Map<Set<Integer>, Double> Lk = new HashMap<>(L1);
Set<Set<Integer>> L = new HashSet<>(L1.keySet());

for(int k = 2 ; !C.isEmpty() ; k++){
    C = getC(Lk, k, transactions, notFrequent);
    if(C.isEmpty()) break;

    Lk = getL(C, notFrequent, minSup);
    L.addAll(Lk.keySet());
}
```

- L: 각 단계에서 생성된 모든 frequent item set들의 Set
  - support를 굳이 취하지 않고 item set들만을 수집

6. Get association rules

```
Set<ArrayList<Set<Integer>>> associations = getAssociations(L);
```

- associations: L을 파라미터로 하여 association rules를 수집
  - 각 association은 item set과 그 associative item set의 순서 → ArrayList

7. Get the results of specific format

```
printOutput(associations, transactions, outputFile);
```

- printOutput: 앞에서 구한 association rules의 support와 confidence를 구하여 명시된 형태로 출력하는 함수

# 2. Details

### getTransactions()

1. Specification
    - Paramter: String inputFile
    - Return type: ArrayList<Set<Integer>>
2. Description

> BufferedReader와 FileReader를 통해 파라미터로 받아온 inputFile의 이름을 가진 파일을 열고 각 줄을 읽는다. StringTokenizer를 이용하여 \t로 구별된 각 item들을 Set 타입으로 받아 ArrayList에 넣는다.

3. Code

```
public static ArrayList<Set<Integer>> getTransactions(String inputFile){

    ArrayList<Set<Integer>> transactions = new ArrayList<>();

    try {
        BufferedReader reader = new BufferedReader(new FileReader(inputFile));
        String line = reader.readLine();

        while (line != null) {
            StringTokenizer tokenizer = new StringTokenizer(line, "\t");
            Set<Integer> nowItemSet = new HashSet<>();

            while(tokenizer.hasMoreTokens()){
                Integer nowItem = Integer.valueOf(tokenizer.nextToken());
                nowItemSet.add(nowItem);
            }

            transactions.add(nowItemSet);
            line = reader.readLine();
        }
        reader.close();

    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    return transactions;
}
```

## makeFormat()

1. Specification
    - Parameter: ArrayList<Set<Integer>> association, ArrayList<Set<Integer>> transactions
    - Return type: String
2. Description

> item_set과 associative_item_set의 ArrayList 형태로 들어온 association의 support와 confidence를 구하여 \t로 구분된 한 줄의 String을 만든다. support와 confidence는 소수점 셋째 자리에서 반올림하여(%.2f) 구한다.

3. Code

```
public static String makeFormat(ArrayList<Set<Integer>> association, ArrayList<Set<Integer>> transactions){

    Set<Integer> left = association.get(0);
    Set<Integer> right = association.get(1);

    Set<Integer> all = new HashSet<>(left);
    all.addAll(right);
    Double support = getSupport(all, transactions);

    Double confidence = getConfidence(left, right, transactions);

    StringBuilder output = new StringBuilder("{");
    Iterator<Integer> leftIter = left.iterator();
    while(leftIter.hasNext()){
        output.append(leftIter.next().toString());
        if(leftIter.hasNext()){ output.append(","); }
```

```
    }
    output.append("}\t{");

    Iterator<Integer> rightIter = right.iterator();
    while(rightIter.hasNext()){
        output.append(rightIter.next().toString());
        if(rightIter.hasNext()){ output.append(","); }
    }
    output.append("}\t");
    output.append(String.format("%.2f", support)).append("\t");
    output.append(String.format("%.2f", confidence));

    return output.toString();
}
```

## printOutput()

1. Specification
   - Parameter: Set<ArrayList<Set<Integer>>> associations, ArrayList<Set<Integer>> transactions, String outputFile, Double minSup
   - Return type: void
2. Description

> 💡 BufferedWriter와 FileWriter를 통해 파라미터로 들어온 outputFile의 이름을 가진 파일을 만들고 각 줄을 쓴다. associations에서 각 association은 과제에 명시된 형식으로 변환 과정을 거친다.

3. Code

```
public static void printOutput(Set<ArrayList<Set<Integer>>> associations, ArrayList<Set<Integer>> transactions, String outputFile)

    try {
        BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

        for( ArrayList<Set<Integer>> association : associations ){
            String output = makeFormat(association, transactions);

            writer.write(output);
            writer.newLine();
        }
        writer.close();

    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
}
```

## getAssociations()

1. Specification
   - Parameter: Set<Set<Integer>> L
   - Return type: Set<ArrayList<Set<Integer>>>
2. Description

> 💡 최종 frequent item set(L)에 대해 멱집합(공집합 제외)을 구한다. 해당 subsets에 대해 가능한 모든 item_set(left), associative_item_set(right) 조합을 찾아 assocations를 구한다.

3. Code

```
public static Set<ArrayList<Set<Integer>>> getAssociations(Set<Set<Integer>> L){

    Set<ArrayList<Set<Integer>>> associations = new HashSet<>();

    for( Set<Integer> itemSet : L ){
        Set<Set<Integer>> powerSets = getPowerSet(itemSet);
```

```
        for( Set<Integer> powerSet : powerSets ){
            Set<Integer> nowPowerSetLeft = new HashSet<>(powerSet);
            Set<Integer> nowPowerSetRight = new HashSet<>(itemSet);
            nowPowerSetRight.removeAll(nowPowerSetLeft);

            ArrayList<Set<Integer>> nowAssociation = new ArrayList<>();
            nowAssociation.add(nowPowerSetLeft);
            nowAssociation.add(nowPowerSetRight);

            if(!associations.contains(nowAssociation)){ associations.add(nowAssociation); }
        }
    }
    return associations;
}
```

## getPowerSet()

1. Specification
   - Parameter: Set<Integer> itemSet
   - Return type: Set<Set<Integer>>
2. Description

> 비트 연산을 이용하여 item set에 대한 멱집합을 구한다. 1<<N에 대해 N=3이면 i는 000부터 111까지 증가하기 때문에 AND연산자(&)를 이용하여 가능한 모든 조합을 찾을 수 있다.

3. Code

```
public static Set<Set<Integer>> getPowerSet(Set<Integer> itemSet){

    Set<Set<Integer>> powerSet = new HashSet<>();

    ArrayList<Integer> itemSetArray = new ArrayList<>(itemSet);
    Integer setSize = itemSet.size();
    for(int i = 0 ; i < 1<<setSize ; i++){
        Set<Integer> nowItemSet = new HashSet<>();
        for(int j = 0 ; j < setSize ; j++){
            if((i & 1<<j) != 0){
                nowItemSet.add(itemSetArray.get(j));
            }
        }
        if(!nowItemSet.isEmpty() && !nowItemSet.equals(itemSet)){ powerSet.add(nowItemSet); }
    }
    return powerSet;
}
```

## getC0()

1. Specification
   - Parameter: Map<Set<Integer>, Double> L, Integer k, Set<Set<Integer>> notFrequent
   - Return type: Set<Set<Integer>>
2. Description

> frequent pattern인 L에 대해 item_set의 가능한 조합들을 찾는다. 이때 단계를 나타내는 k개의 개수를 가져야 하며, notFrequent 셋에 포함되어 있는 item을 가지지 않도록 한다.

3. Code

```
public static Set<Set<Integer>> getC0(Map<Set<Integer>, Double> L, Integer k, Set<Set<Integer>> notFrequent){

    Set<Set<Integer>> newC = new HashSet<>();

    Set<Set<Integer>> LKeySet = L.keySet();
    for( Set<Integer> LKey1 : LKeySet ){
        for( Set<Integer> LKey2 : LKeySet ){
            Set<Integer> newLKey = new HashSet<>(LKey1);
```

```
            newLKey.addAll(LKey2);

            if(newLKey.size() != k) continue;
            if(isNotFrequent(newLKey, notFrequent)) continue;

            if(!newC.contains(newLKey)){ newC.add(newLKey); }
        }
    }
    return newC;
}
```

## getC()

1. Specification
   - Parameter: Map<Set<Integer>, Double> L, Integer k, ArrayList<Set<Integer>> transactions, Set<Set<Integer>> notFrequent
   - Return type: Map<Set<Integer>, Double>
2. Description

   💡 frequent patter인 L에 대해 가능한 조합들을 찾고 그 item_set들의 support를 구한다.

3. Code

```
public static Map<Set<Integer>, Double> getC(Map<Set<Integer>, Double> L, Integer k, ArrayList<Set<Integer>> transactions, Set<Set

    Map<Set<Integer>, Double> newC = new HashMap<>();

    Set<Set<Integer>> C = getC0(L, k, notFrequent);
    for( Set<Integer> CKeySet : C ){
        Double support = getSupport(CKeySet, transactions);
        newC.put(CKeySet, support);
    }
    return newC;
}
```

## getL()

1. Specification
   - Parameter: Map<Set<Integer>, Double> C, Set<Set<Integer>> notFrequent, Double minSup
   - Return type: Map<Set<Integer>, Double>
2. Description

   💡 candidates인 C에 대해 support가 min support보다 작은 item_set들을 지우고 notFrequent에 추가하여 frequent item set을 구한다.

3. Code

```
public static Map<Set<Integer>, Double> getL(Map<Set<Integer>, Double> C, Set<Set<Integer>> notFrequent, Double minSup){

    Map<Set<Integer>, Double> L = new HashMap<>(C);

    Iterator<Set<Integer>> LIter = L.keySet().iterator();
    while(LIter.hasNext()){
        Set<Integer> nowItemSet = LIter.next();
        Double nowSupport = L.get(nowItemSet);

        if(nowSupport < minSup){
            LIter.remove();
            notFrequent.add(nowItemSet);
        }
    }
    return L;
}
```

## isNotFrequent()

1. Specification
   - Parameter: Set<Integer> itemSet, Set<Set<Integer>> notFrequent
   - Return type: boolean
2. Description

> 각 candidates를 생성하면서 수집한 not frequent item set을 활용하여 해당 item_set이 not frequent한지 확인한다.

3. Code

```
public static boolean isNotFrequent(Set<Integer> itemSet, Set<Set<Integer>> notFrequent){

    for( Set<Integer> notFrequentItemSet : notFrequent ){
        boolean isNotFrequent = true;
        for( Integer notFrequentItem : notFrequentItemSet ){
            if(!itemSet.contains(notFrequentItem)){ isNotFrequent = false; }
        }
        if(isNotFrequent) return true;
    }
    return false;
}
```

## getSupport()

1. Specification
   - Parameter: Set<Integer> itemSet, ArrayList<Set<Integer>> transactions
   - Return type: Double
2. Description

> transactions을 순회하며 전체 transactions 중 item_set이 포함되어 있는 transaction의 수의 비율을 구한다.

3. Code

```
public static Double getSupport(Set<Integer> itemSet, ArrayList<Set<Integer>> transactions){

    Double transactionSize = transactions.size() * 1.0;
    Double containedNum = 0.0;
    for( Set<Integer> transaction : transactions ){
        boolean isContained = true;
        for( Integer item : itemSet ){
            if(!transaction.contains(item)){ isContained = false; }
        }
        if(isContained){ containedNum++; }
    }
    return containedNum / transactionSize * 100;
}
```

## getConfidence()

1. Specification
   - Parameter: Set<Integer> left, Set<Integer> right, ArrayList<Set<Integer>> transactions
   - Return type: Double
2. Description

> transactions을 순회하며 각 transaction에 item_set(left)이 포함되어 있는지 확인하고, 있다면 associative_item_set(right)도 포함되어 있는지 확인한다. item_set이 포함되어 있는 transaction의 수에 대한 associative_item_set도 함께 포함되어 있는 transaction의 비율을 구한다.

3. Code

```
public static Double getConfidence(Set<Integer> left, Set<Integer> right, ArrayList<Set<Integer>> transactions){

    Double leftContainedNum = 0.0;
    Double allContainedNum = 0.0;
    for( Set<Integer> transaction : transactions ){
        boolean isLeftContained = true;
        for( Integer leftItem : left ){
            if(!transaction.contains(leftItem)){ isLeftContained = false; }
        }
        if(isLeftContained){
            leftContainedNum++;
            boolean isRightContained = true;
            for( Integer rightItem : right ){
                if(!transaction.contains(rightItem)){ isRightContained = false; }
            }
            if(isRightContained){ allContainedNum++; }
        }
    }
    return allContainedNum / leftContainedNum * 100;
}
```

# 3. How to Execute

## Environment

- OS: Mac OS Catalina 10.15.7
- Runtime: JDK 15.0.1
- IDE: IntelliJ Ultimate 2020.03

## Structure

execution - apriori.jar

　　ㄴ input.txt

project - src - cse - ds - Main.class

　　ㄴ out - artifacts - apriori.jar

## Usage

```
hyunjigonji@ihyeonjiui-MacBookPro assignment1 % cd execution
hyunjigonji@ihyeonjiui-MacBookPro execution % ls
apriori.jar     input.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % java -jar apriori.jar 5 input.txt output.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % ls
apriori.jar     input.txt       output.txt
hyunjigonji@ihyeonjiui-MacBookPro execution % cat output.txt
{16,19} {3}     5.60    47.46
{16,8,11}       {3}     6.40    78.05
{16,19} {8}     8.40    71.19
```

```
% java -jar apriori.jar {min_support} {input_file} {output_file}

ex) java -jar apriori.jar 5 input.txt output.txt
```

- execution 폴더에서 실행 가능