< Data Science (ITE4005) >

Programming Assignment#1 – Apriori Algorithm

소프트웨어 전공

2013010926 강은석

1. Introduction

: This is an assignment using 'Apriori Algorithm' that learned in Data Science class.

"Apriori" is an algorithm for frequent item set mining and association rule through transactional database. It proceeds by identifying the frequent individual item sets in the transactional database and extending them. Frequent subsets are extended one item at a time, and candidates are tested against database. It terminates when no further extension is found.

Apriori's main concept is extension and pruning. If the items are larger than the minimum support value in the entire data base, they are determined as frequent pattern. And It generates candidate item sets of length N from frequent item sets of length N-1. Then If candidate item sets have no frequent item sets that are sub set of candidates, Pruning the candidate item set. After, Scanning the database to determine which candidate item sets are frequent patterns.

2. Summary of My algorithm

: In my assignment, it consists of four main functions. First, read input file and make item sets with a length of one and check if they are frequent pattern. Then, it repeats the remained three functions until no further extension of item sets.

1. Remove no frequent item set in candidates that are lower than minimum support value. Save no frequent item set for pruning candidates and frequent item set for extension.
2. Make length N + 1 candidate item sets from length N frequent patterns by using combination function.
3. If each candidate item set has any length N no frequent patterns, Pruning that candidate item set.

After end of above functions, calculate support value and confidence of each frequent item sets and save output file.

3. Description of codes

```python
def main(min_sup, input_file, output_file):
    min_sup = int(min_sup)
    join_count = 1
    trans = read_input_file(input_file)
    patterns, re_patterns = make_one_itemset(trans, min_sup)  # make 1 length frequent patterns
    while len(patterns) != 0:
        join_count += 1
        # remove patterns that are lower than min support
        if join_count > 2:
            fre_patterns, re_patterns = remove_by_minsup(trans, patterns, min_sup)
        if join_count == 2:
            fre_patterns = patterns

        next_patterns = extension_itemset(fre_patterns, join_count)   # make next patterns from frequent patterns
        patterns = pruning(next_patterns, re_patterns)  # pruning patterns that include removed patterns

    output_data = calcul_sup_conf(save_freq_patterns)  # calculate support and confidence
    write_output_file(output_data, output_file)  # make output file


if __name__ == "__main__":
    main(sys.argv[1], sys.argv[2], sys.argv[3])
```

그림 1 - main function

: Main function takes three arguments that are minimum support value, input file name and output file name. Function Process is same as description of "2. Summary of my algorithm". Make length 1 frequent patterns. Check item set is frequent pattern by "remove_by_minsup()" function, extend by "extension_itemset()" function and prune candidate item set by "pruning()" function.

```python
def read_input_file(input_file):    # read input file and make transaction data function
    transaction = []
    with open(input_file) as f:
        lines = f.readlines()
        for line in lines:
            line = line.strip().split('₩t')
            transaction.append(line)

        f.close()
    return transaction
```

그림 2 - read input file function

: Read input file that is transactional database. Make transaction variable.

```python
def make_one_itemset(transaction, min_sup):  # make 1 length frequent patterns
    candidates = []
    removed_patterns = []   # use for remove length 2 patterns that are not frequent pattern
    for tran in transaction:
        candidates += tran
    candidates = list(set(candidates))
    item_set = []
    for pattern in candidates:   # find 1 length frequent patterns and count
        count = 0
        for tran in transaction:
            if pattern in tran:
                count += 1
        if int(count/len(transaction)*100) >= min_sup:  # check item is frequent
            item_set.append(pattern)
            key = tuple([pattern])
            save_freq_patterns[key] = [round(count/len(transaction)*100, 2), count]
        else:  # save no frequent patterns
            removed_patterns.append(pattern)

    return item_set, removed_patterns
```

그림 3 – make one item set function

: Check item set with length of one, it's count is larger than "min_sup". Than, save frequent item to "item_set" and no frequent item to "removed_patterns".

```python
def remove_by_minsup(trans, candidates, min_sup):  # remove no frequent patterns in candidate patterns
    freq_patterns = []
    removed_patterns = []
    for key in candidates:
        count = 0
        for tran in trans:  # count candidate pattern in transactions
            if all(elem in tran for elem in key):
                count += 1
        if int(count/len(trans)*100) >= min_sup:  # check item is frequent pattern
            freq_patterns.append(key)
            key = tuple(sorted(key))
            save_freq_patterns[key] = [round(count/len(trans)*100, 2), count]
        else:  # save no frequent patterns
            removed_patterns.append(key)

    return freq_patterns, removed_patterns
```

그림 4 - remove by minsup function

: remove candidate item set that is lower than "min_sup".

Save "freq_patterns" for item set extension and "removed_patterns" for pruning candidate item set.

```python
def extension_itemset(freq_patterns, join_count):
    # make candidate patterns from frequent patterns ex) length N -> N+1
    patterns = []
    if join_count > 2:
        for freq in freq_patterns:
            for i in range(len(freq)):
                patterns.append(freq[i])
    else:
        patterns = freq_patterns

    patterns = list(set(patterns))
    next_patterns = list(combinations(patterns, join_count))   # make length+1 patterns

    return next_patterns
```

그림 5 - extension item set function

: Extract no duplicated item set from “freq_patterns” and Make length + 1 item set by using Combination function. (make length N+1 from length N.)

```python
def pruning(next_patterns, removed_patterns):  # pruning patterns that include removed patterns
    candidate_patterns = []
    # if pattern include removed patterns, pattern is not frequent pattern
    for pattern in next_patterns:
        flag = True
        for r_pattern in removed_patterns:
            if all(elem in pattern for elem in r_pattern):
                flag = False
                continue
        if flag == True:
            candidate_patterns.append(pattern)

    return candidate_patterns
```

그림 6 - pruning candidates function

: Prune the candidate item set that are include any “removed_patterns”

“removed_patterns” are no frequent patterns that result of “remove_by_minsup” function.

```
def calcul_sup_conf(freq_patterns):  # calculate frequent pattern support and confidence
    result = []
    for key in freq_patterns.keys():
        if len(key) != 1:
            for i in range(1, len(key)):  # make frequent pattern's sub set
                item_set = list(combinations(key, i))
                for set in item_set:  # calculate support confidence
                    association_set = list(key)
                    set_list = list(set)
                    for s in set_list:  # make association item set
                        association_set.pop(association_set.index(s))
                    result.append([set, association_set, freq_patterns[key][0],
                                   round(freq_patterns[key][1]/freq_patterns[set][1]*100, 2)])

    return result
```

그림 7 - calculate support and confidence function

: About all frequent item set, get the sub set of frequent item set, make association set and calculate support and confidence variable.

"freq_patterns" is dict. key is frequent item set and value is [support, count].
"reusult" contain [item set, association set, support, confidence].

```
def write_output_file(output_data, output_file):
    # make output file from calculate support and confidence data
    outputs = []
    for data in output_data:
        out = "{"
        for i in range(len(data[0])):
            if i == len(data[0])-1:
                out += data[0][i]
            else:
                out = out + data[0][i] + ","
        out += "}₩t{"
        for i in range(len(data[1])):
            if i == len(data[1])-1:
                out += data[1][i]
            else:
                out = out + data[1][i] + ","

        out = out + "}₩t" + str(data[2]) + "₩t" + str(data[3]) + "₩n"
        outputs.append(out)

    f = open(output_file, "w")
    for out in outputs:
        f.write(out)
    f.close()
```

그림 8 - write output file function

: make output file which name is "output_file". "output_file" is argument.

4. How to compile code

```
1>python apriori.py 5 input.txt output.txt
```

그림 9 - compile and run code by python file

: python [py file name] [minimum support] [input file name] [output file name]

Python file and input file are must in same path.

And output file is created in same path.

```
:1>apriori.exe 15 input.txt output.txt
```

그림 10 - run apriori.exe file

: apriori.exe [minimum support] [input file name] [output file name]

exe file and input file are must in same path.

And output file is created in same path.

5. Result of test (use given input data)

```
>apriori.exe 15 input.txt output.txt
```

그림 11 - minimum support = 15%

{1}	{8}	15.4	51.68
{8}	{1}	15.4	34.07
{1}	{16}	16.2	54.36
{16}	{1}	16.2	38.21
{3}	{8}	25.8	86.0
{8}	{3}	25.8	57.08
{16}	{3}	25.2	59.43
{3}	{16}	25.2	84.0
{16}	{8}	30.2	71.23
{8}	{16}	30.2	66.81
{16}	{3,8}	24.0	56.6
{3}	{16,8}	24.0	80.0
{8}	{16,3}	24.0	53.1
{16,3}	{8}	24.0	95.24
{16,8}	{3}	24.0	79.47
{3,8}	{16}	24.0	93.02

그림 12 – output.txt