Association report

Team members

Mitali Vijay Bhiwande
Sumedh Sadanand Ambokar
Tejasvi Balaram Sankhe

Algorithm Details:

Apriori Algorithm:

Apriori is designed to operate on databases containing transactions to generate frequent itemsets. Apriori uses the property that states, any subset of a frequent itemset must also be a frequent itemset thus for an infrequent itemset, all its supersets must also be infrequent and using this property non-frequent itemsets are pruned at each step.

Implementation of Apriori Algorithm:

The Code first reads the text file and converts each record to a set and the complete file as list of sets where each set corresponds to a single patient.

For generating the frequent itemsets, support percent and list of sets containing all the records should be passed to the calculate_frequent_set_count() function. It generates and prunes frequent itemsets of different sizes by calling generate_sets() function until number of frequent item-sets obtained is 0. After every iteration the size of the frequent item-sets to be computed is increased by 1.

The generate_sets() function is used for generating candidate sets from the datasets. For frequent sets of length 1, pruning is achieved by deleting the entries of sets from dictionary data structure that do not satisfy the minimum support requirement calculated by get_support_count() function. The dictionary is created by reading through the provided data set and maintaining count of individual genes. For frequent sets of length greater than 2, sets with length k that have same k-1 values are combined to form sets with length k+1. These sets are added to list of frequent sets if they satisfy the minimum support condition.

The get_support_count() function calculates the support count when provided with an input set. It checks the number of times the input set is present in the complete data and returns that as support count.

For rules generation, the function generate_rules() takes frequent datasets, confidence percentage and dictionary with support count of all frequent data sets as input. This function generates all combinations of association rules from frequent sets with size greater than 1 and adds them to a list of rules which satisfy the confidence requirement checked by validate_rules() function.

For template parsing, 3 methods are used for 3 different templates. They return a list of association rules that satisfy the given query and the total count of those rules. print_assoc_rule() function is used to display the rules and count returned by template parsing function.

Part 1:

1. **Support is set to be 30%**
   Number of length-1 frequent itemsets: 196
   Number of length-2 frequent itemsets: 5340
   Number of length-3 frequent itemsets: 5287
   Number of length-4 frequent itemsets: 1518
   Number of length-5 frequent itemsets: 438
   Number of length-6 frequent itemsets: 88
   Number of length-7 frequent itemsets: 11
   Number of length-8 frequent itemsets: 1

2. **Support is set to be 40%**
   Number of length-1 frequent itemsets: 167
   Number of length-2 frequent itemsets: 753
   Number of length-3 frequent itemsets: 149
   Number of length-4 frequent itemsets: 7
   Number of length-5 frequent itemsets: 1

3. **Support is set to be 50%**
   Number of length-1 frequent itemsets: 109
   Number of length-2 frequent itemsets: 63
   Number of length-3 frequent itemsets: 2

4. **Support is set to be 60%**
   Number of length-1 frequent itemsets: 34
   Number of length-2 frequent itemsets: 2

5. **Support is set to be 70%**
   Number of length-1 frequent itemsets: 7

Part 2:

1. **Template 1 queries:**

   i. (result11, cnt) = asso_rule.template1("RULE", "ANY", ['G59_Up'])

   ```
   Count -->  26
   ```

   ii. (result12, cnt) = asso_rule.template1("RULE", "NONE", ['G59_Up'])

   ```
   Count -->  91
   ```

   iii. (result13, cnt) = asso_rule.template1("RULE", 1, ['G59_Up', 'G10_Down'])

   ```
   Count -->  39
   ```

   iv. (result14, cnt) = asso_rule.template1("BODY", "ANY", ['G59_Up'])

   ```
   Count -->  9
   ```

   v. (result15, cnt) = asso_rule.template1("BODY", "NONE", ['G59_Up'])

   ```
   Count -->  108
   ```

   vi. (result16, cnt) = asso_rule.template1("BODY", 1, ['G59_Up', 'G10_Down'])

   ```
   Count -->  17
   ```

   vii. (result17, cnt) = asso_rule.template1("HEAD", "ANY", ['G59_Up'])

   ```
   Count -->  17
   ```

viii. (result18, cnt) = asso_rule.template1("HEAD", "NONE", ['G59_Up'])

```
Count -->  100
```

ix. (result19, cnt) = asso_rule.template1("HEAD", 1, ['G59_Up', 'G10_Down'])

```
Count -->  24
```

2. <u>Template 2 queries:</u>

i. (result21, cnt) = asso_rule.template2("RULE", 3)

```
Count -->  9
```

ii. (result22, cnt) = asso_rule.template2("BODY", 2)

```
Count -->  6
```

iii. (result23, cnt) = asso_rule.template2("HEAD", 1)

```
Count -->  117
```

3. <u>Template 3 queries:</u>

i. (result31, cnt) = asso_rule.template3("1or1", "BODY", "ANY", ['G10_Down'], "HEAD", 1, ['G59_Up'])

```
Count -->  24
```

ii. (result32, cnt) = asso_rule.template3("1and1", "BODY", "ANY", ['G10_Down'], "HEAD", 1, ['G59_Up'])

```
Count -->  1
```

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

```
Count -->  11
```

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

```
Count -->  0
```

v. (result35, cnt) = asso_rule.template3("2or2", "BODY", 1, "HEAD", 2)

```
Count -->  117
```

vi. (result36, cnt) = asso_rule.template3("2and2", "BODY", 1, "HEAD", 2)

```
Count -->  3
```