

**DataScience Class**

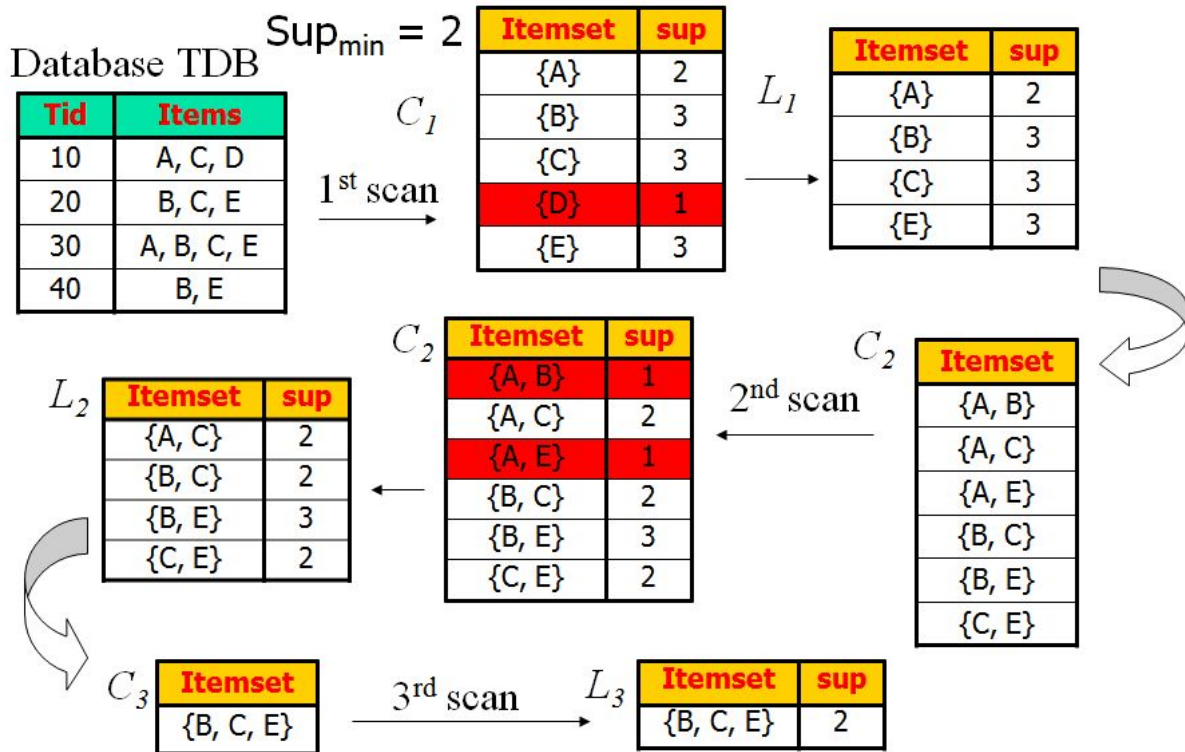
# Apriori Algorithm

2016025096 강수진

# 1. Apriori algorithm

Apriori is an algorithm for **frequent item set mining** and **association rule learning** over relational databases. The frequent item sets determined by Apriori can be used to determine association rules

Apriori algorithm has pruning principle : **If there is any itemset which is infrequent, its superset should not be generated/tested.** It is helpful to reduce the number of candidates.



## process

- Initially, scan DB once to get frequent 1-itemset
  - first candidates include all item ids. First frequent itemset is obtained by scanning database and getting support value.
- Generate candidate itemsets of length  $(k+1)$  from frequent itemsets of length  $k$ 
  - For generating candidates, self-joining and pruning can be used.
  - To be adequate candidates, all subsets in that item set should be a frequent item set.
- Test the candidates against DB
  - Find frequent item sets whose support value are more than minimum support
- Terminate when no frequent or candidate set can be generated

## Pseudo code

```
 $C_k$ : Candidate itemset of size  $k$   
 $L_k$ : frequent itemset of size  $k$   
  
 $L_1 = \{\text{frequent items}\};$   
for ( $k = 1; L_k \neq \emptyset; k++$ ) do begin  
     $C_{k+1}$  = candidates generated from  $L_k$ ;  
    for each transaction  $t$  in database do  
        increment the count of all candidates in  $C_{k+1}$   
        that are contained in  $t$   
     $L_{k+1}$  = candidates in  $C_{k+1}$  with min_support  
    end  
return  $\cup_k L_k$ ;
```

## 2. Code implementation

### (1) Code structure

To simplify my code, I listed all classes with their member variables and method without specifying data types or return values.

#### class AprioriProject

private:

```
numOfScan;//The number of scanning database  
minSupport;//minimum support specified by users  
transactions;//database transactions  
candidates;//current generated candidates  
frequentItemsets;//<frequent itemset,support> in every scans  
associationRules;//association rules based on frequent item sets
```

public:

```
AprioriProject();//constructor  
run();//method for running apriori algorithm  
getAssociationRules();//getter  
generateCandidates();//method for generating candidates  
joining();//first process of generating candidates  
pruning();//second process of generating candidates  
findFrequentItems();//select item set which has support of more than  
specified threshold
```

**generateAssociationRule()**//generate association rule after getting frequent item set

## class IO

private:

**string inputFile**;//input file name ex. input.txt  
**string outputFile**;//output file name ex. output.txt  
**ifstream readFile**;//input file stream  
**ofstream writeFile**;//output file stream  
**vector<vi>transactions**;//to store transactions from input file

public:

**IO()**//constructor  
**getTransactions()**//getter  
**inputRead()**//read input file  
**createOutputFile()**//create output file  
**itemsetToStr()**/change vector to string  
**split()**//split string by delimiter

**struct association**//structure for association rule x->y with support and  
//confidence value

xltemset;  
yltemset;  
support;  
confidence;  
};

## (2) Function description

I explained each member variables and functions(method) in details.

### - AprioriProject class

AprioriProject class is for running apriori algorithm and generating association rules based on frequent item sets.

## member variables

```

int numOfScan;
double minSupport;
vector<vi> transactions;
vector<vi> candidates;
vector<map<vi, double>> frequentItemsets;
vector<association> associationRules;

```

**vi** : is the abbreviation for vector<int>

**numOfScan** : the number of database scan(== the number of iterations)

**minSupport** : minimum support(threshold)

**frequentItemsets[i]** : map <frequent item set, support> in i th iteration

**frequentItemsets** : All the frequent item sets which are found so far

**transactions** : sequence of item id lists

**candidates** : candidates for finding frequent item sets(the most recent ones)

## constructor

```

AprioriProject(double minSup, vector<vi> trans){
    numOfScan = 0;
    minSupport = minSup;
    transactions.assign(trans.begin(), trans.end());
    map<vi, double> tmp;
    tmp.insert(pvi({}, 0));
    frequentItemsets.push_back(tmp); //for index 0
}

```

This is a constructor in AprioriProject class.

For index 0 in frequentItemsets, I added empty variable in map data type.

## run()

```

void run(){
    while(true){
        candidates = generateCandidates();
        if(candidates.size()==0) break;
        numOfScan++;
        findFrequentItems();
        for(auto frequentItems: frequentItemsets[numOfScan])
            generateAssociationRule(frequentItems.first);
    }
}

```

This is for running apriori algorithm. It runs until no more candidates are generated.

For every iteration, candidates are generated based on apriori algorithm, frequent item sets are selected which have more than minimum support and association rules are generated from frequent item sets.

### getAssociationRules()

```
vector<association> getAssociationRules(){  
    return associationRules;  
}
```

For creating output file, we need the variable 'associationRules'.

### generateCandidates()

```
vector<vector<int>> generateCandidates(){  
    if(numOfScan==0){  
        set<int>s;  
        vector<vi>c;  
        for(auto transaction:transactions)  
            for(auto item:transaction)  
                s.insert(item);  
        for(auto item:s)  
            c.push_back({item});  
        return c;  
    }  
    else{  
        //joining and pruning  
        vector<vi> joinedItems=joining();  
        return pruning(joinedItems);  
    }  
}
```

For generating next candidates, apriori algorithm has self-joining and pruning process. From previous candidates, new itemsets are created by self-joining and some of them are removed by pruning.

In initial step(numOfScan==0), each items with length 1 are candidates. I used 'set' data structure to eliminate duplicate item. I'll explain joining() and pruning() methods in detail below.

### joining()

```

vector<vi> joining(){
    map<vi,double>prevL=(map<vi,double>)frequentItemsets[numOfScan];
    vector<vi> joinedItems;
    //self-joining
    for(pvi L1:prevL){
        for(pvi L2:prevL){
            if(L1>=L2)continue;
            int cnt=0;
            for(int k=0;k<numOfScan-1; k++) {
                if(L1.first[k] == L2.first[k])
                    cnt++;
            }
            if(cnt == numOfScan-1) {
                vi unionVector;
                unionVector.resize((int)(L1.first.size()));
                copy(L1.first.begin(),L1.first.end(),unionVector.begin());
                unionVector.push_back(L2.first[numOfScan-1]);
                if(unionVector[numOfScan-1]>unionVector[numOfScan])
                    swap(unionVector[numOfScan-1],unionVector[numOfScan]);
                joinedItems.push_back(unionVector);
            }
        }
    }
    return joinedItems;
}

```

In apriori algorithm, any subset of a frequent itemset must be a frequent. By following this rule, I selected itemset as a candidate if union of two item set pair has (numOfScan+1) length and they have same item id except last item id.(because all of the candidates are sorted in ascending order) This condition is helpful to eliminate duplicate candidates.

## pruning()

```

vector<vi> pruning(vector<vi>joinedItems){
    vector<vi>c;
    map<vi,double>prevL=(map<vi,double>)frequentItemsets[numOfScan];
    //pruning
    for(auto candidate:joinedItems){
        int cnt=0;
        for(int i=0;i<candidate.size();i++){
            vi tmp = candidate;
            tmp.erase(tmp.begin()+i);
            if(prevL.find(tmp) == prevL.end()){
                cnt++;
                break;
            }
        }
        if(cnt==0)
            c.push_back(candidate);
    }
    return c;
}

```

In vector<vi> joinedItems, self-joined item sets are stored. For every element of joinedItems, all the subsets should be existed in previous frequent item sets. It is enough to check only the previous frequent item sets.



## findFrequentItems()

```
void findFrequentItems(){
    map<vi,double>frequent;
    for(auto candidate:candidates){
        double support=0.0;
        for(auto transaction:transactions){
            int cnt=0;
            for(auto item:candidate){
                if(find(transaction.begin(),transaction.end(),item)==transaction.end()){
                    cnt++;
                    break;
                }
            }
            if(cnt==0)
                support+=1.0;
        }
        support = support/transactions.size()*100;
        if(support>=minSupport){
            frequent[candidate]=support;
        }
    }
    frequentItemsets.push_back(frequent);
}
```

For every iteration, I call findFrequentItems method. This method counts the number of each item id in transactions and get support value for each candidates. If support value of candidates are more than minimum support value, they are stored in frequentItemsets in map format.

## generateAssociationRule()

```
void generateAssociationRule(vi items){
    double support = frequentItemsets[numOfScan][items];
    for(int i=1;i<numOfScan;i++){
        for(auto pair:frequentItemsets[i]){
            vi xItemset;
            vi yItemset;
            map<int,bool>xChecker;

            for(int item:pair.first)
                xChecker[item]=true;
            int xcnt=int(xChecker.size());
            int ycnt=0;
            for(int item:items){
                if(xChecker.find(item)==xChecker.end()){
                    yItemset.push_back(item);
                    ycnt++;
                }
            }
            if(xChecker.size()+ycnt==items.size()){
                for(auto item:xChecker)
                    xItemset.push_back(item.first);
                double xconfidence = support/frequentItemsets[xcnt][xItemset]*100;
                xconfidence = round(xconfidence*100)/100;
                associationRules.push_back({xItemset,yItemset,support,xconfidence});
            }
        }
    }
}
```



For every iteration, new frequent itemsets are generated. Based on them, I can generate association rules with this method. For  $x \rightarrow y$  association rule, frequent itemsets with 1 to numOfScan-1 length can be  $x$  itemsets. I checked if previous frequent itemsets are subset of current item sets and get support value from frequentItemsets map.

### - IO class

IO class has two roles of reading input file and writing output file.

#### member variable

```
string inputFile;  
string outputFile;  
ifstream readfile;  
ofstream writefile;  
vector<vi> transactions;
```

ex. inputFile = input.txt, outputFile = output.txt

To read input file, I used ifstream(input file stream) and to write output file, I used ofstream(output file stream). Input file data is stored in transactions variable.

#### constructor

```
IO(string input, string output){  
    inputFile=input;  
    outputFile=output;  
}
```

This is a constructor for io class.

#### getTransactions()

```
vector<vi> getTransactions(){  
    return transactions;  
}
```

I added this method for passing transaction vectors to apriori project class.

#### inputRead()

```

void inputRead(){
    readFile.open(inputFile);
    if(readFile.is_open()) {
        string s;
        while(getline(readFile, s)){
            vector<int> tmp;
            vi parse = split(s, '\t');
            if(parse.size()>0){
                transactions.push_back(parse);
            }
        }
        readFile.close();
    }
    else{
        cout << "Input file is not opened\n";
        exit(0);
    }
}

```

This is the method for reading input file. I used input file stream and pushed the splitted value to transactions vector.

### createOutputFile()

```

void createOutputFile(vector<association> associationRules){
    writeFile.open(outputFile);
    writeFile.precision(2);
    writeFile.setf(ios::fixed);
    writeFile.setf(ios::showpoint);
    if(writeFile.is_open()) {
        for(association item:associationRules){
            writeFile<<itemsetToStr(item.xItemset)<<'\t';
            writeFile<<itemsetToStr(item.yItemset)<<'\t';
            writeFile<<item.support<<'\t';
            writeFile<<item.confidence<<'\n';
        }
        writeFile.close();
    }
    else{
        cout << "Output file is not opened\n";
        exit(0);
    }
}

```

This is the method for writing output file. I used output file stream and write values to 2 decimal places.

## itemsetToStr()

```
string itemsetToStr(vector<int> itemSet) {
    string str = "{";
    for(int item:itemSet){
        str += to_string(item);
        if(item != itemSet.back())
            str += ",";
    }
    str += "}";
    return str;
}
```

To write output file in specified format, I added string converter method.

## split()

```
vector<int> split(string str, char delimiter) {
    vector<int> ret;
    stringstream ss(str);
    string temp;
    while (getline(ss, temp, delimiter))
        ret.push_back(stoi(temp));
    return ret;
}
```

For splitting string by tab delimiter, I added split method.

## 3. Instructions for compilation

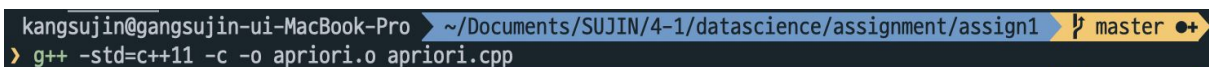
### Environment

OS : Mac OS

Language : C++

I used xcode so, I also attached the folder separately.

1.



```
kangsujin@gangsujin-ui-MacBook-Pro ~/Documents/SUJIN/4-1/datascience/assignment/assign1 master ➤
> g++ -std=c++11 -c -o apriori.o apriori.cpp
```

command : **g++ -std=c++11 -c -o apriori.o apriori.cpp**

-> apriori.o (object file) is created.

2.

```
kangsujin@gangsujin-ui-MacBook-Pro ~/Documents/SUJIN/4-1/datascience/assignment/assign1 master ●+  
> g++ -std=c++11 -o apriori.exe apriori.o
```

command : **g++ -std=c++11 -o apriori.exe apriori.o**

-> apriori.exe (executable file) is created.

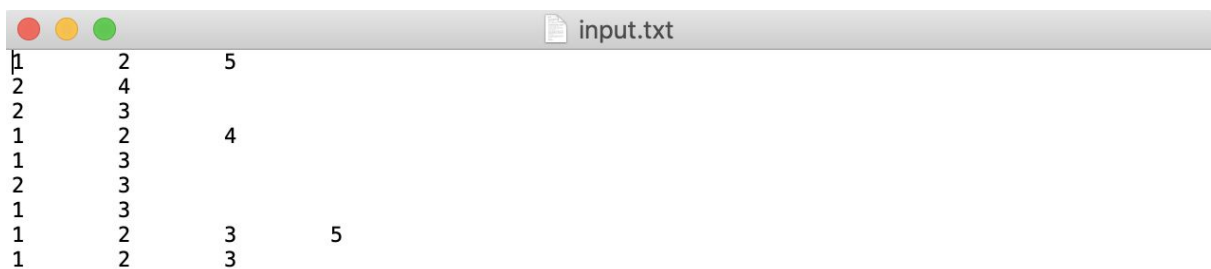
3.

```
kangsujin@gangsujin-ui-MacBook-Pro ~/Documents/SUJIN/4-1/datascience/assignment/assign1 master ●+  
> ./apriori.exe 5 input.txt output.txt
```

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

-> output file is created

## 4. Result



The screenshot shows a text file named 'input.txt' with the following content:

1	2	5	
2	4		
2	3		
1	2	4	
1	3		
2	3		
1	3		
1	2	3	5
1	2	3	

For input.txt file above, I got output.txt file below. In input.txt, each row denotes transaction.

output.txt			
{1}	{2}	44.44	66.67
{2}	{1}	44.44	57.14
{1}	{3}	44.44	66.67
{3}	{1}	44.44	66.67
{1}	{4}	11.11	16.67
{4}	{1}	11.11	50.00
{1}	{5}	22.22	33.33
{5}	{1}	22.22	100.00
{2}	{3}	44.44	57.14
{3}	{2}	44.44	66.67
{2}	{4}	22.22	28.57
{4}	{2}	22.22	100.00
{2}	{5}	22.22	28.57
{5}	{2}	22.22	100.00
{3}	{5}	11.11	16.67
{5}	{3}	11.11	50.00
{1}	{2,3}	22.22	33.33
{2}	{1,3}	22.22	28.57
{3}	{1,2}	22.22	33.33
{1,2}	{3}	22.22	50.00
{1,3}	{2}	22.22	50.00
{2,3}	{1}	22.22	50.00
{1}	{2,4}	11.11	16.67
{2}	{1,4}	11.11	14.29
{4}	{1,2}	11.11	50.00
{1,2}	{4}	11.11	25.00
{1,4}	{2}	11.11	100.00
{2,4}	{1}	11.11	50.00
{1}	{2,5}	22.22	33.33
{2}	{1,5}	22.22	28.57
{5}	{1,2}	22.22	100.00
{1,2}	{5}	22.22	50.00
{1,5}	{2}	22.22	100.00
{2,5}	{1}	22.22	100.00
{1}	{3,5}	11.11	16.67
{3}	{1,5}	11.11	16.67
{5}	{1,3}	11.11	50.00
{1,3}	{5}	11.11	25.00
{1,5}	{3}	11.11	50.00
{3,5}	{1}	11.11	100.00
{2}	{3,5}	11.11	14.29
{3}	{2,5}	11.11	16.67
{5}	{2,3}	11.11	50.00
{2,3}	{5}	11.11	25.00
{2,5}	{3}	11.11	50.00
{3,5}	{2}	11.11	100.00
{1}	{2,3,5}	11.11	16.67
{2}	{1,3,5}	11.11	14.29
{3}	{1,2,5}	11.11	16.67
{5}	{1,2,3}	11.11	50.00
{1,2}	{3,5}	11.11	25.00
{1,3}	{2,5}	11.11	25.00
{1,5}	{2,3}	11.11	50.00
{2,3}	{1,5}	11.11	25.00
{2,5}	{1,3}	11.11	50.00
{3,5}	{1,2}	11.11	100.00
{1,2,3}	{5}	11.11	50.00
{1,2,5}	{3}	11.11	50.00
{1,3,5}	{2}	11.11	100.00
{2,3,5}	{1}	11.11	100.00