Assignment2

October 18, 2023

1 ECE 59500 : Assignment 2: Association Analysis

[ ]:
```
from apriori_templete import *
```

[ ]:
```
data_path = "gene_data_transaction.txt"
min_support = 0.5
F, support = run_apriori(data_path, min_support, verbose=False)
```

1.1 The frequent itemsets on Gene dataset (L1, L2, L3)

[ ]:
```
print("Length of each frequent itemset")
for i in range(3):
    print(f"L{i+1} = {len(F[i])}")
```

```
Length of each frequent itemset
L1 = 51
L2 = 29
L3 = 2
```

1.1.1 L1

[ ]:
```
F[0]
```

[ ]:
```
[frozenset({'gene_1'}),
 frozenset({'gene_12'}),
 frozenset({'gene_17'}),
 frozenset({'gene_21'}),
 frozenset({'gene_22'}),
 frozenset({'gene_23'}),
 frozenset({'gene_3'}),
 frozenset({'gene_39'}),
 frozenset({'gene_4'}),
 frozenset({'gene_45'}),
 frozenset({'gene_48'}),
 frozenset({'gene_55'}),
 frozenset({'gene_59'}),
 frozenset({'gene_6'}),
 frozenset({'gene_63'}),
```

```
 frozenset({'gene_66'}),
 frozenset({'gene_72'}),
 frozenset({'gene_77'}),
 frozenset({'gene_83'}),
 frozenset({'gene_84'}),
 frozenset({'gene_93'}),
 frozenset({'gene_99'}),
 frozenset({'gene_14'}),
 frozenset({'gene_26'}),
 frozenset({'gene_27'}),
 frozenset({'gene_36'}),
 frozenset({'gene_37'}),
 frozenset({'gene_47'}),
 frozenset({'gene_5'}),
 frozenset({'gene_50'}),
 frozenset({'gene_54'}),
 frozenset({'gene_56'}),
 frozenset({'gene_60'}),
 frozenset({'gene_75'}),
 frozenset({'gene_78'}),
 frozenset({'gene_81'}),
 frozenset({'gene_87'}),
 frozenset({'gene_89'}),
 frozenset({'gene_25'}),
 frozenset({'gene_43'}),
 frozenset({'gene_53'}),
 frozenset({'gene_71'}),
 frozenset({'gene_8'}),
 frozenset({'gene_9'}),
 frozenset({'gene_90'}),
 frozenset({'gene_91'}),
 frozenset({'gene_98'}),
 frozenset({'gene_31'}),
 frozenset({'gene_94'}),
 frozenset({'gene_64'}),
 frozenset({'gene_67'})]
```

1.1.2 L2

[ ]:
```
F[1]
```

[ ]:
```
[frozenset({'gene_1', 'gene_84'}),
 frozenset({'gene_1', 'gene_6'}),
 frozenset({'gene_3', 'gene_72'}),
 frozenset({'gene_59', 'gene_6'}),
 frozenset({'gene_1', 'gene_72'}),
 frozenset({'gene_1', 'gene_3'}),
```

```
 frozenset({'gene_1', 'gene_59'}),
 frozenset({'gene_59', 'gene_72'}),
 frozenset({'gene_1', 'gene_21'}),
 frozenset({'gene_3', 'gene_59'}),
 frozenset({'gene_1', 'gene_5'}),
 frozenset({'gene_5', 'gene_72'}),
 frozenset({'gene_3', 'gene_47'}),
 frozenset({'gene_1', 'gene_54'}),
 frozenset({'gene_5', 'gene_59'}),
 frozenset({'gene_5', 'gene_87'}),
 frozenset({'gene_1', 'gene_89'}),
 frozenset({'gene_1', 'gene_47'}),
 frozenset({'gene_1', 'gene_81'}),
 frozenset({'gene_59', 'gene_87'}),
 frozenset({'gene_3', 'gene_5'}),
 frozenset({'gene_47', 'gene_5'}),
 frozenset({'gene_1', 'gene_87'}),
 frozenset({'gene_5', 'gene_6'}),
 frozenset({'gene_1', 'gene_8'}),
 frozenset({'gene_5', 'gene_91'}),
 frozenset({'gene_1', 'gene_91'}),
 frozenset({'gene_1', 'gene_67'}),
 frozenset({'gene_1', 'gene_94'})]
```

1.1.3 L3

[ ]:
```
F[2]
```

[ ]:
```
[frozenset({'gene_1', 'gene_59', 'gene_72'}),
 frozenset({'gene_1', 'gene_3', 'gene_5'})]
```

1.2 The length-3 candidate itemsets generated during Apriori (C3) on Gene dataset

[ ]:
```
dataset = loadDataSet(data_path)
_ , _ , candidates =  apriori(dataset, min_support=min_support,verbose=False ,␣
 ↪get_candidate=True)
```

[ ]:
```
candidates[2]
```

[ ]:
```
{frozenset({'gene_5', 'gene_59', 'gene_6'}),
 frozenset({'gene_3', 'gene_5', 'gene_59'}),
 frozenset({'gene_1', 'gene_47', 'gene_5'}),
 frozenset({'gene_1', 'gene_59', 'gene_6'}),
 frozenset({'gene_1', 'gene_5', 'gene_72'}),
 frozenset({'gene_1', 'gene_3', 'gene_59'}),
 frozenset({'gene_1', 'gene_5', 'gene_59'}),
```

```
frozenset({'gene_1', 'gene_3', 'gene_5'}),
frozenset({'gene_3', 'gene_59', 'gene_72'}),
frozenset({'gene_1', 'gene_59', 'gene_72'}),
frozenset({'gene_1', 'gene_5', 'gene_6'}),
frozenset({'gene_1', 'gene_5', 'gene_91'}),
frozenset({'gene_5', 'gene_59', 'gene_87'}),
frozenset({'gene_3', 'gene_5', 'gene_72'}),
frozenset({'gene_5', 'gene_59', 'gene_72'}),
frozenset({'gene_1', 'gene_3', 'gene_47'}),
frozenset({'gene_1', 'gene_5', 'gene_87'}),
frozenset({'gene_1', 'gene_59', 'gene_87'}),
frozenset({'gene_1', 'gene_3', 'gene_72'}),
frozenset({'gene_3', 'gene_47', 'gene_5'})}
```

1.3 Codes of the two functions: apriori_gen and get_freq.

[ ]:
```python
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
        itemsets.

    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.
    """

    # Count the support for each candidate itemset
    freq_data = {} #itemset -> freq
```

```
    total_transactions = len(dataset)
    for transaction in dataset:
        for candidate in candidates:
            if candidate.issubset(transaction):
                #candidate_str = " ".join(candidate)
                if candidate not in freq_data:
                    freq_data[candidate] = 1
                else:
                    freq_data[candidate] += 1
    freq_list = []
    support_data = {}
    for itemset, freq in freq_data.items():
        support_data[itemset] = freq/total_transactions
        if support_data[itemset]>=min_support:
            freq_list.append(itemset)

    if verbose:
        for itemset in freq_list:
            freq = freq_data[itemset]
            print(f"Itemset: {list(itemset)}, Support: {freq} /␣
 ↪{total_transactions} = {freq / total_transactions:.2f}")

    return freq_list, support_data

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.

    k : integer
        The cardinality of the current itemsets being evaluated.

    Returns
    -------
    candidate_list : list
        The list of candidate itemsets.
```

```
    """
    # TODO
    #print(freq_sets)
    candidateSet = set([i.union(j) for i in freq_sets for j in freq_sets if␣
↪len(i.union(j)) == k])
    #print(candidateSet)
    candidateSet_copy = candidateSet.copy()
    for item in candidateSet:
        subsets = combinations(item, k-1)
        #print(str(subsets))
        for subset in subsets:
            if(frozenset(subset) not in freq_sets):
                candidateSet_copy.remove(item)
                break

    return candidateSet_copy
```