Apriori

- Programming assignment #1

2013011060
양승호

1. Environment

Ubuntu Linux 18.04
Python 3.6.7

2. How to run

$ python3 apriori.py [support] [input_file_name] [output_file_name]

3. Algorithm Summary

1. Find frequent 1-itemsets.
2. Generate combinations from the set.
3. Check whether the each combination satisfies the support and abandon the sets which don't satisfy.
4. Print the remaining results and iterate from [Procedure 2] until there is no candidates happen.

4. Details

```
 5 # Return a frequent 1-itemset
 6 def make_first_freq(dbs: list, sup: int) -> 'dict':
 7     freq = dict()
 8     for tx in dbs:
 9         for i in tx:
10             if i in freq:
11                 freq[i] += 1
12             else:
13                 freq[i] = 1
14     for j in freq:
15         if freq[j] < sup:
16             del freq[j]
17     return freq
```

- Return a frequent 1-itemset from the dbs(input file)

```
19 # Return whether all the items in tuple are in a transcatior
20 def tuple_in_db(tup: tuple, db: list) -> 'Bool':
21     for i in range(len(tup)):
22         if list(tup)[i] not in db:
23             return False
24     return True
```

- check whether the each combination is in the db(transacton of input file)

```
26 # Return whether all the subsets of big tuple are in a small tuple
27 def subset_satisfy(tup: tuple, freq: dict, length: int) -> 'Bool':
28     if length == 1:
29         for comb in itertools.combinations(list(tup), length):
30             if comb[0] not in freq:
31                 return False
32         return True
33     else:
34         for comb in itertools.combinations(list(tup), length):
35             if comb not in freq:
36                 return False
37         return True
```

- check whether the subsets of each candidates are in frequent itemsets

```
81 # Return the values as a suggested form
82 def make_output_format(tu_1: tuple, tu_2: tuple, val_1: float, val_2:
   float) -> 'str':
83     ret = '{'
84     for item in tu_1:
85         ret += str(item)
86         if tu_1.index(item) != len(tu_1)-1:
87             ret += ','
88     ret += '}\t{'
89     for item in tu_2:
90         ret += str(item)
91         if tu_2.index(item) != len(tu_2)-1:
92             ret += ','
93     ret += '}\t'+ '{}\t'.format(round(val_1, 2)) + '{}\n'.format(round
   (val_2, 2))
94     print(ret)
95     return ret
```

- return an output form using each itemsets, support and confidence

```
 97 # Print the results to an output file
 98 def make_output(dbs: list, freqs: list, output_file_name: str):
 99     output = open(output_file_name, 'w')
100     for output_len in range(1, len(freqs)):
101         # When the length of associated itemset is 2
102         if output_len == 1:
103             for key in freqs[output_len]:
104                 first = key[0]
105                 second = key[1]
106                 third = (freqs[output_len][key] / 500)  * 100
107 
108                 # Ordinary output
109                 conditional = 0
110                 for j in dbs:
111                     if first in j:
112                         conditional += 1
113                 fourth = (freqs[output_len][key] / conditional) * 100
114 
115                 # Save as a suggested form
116                 line = '{' + '{}'.format(first) + '}\t' + '{' + '{}'.format(second) + '}\t'
    + '{}\t'.format(round(third, 2)) + '{}\n'.format(round(fourth, 2))
117                 print(line)
118                 output.write(line)
119 
120                 # Reverse the previous output
121                 conditional = 0
122                 for j in dbs:
123                     if second in j:
124                         conditional += 1
125                 fourth = (freqs[output_len][key] / conditional) * 100
126 
127                 # Save as a suggested form
128                 line = '{' + '{}'.format(second) + '}\t' + '{' + '{}'.format(first) + '}\t'
    + '{}\t'.format(round(third, 2)) + '{}\n'.format(round(fourth, 2))
129                 print(line)
130                 output.write(line)
```

```
132         # When the length of associated itemset is more than 2
133         else:
134             for key in freqs[output_len]:
135                 list_for_comb = list(key)
136                 for select in range(1, int((len(key) / 2)) + 1):
137                     comb_first = itertools.combinations(list_for_comb, select)
138                     for first in comb_first:
139                         tmp = list_for_comb.copy()
140                         for j in range(len(first)):
141                             tmp.remove(first[j])
142                         second = tuple(tmp)
143                         third = (freqs[output_len][key] / 500) * 100
144 
145                         # Ordinary output
146                         conditional = 0
147                         for j in dbs:
148                             if tuple_in_db(first, j):
149                                 conditional += 1
150                         fourth = (freqs[output_len][key] / conditional) * 100
151                         line = make_output_format(first, second, third, fourth)
152                         output.write(line)
153 
154                         # Reverse the previous output
155                         conditional = 0
156                         for j in dbs:
157                             if tuple_in_db(second, j):
158                                 conditional += 1
159                         fourth = (freqs[output_len][key] / conditional) * 100
160                         line = make_output_format(second ,first, third, fourth)
161                         output.write(line)
162     output.close()
```

- Print the result as a suggested form using make_output_format(params) function

```python
# Make the frequent itemsets until there are no candidates
def apriori(dbs: list, sup: int, freqs: list):

    # Starting at making frequent 2-itemset
    length = 2

    # Initialize for further combinations
    list_for_comb = []
    for i in freqs[0]:
        list_for_comb.append(i)

    while True:
        new_freq = dict()
        ret = dict()
        candidate = []

        for comb in itertools.combinations(list_for_comb, length):
            if subset_satisfy(comb, freqs[length-2], length-1):
                tmp = set()
                for item in comb:
                    tmp.add(item)
                candidate.append(tmp)

        # If no candidate is generated, stop the Apriori progress
        if len(candidate) == 0:
           break

        for itemset in candidate:
            for tx in dbs:
                if tuple_in_db(itemset, tx):
                    if tuple(itemset) in new_freq:
                        new_freq[tuple(itemset)] += 1
                    else:
                        new_freq[tuple(itemset)] = 1

        # Remove the sets which don't satisfy the support
        for key in new_freq:
            if new_freq[key] >= sup:
                ret[key] = new_freq[key]
        freqs.append(ret)
        length += 1
```

- Make frequent itemsets using candidates until there are no candidiates

```python
if __name__ == "__main__":
    min_sup_percent = sys.argv[1]
    input_file_name = sys.argv[2]
    output_file_name = sys.argv[3]

    datas = open(input_file_name, 'r')

    lines = datas.readlines()
    total_xs = len(lines)
    min_sup_times = float(min_sup_percent) * 0.01 * float(total_xs)

    dbs = [] # List of each transactionsi

    # Distinguish the numerical values from the
    for line in lines:
        tmp = re.findall('\d+', line)
        int_tmp = []
        for i in tmp:
            int_tmp.append(int(i))
        dbs.append(int_tmp)

    freqs = [] # List of dictionary(frequent (index)-itemset with support)
    freqs.append(make_first_freq(dbs, min_sup_times))
    apriori(dbs, min_sup_times, freqs)
    make_output(dbs, freqs, output_file_name)
    datas.close()
```

- main func

- first get the command line arguments, and convert the percentage to appear_times

- and refine the datas from input file

-and make progress to Apriori, output file