CSE 601 : Data Mining and Bioinformatics

Project 1 Association Report

Rohan Hemanshu Sheth

Niranjan Deshpande

Apriori Algorithm and Association Rule Generation Algorithm:

Apriori Algorithm is a technique for frequent itemset mining and association rule learning. The frequent itemsets found by Apriori can be used to determine association rules that highlight certain trends in the data (Gene expressions in this case).

Frequent Itemset Generation implementation:

- First, we read the file in python in a dataframe, convert it into an array and process it by replacing each element by prefixing them with G1_, G2_,….,G100_ as per the respective column numbers. Then we extract unique elements in each column and add them to a list. This is the 1-length candidate itemset.
- Then we define function oneCset(a,sc) which takes the 1-length candidate itemset and minimum support value as arguments and returns 1-length frequent itemset.
- The function oneCset calculates the support count of each item in the 1-length candidate itemset by checking if the item is in any of the samples. It increments counter when it finds the item is in a data sample.
- Then every item that does not meet the minimum support requirement is eliminated from the list which forms the 1-length frequent itemset.
- Next, we define function jointSet(a,length,sc) that takes the k-length frequent itemset, integer length k+1, and minimum support value as arguments and returns k+1-length frequent itemset.
- For every two itemsets in the k-length frequent itemset, it first sorts the two itemsets and checks if the two items have the same first k-1 items. If true, the two itemsets are merged using set union and added to a list if the merged itemset is of length k+1. This forms the k+1-length candidate itemset.
- Then support count of every itemset in the list is calculated by checking if it is a subset of any of the samples and incrementing counter. The itemsets that do not satisfy the minimum support requirement are eliminated. This forms the k+1-length frequent itemset.
- The following code snippet generates the frequent itemsets until jointSet returns an empty set:

```
k=2
total=0
while(currentLSet!=[]):
    largeSet[k-1]=currentLSet
    total = total + len(currentLSet)
    currentLSet = jointSet(currentLSet,k,0.6)
#      print(currentLSet)
    k = k + 1
```

- largeSet is a dictionary that stores different length frequent itemsets with the keys as the length and total calculates the total number of frequent itemsets. Initially, currentLSet contains the 1-length frequent itemset returned by oneCset.

Rule Generation implementation:

- We import chain and combinations from the package itertools to generate non-empty subsets of an itemset in the function subsets(arr).
- Function getSupport(arr) calculates and returns the support count of a list of items and function printRules(rulelist) prints the rules with their confidence value.
- Function ruleGen(minconf) iterates over the largeSet dictionary starting with key value 2 and for every item in each value, it generates all non-empty subsets of the item, and for every subset as the HEAD of a rule, it calculates BODY by subtracting the subset from the item and calculates the confidence of this potential rule. If the rule does not satisfy the minimum confidence requirement, it is ignored, else it is added to a list along with its confidence. This list of rules is returned.
- The codes for the three templates are written accordingly.

TASK 1 results in terms of number of k-length frequent itemsets:

For Support = 30%

Number of 1-length frequent itemsets: 196
Number of 2-length frequent itemsets: 5340
Number of 3-length frequent itemsets: 5287
Number of 4-length frequent itemsets: 1518
Number of 5-length frequent itemsets: 438
Number of 6-length frequent itemsets: 88
Number of 7-length frequent itemsets: 11
Number of 8-length frequent itemsets: 1
Number of 9-length frequent itemsets: 0
Total number of frequent itemsets: 12879

For Support = 40%

Number of 1-length frequent itemsets: 167
Number of 2-length frequent itemsets: 753
Number of 3-length frequent itemsets: 149
Number of 4-length frequent itemsets: 7
Number of 5-length frequent itemsets: 1
Number of 6-length frequent itemsets: 0
Total number of frequent itemsets: 1077

For Support = 50%

Number of 1-length frequent itemsets: 109
Number of 2-length frequent itemsets: 63
Number of 3-length frequent itemsets: 2
Number of 4-length frequent itemsets: 0
Total number of frequent itemsets: 174

For Support = 60%

Number of 1-length frequent itemsets: 34
Number of 2-length frequent itemsets: 2
Number of 3-length frequent itemsets: 0
Total number of frequent itemsets: 36

For Support = 70%

Number of 1-length frequent itemsets: 7
Number of 2-length frequent itemsets: 0
Total number of frequent itemsets: 7

TASK 2 results in terms of number of generated rules for Support = 50% and Confidence = 70%:

Total number of rules: 117

```
(result11, cnt) = asso_rule.template1("RULE", "ANY",
['G59_UP'])
Result: 26

(result12, cnt) = asso_rule.template1("RULE", "NONE",
['G59_UP'])
Result: 91

(result13, cnt) = asso_rule.template1("RULE", 1,
['G59_UP', 'G10_Down'])
Result: 39

(result14, cnt) = asso_rule.template1("HEAD", "ANY",
['G59_UP'])
Result: 9

(result15, cnt) = asso_rule.template1("HEAD", "NONE",
['G59_UP'])
Result: 108

(result16, cnt) = asso_rule.template1("HEAD", 1,
['G59_UP', 'G10_Down'])
Result: 17

(result17, cnt) = asso_rule.template1("BODY", "ANY",
['G59_UP'])
```

```
Result: 17

(result18, cnt) = asso_rule.template1("BODY", "NONE",
['G59_UP'])
Result: 100

(result19, cnt) = asso_rule.template1("BODY", 1,
['G59_UP', 'G10_Down'])
Result: 24

(result21, cnt) = asso_rule.template2("RULE", 3)
Result: 9

(result22, cnt) = asso_rule.template2("HEAD", 2)
Result: 6

(result23, cnt) = asso_rule.template2("BODY", 1)
Result: 117
(result31, cnt) = asso_rule.template3("1or1", "HEAD",
"ANY", ['G10_Down'], "BODY", 1, ['G59_UP'])
Result: 24

(result32, cnt) = asso_rule.template3("1and1", "HEAD",
"ANY", ['G10_Down'], "BODY", 1, ['G59_UP'])
Result: 1

(result33, cnt) = asso_rule.template3("1or2", "HEAD",
"ANY", ['G10_Down'], "BODY", 2)
Result: 11
```

```
(result34, cnt) = asso_rule.template3("1and2", "HEAD",
"ANY", ['G10_Down'], "BODY", 2)
Result: 0

(result35, cnt) = asso_rule.template3("2or2", "HEAD",
1, "BODY", 2)
Result: 117

(result36, cnt) = asso_rule.template3("2and2", "HEAD",
1, "BODY", 2)
Result: 3
```