Assignment1

#Apriori algorithm

제출일	2023. 04. 03	**전공**	컴퓨터소프트웨어학부
과목	데이터 사이언스	**학번**	2019041703
담당교수	채동규 교수님	**이름**	김경민

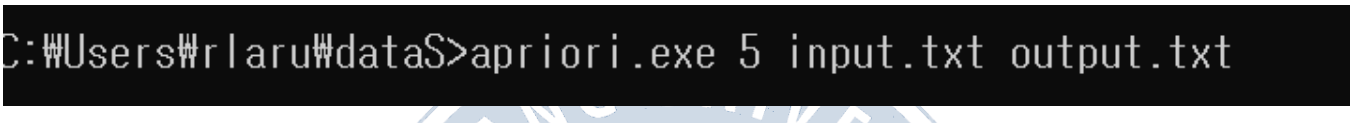
Environment

OS: Windows 10

Runtime: JDK 17.0.2

IDE: IntelliJ Ultimate 2021.03

Usage



apriori.exe {minSupport} {inputfile} {outputfile}

terminal 에서 apriori.exe 파일과 input.txt 파일이 있는 경로로 이동하여 위와 같은 명령어를 수행합니다.

Main idea

input.txt 에서 transactions 를 받아와서 List<Set<String>>에 저장한다

그리고 그 중에서 minimum Support 이상인 요소 쌍들을 Set<Set<String>>에 저장한다.

그 후 요소가 한 개 보다 많은 것을 골라 그것 자신을 뺀 서브셋에 대하여 support 와 confidence 를 구해준다.

1) getFrequentItemsets()

```java
public static Set<Set<String>> getFrequentItemsets(List<Set<String>> transactions, double minSupport) {

    Map<String, Integer> itemCounts = new HashMap<>();
    for (Set<String> transaction : transactions) {
        for (String item : transaction) {
            itemCounts.put(item, itemCounts.getOrDefault(item, defaultValue: 0) + 1);
        }
    }

    Set<Set<String>> frequentItemsets = new LinkedHashSet<>();

    Set<Set<String>> candidateItemsets = new HashSet<>();
    for (String item : itemCounts.keySet()) {
        Set<String> itemset = new HashSet<>(Arrays.asList(item));
        candidateItemsets.add(itemset);
    }

    int k = 1;
    while (!candidateItemsets.isEmpty()) {
        Map<Set<String>, Integer> candidateCounts = new HashMap<>();
        for (Set<String> candidate : candidateItemsets) {
            for (Set<String> transaction : transactions) {
                if (transaction.containsAll(candidate)) {
                    candidateCounts.put(candidate, candidateCounts.getOrDefault(candidate, defaultValue: 0) + 1);
                }
            }
        }

        for (Set<String> candidate : candidateCounts.keySet()) {
            double support = (double) candidateCounts.get(candidate) / transactions.size();
            if (support >= minSupport) {
                frequentItemsets.add(candidate);
            }
        }

        Set<Set<String>> nextCandidateItemsets = new HashSet<>();
        for (Set<String> itemset1 : frequentItemsets) {
            for (Set<String> itemset2 : frequentItemsets) {
                if (itemset1 != itemset2) {
                    Set<String> candidate = new HashSet<>(itemset1);
                    candidate.addAll(itemset2);
                    if (candidate.size() == k + 1) {
                        nextCandidateItemsets.add(candidate);
                    }
                }
            }
        }

        candidateItemsets = nextCandidateItemsets;
        k++;
    }
    return frequentItemsets;
```

모든 아이템의 개수를 세기 위해 Map<String, Integer> itemCounts 를 생성한다.

각 트랜잭션을 반복하면서, 각 아이템이 몇 번 등장했는지 itemCounts 에 저장한다.

후보 아이템셋을 저장하기 위해 Set<Set<String>> candidateItemsets 를 생성한다.

itemCounts 의 각 키를 기반으로, 후보 아이템셋을 생성하여 candidateItemsets 에 추가하고

while 루프를 실행한다.

각 후보 아이템셋이 포함된 트랜잭션의 개수를 세기 위해 Map<Set<String>, Integer> candidateCounts 를 생성한다.

모든 후보 아이템셋에 대해, 모든 트랜잭션을 반복하면서 후보 아이템셋이 포함된 트랜잭션의 개수를 candidateCounts 에 저장한다.

후보 아이템셋 중, minimumSupport 를 만족하는 아이템셋을 frequentItemsets 에 추가한다.

다음 후보 아이템셋을 생성하기 위해 Set<Set<String>> nextCandidateItemsets 를 생성한다.

frequentItemsets 를 반복하면서, 두 아이템셋을 결합하여 후보 아이템셋을 생성한다.

후보 아이템셋의 길이가 k+1 인 경우, nextCandidateItemsets 에 추가한다.

k 를 1 증가시키고, 다음 루프를 실행한다.

frequentItemsets 를 반환한다.

2) calculateSupport()

```java
public static double calculateSupport(Set<String> itemset, List<Set<String>> transactions) {
    double support = 0.0;

    for (Set<String> transaction : transactions) {
        if (transaction.containsAll(itemset)) {
            support += 1.0;
        }
    }

    return support / transactions.size();
}
```

support 값을 저장하기 위한 double 변수인 support 를 0.0 으로 초기화한다.

모든 트랜잭션을 반복하면서, 각 트랜잭션이 itemset 을 포함하는 경우 support 를 1.0 증가시킨다.

모든 트랜잭션의 수로 support 를 나누어, itemset 의 support 를 계산한다.

3) calculateConfidence

```java
public static double calculateConfidence(Set<String> Antecedent, Set<String> Consequent, List<Set<String>> transactions) {
    double jointSupport = 0.0;
    double antecedentSupport = 0.0;
    double confidence = 0.0;

    for (Set<String> transaction : transactions) {
        if (transaction.containsAll(Antecedent) && transaction.containsAll(Consequent)) {
            jointSupport += 1.0;
        }
    }

    for (Set<String> transaction : transactions) {
        if (transaction.containsAll(Antecedent)) {
            antecedentSupport += 1.0;
        }
    }

    if (antecedentSupport > 0) {
        confidence = jointSupport / antecedentSupport;
    }
    return confidence;
}
```

두 항목들이 함께 등장한 횟수를 저장하기 위한 double 변수인 jointSupport 를 0.0 으로 초기화한다.

Antecedent 와 Consequent 가 모두 포함된 트랜잭션의 수를 계산하여 jointSupport 를 증가시킨다.

Antecedent 가 포함된 트랜잭션의 수를 저장하기 위한 double 변수인 antecedentSupport 를 0.0 으로 초기화한다.

모든 트랜잭션을 반복하면서, 각 트랜잭션이 Antecedent 를 포함하는 경우 antecedentSupport 를 1.0 증가시킨다.

Antecedent 가 등장한 트랜잭션의 수가 0 보다 큰 경우 confidence 를 계산합니다.

4) getSubsets()

```java
public static Set<Set<String>> getSubsets(Set<String> set) {
    List<String> itemList = new ArrayList<>(set);
    Set<Set<String>> subsets = new HashSet<>();
    getSubsetsHelper(itemList, subsets, new HashSet<>(), index: 0);
    return subsets;
}

public static void getSubsetsHelper(List<String> itemList, Set<Set<String>> subsets, Set<String> currentSubset, int index) {
    if (index == itemList.size()) {
        if (!currentSubset.isEmpty()) {
            subsets.add(new HashSet<>(currentSubset));
        }
        return;
    }

    String item = itemList.get(index);

    getSubsetsHelper(itemList, subsets, currentSubset, index: index + 1);

    currentSubset.add(item);
    getSubsetsHelper(itemList, subsets, currentSubset, index: index + 1);
    currentSubset.remove(item);
}
```

1) getSubset()

입력된 set 을 List<String> itemList 으로 변환한다.

부분집합을 저장하기 위한 Set<Set<String>> subsets 을 생성한다.

getSubsetsHelper 메서드를 호출하여 부분집합을 생성한다.

2) getSubsetHelper()

현재 인덱스가 itemList 의 길이와 같은 경우, currentSubset 이 빈 집합이 아닌 경우에 subsets 에 추가하고 반환한다.

itemList 에서 현재 인덱스의 아이템을 가져온다.

currentSubset 을 그대로 사용하여, itemList 의 다음 인덱스부터의 모든 부분집합을 추출한다.

currentSubset 에 현재 아이템을 추가하고, itemList 의 다음 인덱스(index+1)부터의 모든 부분집합을 추출한다.

currentSubset 에서 현재 아이템(item)을 제거한다.