CSE 601: Data Mining and Bioinformatics

Project 1: Dimensionality Reduction & Association Analysis

Part 2: Association Analysis

Team Members:

1. **Name:** Bhagutharivalan Natarajan Muthukkannu
   UB Person No: 50314871
   UBIT Name: bhagutha
2. **Name:** Harish Kannan Venkataramanan
   UB Person No: 50316999
   UBIT Name: hvenkata
3. **Name:** Praveen Mohan
   UB Person No: 50321225
   UBIT Name: pmohan2

Generating Frequent Itemset using Apriori Algorithm

Packages used:

1. *"import itertools"* - To iterate through each frequent itemsets to generate candoate rules.
2. *"import pandas as pd"* - To store and retrieve RULES | HEAD | BODY for obtaining template results.

Functions implemented:

- ***freq_items_generation(candidate_items, tr_list, support, length)***

```
def freq_items_generation(candidate_items, tr_list, support, length):
    frequent_dict = dict()
    for candidate in candidate_items:
        if length != 1:
            candidate_set = set(candidate.split(','))
        else:
            candidate_set = {candidate}
        count = 0
        for row in tr_list:
            if(candidate_set.issubset(row)):
                count+=1
        if (count/len(tr_list))*100 >= support:
            frequent_dict[candidate] = count
    return list(frequent_dict.keys())
```

The function freq_items_generation(candidate_items, tr_list, support, length) generates a list of frequent itemset.

Inputs:

1. candidate_items - A set of candidate frequent itemset for generating frequent itemsets for the corresponding threshold.
2. tr_list - Transaction list or list of all rows from the original data to find the support count for candidate frequent itemset.
3. support - Minimum Support Threshold in percentage.
4. length - Length of the candidate frequent itemset

Output:

1. list(frequent_dict.keys()) - A list of frequent itemset.

- ***candidate_set_generation(items, length)***

```
def candidate_set_generation(items, length):
    candidate_items = set()
    for a in range(len(items)-1):
        items_a = set(items[a].split(','))
        for b in range(a+1, len(items)):
            items_b = set(items[b].split(','))
            if(len(items_a | items_b)==length):
                temp = sorted(items_a | items_b)
                temp_list = ','.join(temp)
                candidate_items.add(temp_list)
    return candidate_items
```

The function candidate_set_generation(items, length) generates a set of candidate frequent itemset.

Inputs:

1. items - A set of frequent itemset for generating next length candidate frequent itemset.
2. length - Length of the itemset.

Output:

1. candidate_items - A set of candidate frequent itemset.

- ***open_file(filename)***

```
def open_file(filename):
    file = open(filename, "r")
    candidate_items_l1 = set()
    tr_list = []
    for line in file:
        row = line.strip("\n").split("\t")
        for i in range(len(row)):
            if i != len(row)-1:
                row[i] = "G"+str(i+1)+"_"+row[i].upper()
            candidate_items_l1.add(row[i])
        tr_list.append(set(row))
    return(candidate_items_l1, tr_list)
```

The function open_file(filename) generates a list of candidate frequent itemset of length 1 and a transaction list or list of all rows from the original data to find the support count for candidate frequent itemset.

Inputs:

1. filename - Name of the original gene data file.

Output:

1. candidate_items_l1 - A list of candidate frequent itemset of length 1.
2. tr_list - Transaction list or list of all rows from the original data to find the support count for candidate frequent itemset.

- ***def apriori_imp_template(filename, support)***

```
def apriori_imp_template(filename, support):
    candidate_items_l1, tr_list = open_file(filename)
    length = 1
    candidate_items = candidate_items_l1
    freq_items = freq_items_generation(candidate_items, tr_list, support, length)
    ans = [len(freq_items)]
    while True:
        length += 1
        candidate_items = candidate_set_generation(freq_items, length)
        freq_items = freq_items_generation(candidate_items, tr_list, support, length)
        if len(freq_items) == 0:
            break
        else:
            ans.append(len(freq_items))
    print("Support is set to be "+ str(support)+"%")
    for i in range(len(ans)):
        print("number of length-"+str(i+1)+ " frequent itemsets: "+str(ans[i]))
    print("number of all lengths frequent itemsets: "+str(sum(ans))+"\n\n")
```

The function apriori_imp_template(filename, support) generates the template for part 1 of Apriori Algorithm in generating the frequent itemset for the given support.

Inputs:

1. filename - Name of the original gene data file.
2. support - Minimum Support Threshold in percentage.

Output:

1. The template for part 1 of Apriori Algorithm in generating the frequent itemsets for the given support.

- ***apriori_imp_result(filename, support)***

```
def apriori_imp_result(filename, support):
    candidate_items_l1, tr_list = open_file(filename)
    length = 1
    candidate_items = candidate_items_l1
    freq_items = freq_items_generation(candidate_items, tr_list, support, length)
    result = set(freq_items)
    while True:
        length += 1
        candidate_items = candidate_set_generation(freq_items, length)
        freq_items = freq_items_generation(candidate_items, tr_list, support, length)
        if len(freq_items) == 0:
            break
        else:
            result = result | set(freq_items)
    return result, tr_list
```

Results for given support values [30%, 40%, 50%, 60%, 70%]

```
support_values = [30, 40, 50, 60, 70]
for val in support_values:
    apriori_imp_template("association-rule-test-data.txt", val)
```

```
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
number of all lengths frequent itemsets: 12879


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
number of all lengths frequent itemsets: 1077


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
number of all lengths frequent itemsets: 174


Support is set to be 60%
number of length-1 frequent itemsets: 34
number of length-2 frequent itemsets: 2
number of all lengths frequent itemsets: 36


Support is set to be 70%
number of length-1 frequent itemsets: 7
number of all lengths frequent itemsets: 7
```

Generating Association Rules from frequent itemset

- ***freq_count(freq_itemset, tr_list)***

```
def freq_count(freq_itemset, tr_list):
    freq_itemset = set(freq_itemset)
    count = 0
    for row in tr_list:
        if(freq_itemset.issubset(row)):
            count+=1
    return count
```

The function freq_count(freq_itemset, tr_list) generates the count for the given itemset in the transaction database.

Inputs:

1. freq_itemset - A frequent itemset for the given support threshold.
2. tr_list - Transaction list or list of all rows from the original data to find the support count for candidate frequent itemset.

Output:

1. count = The count for the given frequent itemset in the tr_list.

- ***association_rules(filename, support, confidence)***

```python
def association_rules(filename, support, confidence):
    import itertools
    import pandas as pd
    rules = set()
    rules_list = []
    result, tr_list = apriori_imp_result(filename, support)
    result = list(result)
    for i in range(len(result)):
        result[i] = result[i].split(",")
    result = list(sorted(result, key = len, reverse=True))
    for freq_itemset in result:
        if len(freq_itemset) != 1:
            rule_cnt = freq_count(freq_itemset, tr_list)
            for i in range(len(freq_itemset) - 1 , 0, -1):
                head_list = list(itertools.combinations(freq_itemset, i))
                for head in head_list:
                    head_cnt = freq_count(head, tr_list)
                    rule_confidence = round((rule_cnt/head_cnt)*100, 2)################
                    if rule_confidence >= confidence:
                        freq_itemset = set(freq_itemset)
                        head = set(head)
                        body = head ^ freq_itemset
                        rule_set = set(head|body)
                        rule = ",".join(head) + " " + "->" + " " + ",".join(body)
                        if rule not in rules:
                            rules.add(rule)
                            rules_list.append([rule, rule_set, head, body, rule_confidence])
    rules_df = pd.DataFrame(rules_list, columns= ["RULE_SET", "RULE", "HEAD", "BODY", "Confidence"])
    return rules_df
```

Generating Association Rules with a minimum Support Threshold of 50% and Confidence Threshold of 70%.

```python
rules = association_rules("association-rule-test-data.txt", 50, 70)
rules
```

	RULE_SET	RULE	HEAD	BODY	Confidence
0	G72_UP,G59_UP -> G82_DOWN	{G72_UP, G82_DOWN, G59_UP}	{G72_UP, G59_UP}	{G82_DOWN}	83.87
1	G82_DOWN,G59_UP -> G72_UP	{G72_UP, G82_DOWN, G59_UP}	{G82_DOWN, G59_UP}	{G72_UP}	91.23
2	G82_DOWN,G72_UP -> G59_UP	{G82_DOWN, G59_UP, G72_UP}	{G82_DOWN, G72_UP}	{G59_UP}	89.66
3	G72_UP -> G82_DOWN,G59_UP	{G82_DOWN, G59_UP, G72_UP}	{G72_UP}	{G82_DOWN, G59_UP}	70.27
4	G82_DOWN -> G72_UP,G59_UP	{G72_UP, G59_UP, G82_DOWN}	{G82_DOWN}	{G72_UP, G59_UP}	76.47
...	...	...	...	...	...
112	G88_DOWN -> G24_DOWN	{G88_DOWN, G24_DOWN}	{G88_DOWN}	{G24_DOWN}	70.42
113	G59_UP -> G88_DOWN	{G88_DOWN, G59_UP}	{G59_UP}	{G88_DOWN}	72.37
114	G88_DOWN -> G59_UP	{G88_DOWN, G59_UP}	{G88_DOWN}	{G59_UP}	77.46
115	G88_DOWN -> G38_DOWN	{G88_DOWN, G38_DOWN}	{G88_DOWN}	{G38_DOWN}	70.42
116	G2_DOWN -> G38_DOWN	{G38_DOWN, G2_DOWN}	{G2_DOWN}	{G38_DOWN}	75.76

117 rows × 5 columns

- ***asso_rule_template1(a, b, c)***

```python
def asso_rule_template1(a, b, c):
    c = set(",".join(c).upper().split(','))
    result = []
    for i in range(len(rules)):
        if b == 'ANY' and len(c & rules.iloc[i][a]) > 0:
            result.append(rules.iloc[i]["RULE_SET"])
        elif b == 'NONE' and len(c & rules.iloc[i][a]) == 0:
            result.append(rules.iloc[i]["RULE_SET"])
        elif b == 1 and len(c & rules.iloc[i][a]) == 1:
            result.append(rules.iloc[i]["RULE_SET"])
    return result, len(result)
```

The function asso_rule_template1(a, b, c) generates the results for template 1 for the given query.

Inputs:

1. a - "RULE" | "HEAD" | "BODY"
2. b - "ANY" | "NONE" | 1
3. c - ["Gene", ...]

Output:

1. result - A list of rules for the given query.
2. len(result) - Total number of rules generated for the given query.

Template 1 results
(support = 50%, confidence = 70%)

```python
template1_query = [["RULE", "ANY", ['G59_UP']], ["RULE", "NONE", ['G59_UP']], ["RULE", 1, ['G59_UP', 'G10_
for val in template1_query:
    result, count = asso_rule_template1(val[0], val[1], val[2])
    print("The total number of rules generated for the template 1 query "+"'"+val[0]+","+" "+str(val[1])+"
```

```
The total number of rules generated for the template 1 query 'RULE, ANY, ['G59_UP']': 26
The total number of rules generated for the template 1 query 'RULE, NONE, ['G59_UP']': 91
The total number of rules generated for the template 1 query 'RULE, 1, ['G59_UP', 'G10_Down']': 39
The total number of rules generated for the template 1 query 'HEAD, ANY, ['G59_UP']': 9
The total number of rules generated for the template 1 query 'HEAD, NONE, ['G59_UP']': 108
The total number of rules generated for the template 1 query 'HEAD, 1, ['G59_UP', 'G10_Down']': 17
The total number of rules generated for the template 1 query 'BODY, ANY, ['G59_UP']': 17
The total number of rules generated for the template 1 query 'BODY, NONE, ['G59_UP']': 100
The total number of rules generated for the template 1 query 'BODY, 1, ['G59_UP', 'G10_Down']': 24
```

- ***asso_rule_template2(a, b)***

```
def asso_rule_template2(a, b):
    result = []
    count = 0
    for i in range(len(rules)):
        if len(rules.iloc[i][a]) >= b:
            result.append(rules.iloc[i]["RULE_SET"])
            count += 1
    return result, count
```

The function asso_rule_template2(a, b) generates the results for template 2 for the given query.

Inputs:

1. a - "RULE" | "HEAD" | "BODY"
2. b - integer (length)

Output:

1. result - A list of rules for the given query.
2. count - Total number of rules generated for the given query.

Template 2 results

```
template2_query = [["RULE", 3], ["HEAD", 2], ["BODY", 1]]
for val in template2_query:
    result, count = asso_rule_template2(val[0], val[1])
    print("The total number of rules generated for the template 2 query "+"'"+val[0]+","+" "+str(val[1])+"
```

```
The total number of rules generated for the template 2 query 'RULE, 3': 9
The total number of rules generated for the template 2 query 'HEAD, 2': 6
The total number of rules generated for the template 2 query 'BODY, 1': 117
```

- ***temp_operator(string)***

```
def temp_operator(string):
    if len(string) == 4:
        string = string.split("or")
        string.append("or")
    elif len(string) == 5:
        string = string.split("and")
        string.append("and")
    return string
```

The function temp_operator(string) splits the first input for template 3 into respective template value and the corresponding operator.

Inputs:

1. string - "1or1", "2or2", "1or2", "1and1", "2and2", "1and2".

Output:

1. list - [template number, template number, operator]

- ***asso_rule_template3(a,b,c,d,e,f=None,g=None)***

```python
def asso_rule_template3(a,b,c,d,e,f=None,g=None):
    a = temp_operator(a)
    if a[0] == '1' and a[1] == '1':
        result1, count1 = asso_rule_template1(b, c, d)
        result2, count2 = asso_rule_template1(e, f, g)
    elif a[0] == '2' and a[1] == '2':
        result1, count1 = asso_rule_template2(b, c)
        result2, count2 = asso_rule_template2(d, e)
    elif a[0] == '1' and a[1] == '2':
        result1, count1 = asso_rule_template1(b, c, d)
        result2, count2 = asso_rule_template2(e, f)
    elif a[0] == '2' and a[1] == '1':
        result1, count1 = asso_rule_template2(b, c)
        result2, count2 = asso_rule_template1(d, e, f)

    if a[2] == "and":
        final = set(result1) & set(result2)
        final = set(final)
        count = len(final)
        return final, count
    if a[2] == "or":
        final = result1 + result2
        final = set(final)
        count = len(final)
        return final, count
```

The function asso_rule_template2(a, b) generates the results for template 3 for the given query.

Inputs:

1. a - "1or1", "2or2", "1or2", "1and1", "2and2", "1and2".
2. b - "RULE" | "HEAD" | "BODY"
3. c - "ANY" | "NONE" | 1 (or) integer
4. d - ["Gene", ...] (or) "RULE" | "HEAD" | "BODY"
5. e - "RULE" | "HEAD" | "BODY" (or) "ANY" | "NONE" | 1
6. f - "ANY" | "NONE" | 1 (or) ["Gene", ...] (or) None
7. g - ["Gene", ...] (or) None

Output:

1. final - A set of rules for the given query.
2. count - Total number of rules generated for the given query.

Template 3 results

```python
template3_query = [["1or1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_UP']], ["1and1", "HEAD", "ANY",
for val in template3_query:
    if len(val) == 7:
        result, count = asso_rule_template3(val[0], val[1], val[2], val[3], val[4], val[5], val[6])
        print("The total number of rules generated for the template 3 query "+str(val)+": "+ str(count)+"\
    elif len(val) == 6:
        result, count = asso_rule_template3(val[0], val[1], val[2], val[3], val[4], val[5])
        print("The total number of rules generated for the template 3 query "+str(val)+": "+ str(count)+"\
    elif len(val) == 5:
        result, count = asso_rule_template3(val[0], val[1], val[2], val[3], val[4])
        print("The total number of rules generated for the template 3 query "+str(val)+": "+ str(count)+"\


```

```
The total number of rules generated for the template 3 query ['1or1', 'HEAD', 'ANY', ['G10_Down'], 'BODY', 1,
['G59_UP']]: 24

The total number of rules generated for the template 3 query ['1and1', 'HEAD', 'ANY', ['G10_Down'], 'BODY', 1
, ['G59_UP']]: 1

The total number of rules generated for the template 3 query ['1or2', 'HEAD', 'ANY', ['G10_Down'], 'BODY', 2]
: 11

The total number of rules generated for the template 3 query ['1and2', 'HEAD', 'ANY', ['G10_Down'], 'BODY', 2
]: 0

The total number of rules generated for the template 3 query ['2or2', 'HEAD', 1, 'BODY', 2]: 117

The total number of rules generated for the template 3 query ['2and2', 'HEAD', 1, 'BODY', 2]: 3
```