# Data Mining and Bioinformatics

# CSE601

# PROJECT 1

**Devavrat D Raikar**
**Devavrat**
**50291115**

**Divya JeevanKumar Sanghvi**
**Divyajee**
**50288057**

# Apriori Algorithm:

The main task of Apriori algorithm is generate frequent item sets and associate the item sets by rules on transactional databases. An Anti-monotone property states any subset of a frequent item set must also be frequent and any superset of an infrequent item set must also be an infrequent. The idea was to prune all the infrequent items from the set whose support was lesser than the minimum support count.

**Task 1:**

**Frequent itemset generation:**

We had to generate all the frequent items that satisfied the criteria of being greater than the minimum support. Different length of frequent itemset were generated unto n-1 items of the transaction database.

**Task 2:**

**Association Rule generation:**

In this we had to generate association rules from the frequent itemset generated in the previous step. The rules are generated based on a specific format.

**Flow of the Algorithm:**

**Language: Python programming**

**Library: pandas, itertools**

**Steps:**

**TASK 1:**

- Data is read from the input file and it is preprocessed based as per the given format of G1_Up or G1_Down
- The algorithm was set to different support values such as 30%, 40%, 50%, 60% and 70% and confidence of 70% to generate frequent itemset
- The first step was to compute length-1 frequent itemset from the input data. Iterate over the list of the data and store its unique elements from the list and increment its count. We have stored the frequency of each occurrences of the item by using groupby() function. If each item frequency is greater than the minimum support, then add it to the candidate list of the frequent itemset generator.

- aprioriAlgorithm() function was written to generate the next length frequent itemset and candidate item list. In this algorithm, we send the first initial candidate list, next length frequent itemset to be generated and total number of frequent itemset in the initial case.
- candidateItemGenerator() function is called from aprioriAlgorithm() to generate next candidate itemset. In this we get combinations of each items and return the unique items generated from it.
- pruneCandidateItemset() compares the candidateItem() with the Input data set to check the occurrences of the itemset. If the itemset is present in the input data item then we count its frequency. If its frequency is greater than the support of the algorithm that item is added on the next candidate item list, if not its pruned and all its subsequent item sets are also pruned.
- Once the pruned itemset is obtained, we get the union of its set so that we can generate next candidate itemlist and it returns the length of the highest frequent itemset so we can proceed to next frequent itemset generator.
- Once we have tried with all the different supports and confidence we need to generate association rules for all those frequent item sets.

**TASK 2:**

- The query is parsed and based on the template number each of the function is called. If its template 1 then we have “RULE’,”HEAD”,”BODY” and “ANY”,”NONE”,1 as the tags and an item. If the query satisfies all these requirements then the count of the occurrences is increased. The item is checked in each of its column in the frequent item list generated.
- If the template is 2 then we have to search for items with an item count greater than what ever is given in rule.
- Template 3 is combination of template 1 and template 2

# FREQUENT ITEMSET OUTPUT

**support 70%, Confidence 70%**

- number of length-1 frequent itemsets 7
- Total frequent itemsets 7

Association rule count 0

**support 60%, Confidence 70%**

- number of length-1 frequent itemsets 34
- number of length-2 frequent itemsets 2

Total frequent itemsets 36

Association rule count 4

**support 50%, Confidence 70%**

- number of length-1 frequent itemsets 109
- number of length-2 frequent itemsets 63
- number of length-3 frequent itemsets 2

Total frequent itemsets 174

Association rule count 117

**support 40%, Confidence 70%**

- 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

Total frequent itemsets 1077

Association rule count 1130

**support 30%,, Confidence 70%**

- 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

Total frequent itemsets 12879

## Q U E R Y  O U T P U T

**Support = 50%, Confidence = 70%**

**Template 1**

- asso_rule.template1("RULE", "ANY", ['G59_UP']) --- 26
- asso_rule.template1("RULE", "None", ['G59_UP']) --- 91
- asso_rule.template1("RULE", 1, ['G59_UP', 'G10_Down']) --- 39
- asso_rule.template1("HEAD", "ANY", ['G59_UP']) --- 9
- asso_rule.template1("HEAD", "NONE", ['G59_UP']) --- 108
- asso_rule.template1("HEAD", 1, ['G59_UP', 'G10_Down']) --- 17
- asso_rule.template1("BODY", "ANY", ['G59_UP']) --- 17
- asso_rule.template1("BODY", "NONE", ['G59_UP']) --- 100
- asso_rule.template1("BODY", 1, ['G59_UP', 'G10_Down']) --- 24

**Template 2**

- asso_rule.template2("RULE", 3) --- 9
- asso_rule.template2("HEAD", 2) --- 6
- asso_rule.template2("BODY", 1) --- 114

**Template 3**

- asso_rule.template3("1or1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_UP']) --- 24
- asso_rule.template3("1and1", "HEAD", "ANY", ['G10_Down'], "BODY", 1, ['G59_UP']) --- 1
- asso_rule.template3("1or2", "HEAD", "ANY", ['G10_Down'], "BODY", 2) --- 11
- asso_rule.template3("1and2", "HEAD", "ANY", ['G10_Down'], "BODY", 2) --- 0

- asso_rule.template3("2or2", "HEAD", 1, "BODY", 2)  ---  111
- asso_rule.template3("2and2", "HEAD", 1, "BODY", 2)  ---  3

**Code Snippet:**

```
#############################################################################
# This function is used to generate candidate itemsets of desired length
#############################################################################
def candidateItemsetGenerator(inputItemset,desiredLength):
    candidateItemset = itertools.combinations(inputItemset,desiredLength)

    temporary = []
    for itr in list(candidateItemset):
        temporary.append(set(itr))

    return temporary

#############################################################################
# This function is used to Prune the Candidate Itemset
#############################################################################
def pruneCandidateItemset(inputItemset,totalFrequentItemset):
    pruned_Itemset = []
    for item in inputItemset:
        currentSupportCount = 0
        for row in inputDataList:
            if(item.issubset(row)):
                currentSupportCount += 1
        if(currentSupportCount >= support):
            pruned_Itemset.append(item)
            if(item != None):
                temp = list(item)
                temp.sort()
                #totalFrequentItemset[str(set(temp))] = currentSupportCount
                totalFrequentItemset.loc[len(totalFrequentItemset)] = pd.Series({'Itemset':set(temp), 'Support':currentSupportCount })

    return pruned_Itemset

#############################################################################
# This function is used to compute all the frequent itemset
#############################################################################
def aprioriAlgorithm(candidateItemset,maximumItemsetLength,totalFrequentItemset):
    for itr in range(len(inputData.columns)-1):
        # Call candidateItemsetGenerator() to generate candidate itemsets
        currentCandidateItemset = candidateItemsetGenerator(candidateItemset, itr + 1)

        # Call pruneCandidateItemset() to refine the candidate itemset and
        # keep only those elements which satisfy the minimum support requirement
        prunedItemset = pruneCandidateItemset(currentCandidateItemset,totalFrequentItemset)

        # If after pruning the size of the list is zero
        # break the loop as no more candidates can be generated
        if(len(prunedItemset) == 0):
            break
        else:
            # Otherwise print the total number of candidates generated
            ss = "number of length-" + str(itr+1) + " frequent itemsets"
            lengthOfItemsets.loc[len(lengthOfItemsets)] = pd.Series({'Type':ss,'Count':str(len(prunedItemset))})
        # Perform union operation on the prunedItemset with itself
        # to compute combinations for next candidate itemset
        candidateItemset = set.union(*prunedItemset)
```