Data Science
Programming Assignment #1 Report

2013011457 컴퓨터전공
송지수

1. Summary of Algorithm

I implemented Apriori algorithm in Python. I write 2 files, “apriori.py” and “apriori_main.py”. First one has several functions for processing algorithm and the second one has some script codes that actually process the algorithm.

In the “apriori_main.py”, first of all, read and parse the input file and convert all transactions into a list of sets. Then, it make a first candidate set and do the iteration with that. Each iteration carry out pruning, making other candidate set using “join” operation. After the iteration end, then it make list of association rules and print it to the output file.

2. Description of Codes

1. parser

```
def parser(filepath):
    with open(filepath) as file:
        lines = file.read().splitlines()

    transaction_list = []

    for line in lines:
        temp = set()
        for value in line.split('\t'):
            temp.add(value)
        transaction_list.append(temp)

    return transaction_list
```

“parser” function parses input file and make a list of set using transactions data. Read input file line by line and make a set for it and append the set to transaction list.

2. initialize

```
def initialize(value_set):
    candidate_list = []

    for value in value_set:
        temp_dict = {}
        temp_set = set([])
        temp_set.add(value)

        temp_dict['set'] = temp_set
        temp_dict['support'] = 0

        candidate_list.append(temp_dict)

    return candidate_list
```

"initialize" function make a dictionary that has "set" and "support" tuples. Each all values that appears on whole transaction will be the each "set". Support value is set to 0 default.

(Before printing association rules on output file, support means how much does the "set" appear on transactions, not ratio(percent).)

3. support_count

```
def support_count(transaction_list, candidate_list):
    for candidate in candidate_list:
        for transaction in transaction_list:
            if len(candidate['set'].difference(transaction)) == 0:
                candidate['support'] = candidate['support'] + 1
```

"support_count" counts each candidate on candidate set appears how much times on whole transaction list, and write it to the candidate list.

4. pruning

```
def pruning(candidate_list, min_support):
    return [x for x in candidate_list if (x.get('support') > min_support)]
```

"pruning" prunes candidate set that has lower support value than mininum support value.

5. join

```
def join(candidate_list, iter_count):
    candidate_join_list = []

    for candidate_1 in candidate_list:
        for candidate_2 in candidate_list:
            union_set = candidate_1['set'].union(candidate_2['set'])
            if len(union_set) is iter_count and
            union_set not in [li['set'] for li in candidate_join_list]:
                temp_dict = {}
                temp_dict['set'] = union_set
                temp_dict['support'] = 0

                candidate_join_list.append(temp_dict)

    return candidate_join_list
```

"join" does self-joining using candidate set.

6. generating_association_rules

```
def generating_association_rules(transaction_list, candidate_list, min_confidence, transaction_count):
    association_rules_list = []
    for candidate in candidate_list:
        candidate_subset = [x for x in chain.from_iterable(combinations(candidate['set'], r)
            for r in range(1, len(candidate['set'])))]

        for subset in candidate_subset:
            itemset = set(subset)
            associative_itemset = candidate['set'].difference(itemset)
            confidence_count = 0
            count = 0

            for transaction in transaction_list:
                if len(itemset.difference(transaction)) is 0:
                    count += 1
                    if len(associative_itemset.difference(transaction)) is 0:
                        confidence_count += 1

            confidence = float(confidence_count) / float(count) * 100
            support = float(candidate['support']) / float(transaction_count) * 100

            temp_dict = {}
            temp_dict['itemset'] = itemset
            temp_dict['associative_itemset'] = associative_itemset
            temp_dict['confidence'] = confidence
            temp_dict['support'] = support

            association_rules_list.append(temp_dict)

    return association_rules_list
```

"generating_association_rules" generates association rules. First, get a powerset of each candidate set, and divide candidate set into itemset and associative_itemset. Then count how much times that itemset appears on the transactions, and calculate confidence.

7. print_association_rules

```
def print_association_rules(association_rules_list, write_file_name):
    write_file = open(write_file_name, 'w')

    for rule in association_rules_list:
        write_file.write("{" + ",".join(str(item) for item in rule['itemset'])
            + "}\t{" + ",".join(str(item) for item in rule['associative_itemset'])
            + "}\t" + str("%.2f" % rule['support'])
            + "\t" + str("%.2f" % rule['confidence']) + "\n")

    write_file.close()
```

“print_association_rules” prints all of association rules to the output file.

8. main file

```
import apriori
import sys

# command line arguments
min_support_percent = int(sys.argv[1])
input_filepath = sys.argv[2]
output_filepath = sys.argv[3]

# initialize
transaction_list = apriori.parser(input_filepath)
min_support = len(transaction_list) / 100 * min_support_percent
min_confidence = 0

# first candidate set
candidate_dict = {}
candidate_count = 1
candidate_set = apriori.initialize(set().union(*transaction_list))

# iteration
while True:
    apriori.support_count(transaction_list, candidate_set)
    if len(apriori.pruning(candidate_set, min_support)) is 0:
        break
    else:
        candidate_set = apriori.pruning(candidate_set, min_support)
        candidate_dict[candidate_count] = candidate_set
        candidate_count += 1
        candidate_set = apriori.join(candidate_set, candidate_count)

# generate association rules and print it to the output file
association_rules_list = apriori.generating_association_rules(transaction_list,
    candidate_dict[candidate_count-1], min_confidence, len(transaction_list))
apriori.print_association_rules(association_rules_list, output_filepath)
```

“main file” describes order of processing and actually does that.

3. Instructions for compiling

```
songjisu$ python apriori_main.py 5 input.txt output.txt
```

Algorithm runs in python.

- first argument : python file name (apriori_main.py)
- second argument : minimum support (percent)
- third argument : input file name
- fourth argument : output file name