**ECE 59500 IDM: PROGRAMMING ASSIGNMENT 2**

**REPORT**

**(sramagir@purdue.edu)**

1. **The frequent itemsets obtained on Gene dataset (L1, L2, L3):**

Length - 1 Frequent Itemsets: **(L1)**

Frequent Itemsets Count = 51

Frequent Itemsets:

[['gene\_1'], ['gene\_12'], ['gene\_14'], ['gene\_17'], ['gene\_21'], ['gene\_22'], ['gene\_23'], ['gene\_25'], ['gene\_26'], ['gene\_27'], ['gene\_3'], ['gene\_31'], ['gene\_36'], ['gene\_37'], ['gene\_39'], ['gene\_4'], ['gene\_43'], ['gene\_45'], ['gene\_47'], ['gene\_48'], ['gene\_5'], ['gene\_50'], ['gene\_53'], ['gene\_54'], ['gene\_55'], ['gene\_56'], ['gene\_59'], ['gene\_6'], ['gene\_60'], ['gene\_63'], ['gene\_64'], ['gene\_66'], ['gene\_67'], ['gene\_71'], ['gene\_72'], ['gene\_75'], ['gene\_77'], ['gene\_78'], ['gene\_8'], ['gene\_81'], ['gene\_83'], ['gene\_84'], ['gene\_87'], ['gene\_89'], ['gene\_9'], ['gene\_90'], ['gene\_91'], ['gene\_93'], ['gene\_94'], ['gene\_98'], ['gene\_99']]

Length - 2 Frequent Itemsets: **(L2)**

Frequent Itemsets Count = 29

Frequent Itemsets:

[['gene\_1', 'gene\_21'], ['gene\_1', 'gene\_3'], ['gene\_1', 'gene\_47'], ['gene\_1', 'gene\_5'], ['gene\_1', 'gene\_54'], ['gene\_1', 'gene\_59'], ['gene\_1', 'gene\_6'], ['gene\_1', 'gene\_67'], ['gene\_1', 'gene\_72'], ['gene\_1', 'gene\_8'], ['gene\_1', 'gene\_81'], ['gene\_1', 'gene\_84'], ['gene\_1', 'gene\_87'], ['gene\_1', 'gene\_89'], ['gene\_1', 'gene\_91'], ['gene\_1', 'gene\_94'], ['gene\_3', 'gene\_47'], ['gene\_3', 'gene\_5'], ['gene\_3', 'gene\_59'], ['gene\_3', 'gene\_72'], ['gene\_47', 'gene\_5'], ['gene\_5', 'gene\_59'], ['gene\_5', 'gene\_6'], ['gene\_5', 'gene\_72'], ['gene\_5', 'gene\_87'], ['gene\_5', 'gene\_91'], ['gene\_59', 'gene\_6'], ['gene\_59', 'gene\_72'], ['gene\_59', 'gene\_87']]

Length - 3 Frequent Itemsets: **(L3)**

Frequent Itemsets Count = 2

Frequent Itemsets:

[['gene\_1', 'gene\_3', 'gene\_5'], ['gene\_1', 'gene\_59', 'gene\_72']]

1. **The Length – 3 Candidate itemsets (C3) during Apriori Algorithm (Frequent Itemset Mining) on Gene dataset:**

Length - 3 Candidate Itemsets: **(C3)**

Candidate Itemsets Count = 20

Candidate Itemsets:

[frozenset({'gene\_3', 'gene\_47', 'gene\_1'}), frozenset({'gene\_3', 'gene\_5', 'gene\_1'}), frozenset({'gene\_3', 'gene\_59', 'gene\_1'}), frozenset({'gene\_3', 'gene\_72', 'gene\_1'}), frozenset({'gene\_5', 'gene\_47', 'gene\_1'}), frozenset({'gene\_59', 'gene\_5', 'gene\_1'}), frozenset({'gene\_6', 'gene\_5', 'gene\_1'}), frozenset({'gene\_72', 'gene\_5', 'gene\_1'}), frozenset({'gene\_5', 'gene\_87', 'gene\_1'}), frozenset({'gene\_5', 'gene\_91', 'gene\_1'}), frozenset({'gene\_6', 'gene\_59', 'gene\_1'}), frozenset({'gene\_72', 'gene\_59', 'gene\_1'}), frozenset({'gene\_59', 'gene\_87', 'gene\_1'}), frozenset({'gene\_3', 'gene\_5', 'gene\_47'}), frozenset({'gene\_3', 'gene\_59', 'gene\_5'}), frozenset({'gene\_3', 'gene\_72', 'gene\_5'}), frozenset({'gene\_3', 'gene\_72', 'gene\_59'}), frozenset({'gene\_59', 'gene\_6', 'gene\_5'}), frozenset({'gene\_59', 'gene\_72', 'gene\_5'}), frozenset({'gene\_59', 'gene\_5', 'gene\_87'})]

1. **The codes of Apriori algorithm implementation, i.e., the two functions (apriori\_gen and get\_freq).**
2. **get\_freq(dataset, candidates, min\_support, verbose=False):**

def get\_freq(dataset, candidates, min\_support, verbose=False):

"""

This function separates the candidates itemsets into frequent itemset and infrequent itemsets based on the min\_support, and returns all candidate itemsets that meet a minimum support threshold.

Parameters

----------

dataset : list

The dataset (a list of transactions) from which to generate candidate

candidates : frozenset

The list of candidate itemsets.

min\_support : float

The minimum support threshold.

Returns

-------

freq\_list : list

The list of frequent itemsets.

support\_data : dict

The support data for all candidate itemsets.

"""

support\_data = {} # Dictionary which has support counts for all Candidate Itemsets

freq\_list = [] # The list of frequent itemsets.

for c in candidates:

support\_data[c] = float(0) # Initializing all the Support Counts to 0

for transaction in dataset: # Counting the support of each candidate itemset

for c in candidates:

if c.issubset(transaction):

support\_data[c] = support\_data[c] + (1 / len(dataset));

for c in candidates:

if (support\_data[c] >= min\_support): # if the support of c is not less than minSupport, it is said to be "frequent"

c = list(c)

c.sort()

freq\_list.append(c)

if freq\_list:

if len(freq\_list[0])==1:

print("Length - 1 Frequent Itemsets: ")

print("\nFrequent Itemsets Count =", len(freq\_list))

print("\nFrequent Itemsets: \n", freq\_list)

print("\n------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------")

print()

return freq\_list, support\_data

1. **apriori\_gen(freq\_sets,k):**

**import itertools**

def apriori\_gen(freq\_sets, k):

"""Generates candidate itemsets (via the F\_k-1 x F\_k-1 method).

This part generates new candidate k-itemsets based on the frequent

(k-1)-itemsets found in the previous iteration.

The apriori\_gen function performs two operations:

(1) Generate length k candidate itemsets from length k-1 frequent itemsets

(2) Prune candidate itemsets containing subsets of length k-1 that are infrequent

Parameters

----------

freq\_sets : list

The list of frequent (k-1)-itemsets. (Lk-1)

k : integer

The cardinality of the current itemsets being evaluated.

Returns

-------

candidate\_list : list

The list of candidate itemsets.

"""

candidate\_list = [] # The list of candidate itemsets.

freq\_sets1 = freq\_sets # Copy of the freq\_sets: Lk-1 - list of frequent (k-1)-itemsets.

candidate\_list\_before\_pruning = []

# Candidate Set Generation:

# We generate a List of Candidate Sets, Ck by using two Frequent Itemsets from List (Lk-1) if their first k-2 items are identical.

# Lk-1 x Lk-1 method

for L in freq\_sets:

for L1 in freq\_sets1:

if (L != L1):

if L[:k-2] == L1[:k-2]:

C = frozenset(L + L1[k-2:]) # merging two itemsets in Lk-1 if their first k-2 items are identical

if C not in candidate\_list\_before\_pruning:

candidate\_list\_before\_pruning.append(C)

# Candidate Set Pruning:

# We remove an itemset C from Ck if any (k-1)-subset of this candidate itemset is not in the frequent itemset list Lk-1

for Ck in candidate\_list\_before\_pruning:

freq\_sets\_allsets= list(map(set, freq\_sets)) # List of all frequent itemsets

f = 0

Ck\_subsets = itertools.combinations(Ck, k-1) # Finding all (k-1)-subsets of Ck

for subset in Ck\_subsets:

subset = frozenset(subset)

if subset not in freq\_sets\_allsets:

f = 1

break

if f == 0:

candidate\_list.append(Ck)

print("Length -", k,"Candidate Itemsets: ")

print("\nCandidate Itemsets Count =", len(candidate\_list))

print("\nCandidate Itemsets: \n", candidate\_list)

print("\n------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------")

print()

print("Length -", k, "Frequent Itemsets: ")

return candidate\_list