# Data Mining LAB : Experiment 6

## Submitted By:

Name: Debatreya Das  
Roll No. 12212070  
CS A4  
Data Mining LAB

## Part A

Objective: To compute maximal frequent itemset.

Compute candidate 3-itemsets from frequent 2-itemsets using join C3 = L2 x L2. (Han’s book example)

Generalize the algorithm for generating candidate Ci+1 itemsets from frequent Li itemsets Ci+1 = Li x Li

### Generating Candidate 3-itemsets (C3) from Frequent 2-itemsets (L2)

from itertools import combinations  
  
# Function to generate candidate 3-itemsets from frequent 2-itemsets  
def generate\_candidate\_3\_itemsets(L2):  
 C3 = set() # Store candidates in a set to avoid duplicates  
   
 # Join step: combine two frequent 2-itemsets to form a candidate 3-itemset  
 for itemset1 in L2:  
 for itemset2 in L2:  
 # Join if first two items match (i.e., {a, b} U {a, c} -> {a, b, c})  
 if len(itemset1.intersection(itemset2)) == 1:  
 candidate = itemset1.union(itemset2)  
 if len(candidate) == 3:  
 C3.add(frozenset(candidate)) # Frozenset to make itemsets hashable  
   
 return C3

### Generalizing for 𝐶𝑖+1 from 𝐿𝑖

def generate\_candidate\_itemsets(Li, k):  
 Ci\_plus\_1 = set()  
   
 # Join step: combine k-itemsets that differ by only one item  
 for itemset1 in Li:  
 for itemset2 in Li:  
 # Join if the first (k-1) items are the same  
 if len(itemset1.intersection(itemset2)) == k-1:  
 candidate = itemset1.union(itemset2)  
 if len(candidate) == k + 1:  
 Ci\_plus\_1.add(frozenset(candidate))  
   
 return Ci\_plus\_1

## Part B

Objective: To develop prune operation using apriory property.

Prune unnecessary 3-itemsets from the set of generated 3-itemsets C3 to make C3 to set of frequent 3-itemsets L3. (Han book example)

Generalize the algorithm for pruning unnecessary i-itemsets from the set of generated i-itemsets Ci to make Ci to set of frequent i-itemsets Li.

### 3 Itemset Prunning

def prune\_3\_itemsets(C3, L2):  
 pruned\_C3 = set()  
   
 # For each candidate 3-itemset  
 for candidate in C3:  
 valid = True  
 # Generate all 2-itemset subsets (since we're pruning 3-itemsets)  
 for subset in combinations(candidate, 2):  
 # If any 2-itemset subset is not in L2, prune the candidate  
 if frozenset(subset) not in L2:  
 valid = False  
 break  
 # If all 2-itemset subsets are frequent, keep the 3-itemset  
 if valid:  
 pruned\_C3.add(candidate)  
   
 return pruned\_C3

### Prunning infrequent itemset for Ci+1

def prune\_candidates(Ci\_plus\_1, Li):  
 pruned\_Ci\_plus\_1 = set()  
   
 for candidate in Ci\_plus\_1:  
 # Generate all k-sized subsets of the candidate  
 valid = True  
 for subset in combinations(candidate, len(candidate)-1):  
 if frozenset(subset) not in Li:  
 valid = False  
 break  
 if valid:  
 pruned\_Ci\_plus\_1.add(candidate)  
   
 return pruned\_Ci\_plus\_1

## Part C

Write Apriori algorithm using the above join and prune procedures.

from itertools import chain, combinations  
  
# Helper function to generate all candidate itemsets from a dataset  
def get\_itemsets\_from\_transactions(transactions, k):  
 itemsets = set()  
 for transaction in transactions:  
 for itemset in combinations(transaction, k):  
 itemsets.add(frozenset(itemset))  
 return itemsets  
  
# Helper function to calculate support of itemsets  
def calculate\_support(transactions, candidates):  
 support\_count = {itemset: 0 for itemset in candidates}  
 for transaction in transactions:  
 for candidate in candidates:  
 if candidate.issubset(transaction):  
 support\_count[candidate] += 1  
 return support\_count  
  
# Apriori algorithm  
def apriori(transactions, min\_support):  
 # Step 1: Generate frequent 1-itemsets (L1)  
 single\_items = chain.from\_iterable(transactions)  
 item\_count = {}  
 for item in single\_items:  
 item\_count[frozenset([item])] = item\_count.get(frozenset([item]), 0) + 1  
   
 # Filter 1-itemsets by min support  
 L1 = {itemset for itemset, count in item\_count.items() if count >= min\_support}  
 frequent\_itemsets = {1: L1}  
   
 k = 2  
 Li = L1  
 while Li:  
 # Step 2: Generate candidates Ci+1 from frequent Li itemsets  
 candidates = generate\_candidate\_itemsets(Li, k-1)  
   
 # Step 3: Calculate support for candidates  
 support\_count = calculate\_support(transactions, candidates)  
   
 # Step 4: Prune candidates whose support is less than min\_support  
 Li = {itemset for itemset, count in support\_count.items() if count >= min\_support}  
   
 if Li:  
 frequent\_itemsets[k] = Li  
 k += 1  
   
 return frequent\_itemsets  
  
# Example transactions (dataset)  
transactions = [  
 {1, 2, 3},  
 {1, 2, 4},  
 {2, 3, 4},  
 {1, 3, 4},  
 {1, 2, 3, 4}  
]  
  
# Minimum support threshold  
min\_support = 2  
  
# Run the Apriori algorithm  
frequent\_itemsets = apriori(transactions, min\_support)  
  
# Output the result  
for k, itemsets in frequent\_itemsets.items():  
 print(f"Frequent {k}-itemsets: {itemsets}")

Frequent 1-itemsets: {frozenset({3}), frozenset({2}), frozenset({1}), frozenset({4})}  
Frequent 2-itemsets: {frozenset({3, 4}), frozenset({1, 4}), frozenset({2, 3}), frozenset({1, 2}), frozenset({2, 4}), frozenset({1, 3})}  
Frequent 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 3, 4}), frozenset({1, 2, 4})}

## Tests

#### Example 2 Itemset

L2 = [frozenset([1, 2]), frozenset([1, 3]), frozenset([2, 3]), frozenset([2, 4])]

#### Generate candidate 3-itemsets (C3)

C3 = generate\_candidate\_3\_itemsets(L2)  
print("Candidate 3-itemsets:", C3)

Candidate 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 2, 4})}

#### Generalized candidate generation for k+1 from k

L3 = generate\_candidate\_itemsets(L2, 2)  
print("Generalized candidate 3-itemsets:", L3)

Generalized candidate 3-itemsets: {frozenset({1, 2, 3}), frozenset({2, 3, 4}), frozenset({1, 2, 4})}

#### Itemset Prunning

pruned\_C3 = prune\_3\_itemsets(C3, L2)  
print("Pruned 3-itemsets:", pruned\_C3)

Pruned 3-itemsets: {frozenset({1, 2, 3})}

#### Generalized Pruning the candidates

pruned\_L3 = prune\_candidates(L3, L2)  
print("Pruned 3-itemsets:", pruned\_L3)

Pruned 3-itemsets: {frozenset({1, 2, 3})}