Department of
Computer Science and Engineering

Title: Implementation of Apriori algorithm for frequent itemset mining

Data Mining Lab

CSE 424

Green University of Bangladesh

1 Objective(s)

- To gather knowledge of the basics of frequent itemset mining.
- To implement the Apriori algorithm on a real-world dataset.

2 Problem analysis

2.1 Overview of itemset mining

Frequent patterns are patterns (e.g., itemsets, subsequences, or substructures) that appear frequently in a data set. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a frequent itemset. Finding frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data. Moreover, it helps in data classification, clustering, and other data mining tasks. Thus, frequent pattern mining has become an important data mining task and a focused theme in data mining research. Frequent pattern mining searches for recurring relationships in a given data set.

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets.With massive amounts of data continuously being collected and stored, many industries are becoming interested in mining such patterns from their databases. For example, Figure 1 shows few customers buying few things. Now a data analyst needs to decide which itemsets are to be considered for occurring frequently for a certain pre-defined threshold. If this threshold is, say, 2, then the itemset milk, bread is a frequent itemset mined from this small dataset of 3 customers. Now using this information, developers or designers can indeed plan many thigns for business and real-life applications. For data mining purpose, transactional databases are used. A sample transcational database is shown in Figure 2. In transactional datasests, one column contains the transaction IDs while the corresponding column holds the set of items for that transaction.

Figure 1: Frequent itemset mining for 3 customers for threshold equal to 2. The itemset milk, bread is a frequent mined pattern.

TID	*List of item_IDs*
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Figure 2: An example transactional database containing 9 transactions.

2.2 Apriori algorithm basics

To mine out frequent itemsets from transactional databases, Apriori algorithm is the most basic one. The algorithm uses prior knowledge of frequent itemset properties. Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. First, the set of frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted by L1. Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found. The finding of each L_k requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property is used to reduce the search space. The Apriori property states that “All nonempty subsets of a frequent itemset must also be frequent”. By definition, if an itemset I does not satisfy the minimum support threshold, min_sup, then I is not frequent, that is, P (I) < min_sup. If an item A is added to the itemset I, then the resulting itemset (i.e., I ∪ A) cannot occur more frequently than I. Therefore, I ∪ A is not frequent either, that is, P (I ∪ A) < min_sup.The Apriori algorithm implements the following two steps.

1. Joining Step: To find L_{k+1}, a set of candidate (k + 1)-itemsets is generated by joining L_k with itself. This set of candidates is denoted C_{k+1}. Each member of L_k needs to be joined with another member of the same set L_k. For further efficient implementation, every joining is not needed to be done. For this let us consider two member k-itemsets, A and B, in the L_k set of itemsets. A must be joined with B if and only if:

   (a) A≠B

   (b) all the items of A from i = 1 to the i = k − 1, where i denotes the items in the itemset A, is equal to all the items of B in the same positions exactly

   (c) if the k-th item in A < k-th item in B

   The resulting joined output itemset, a member of C_{k+1} will have all items of A and the k-th item of B appended at the end, making this new itemset a candidate (k + 1)-itemset.

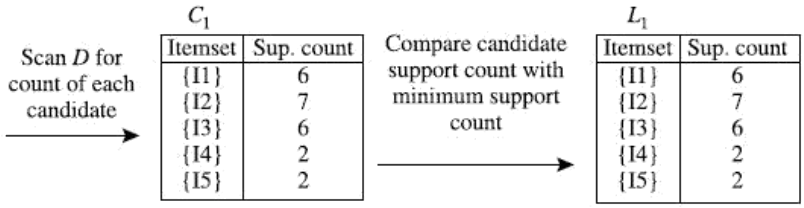
2. Pruning Step: C_{k+1} is a superset of L_{k+1}, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_{k+1}. A database scan to determine the count of each candidate in C_{k+1} would result in the determination of L_{k+1}. Since C_{k+1} can be huge, so the Apriori proerty can be used here to reduce the size of C_{k+1}. Any k-itemset that is not frequent cannot be a subset of a frequent (k + 1)-itemset. Hence, if any k-subset of a candidate k + 1-itemset is not in L_k then the candidate cannot be frequent either and so can be removed from C_{k+1}. Thus the size of C_{k+1} is reduced using the property. After that, a single databse scan of this improved C_k can easily generate the finalized L_k.

2.3 Illustration of Apriori algorithm

For the illustration, we consider the transactional dataset of Figure 2 and define that min_sup = 2. To generate all the frequent itemset patterns, the following steps are carrid out using the Apriori algorithm.

- In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, C_1. The algorithm simply scans all of the transactions to count the number of occurrences of each item. The set of frequent 1-itemsets, L_1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. In our example, all of the candidates in C_1 satisfy minimum support.
- To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join $L_1 \triangleright\triangleleft L_1$ to generate a candidate set of 2-itemsets, C_2. Note that no candidates are removed from C_2 during the prune step because each subset of the candidates is also frequent.
- Next, the transactions in D are scanned and the support count of each candidate itemset in C_2 is accumulated. The set of frequent 2-itemsets, L_2, is then determined, consisting of those candidate 2-itemsets in C_2 having minimum support.
- After that step, using the new join step, we first get $C_3 = L_2 \triangleright\triangleleft L_2$ = {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}. Based on the Apriori property that all subsets of a frequent itemset must also be frequent, we can determine that the four latter candidates cannot possibly be frequent.We therefore remove them from C_3, thereby saving the effort of unnecessarily obtaining their counts during the subsequent scan of D to determine L_3. Note that when given a candidate k-itemset, we only need to check if its (k − 1)-subsets are frequent since the Apriori algorithm uses a level-wise search strategy. The resulting pruned version of C_3 is shown in the bottom row of the entire illustration in Figure 3.
- The transactions in D are scanned to determine L_3, consisting of those candidate 3-itemsets in C_3 having minimum support.
- Finally, for the last time, the algorithm uses $L_3 \triangleright\triangleleft L_3$ to generate a candidate set of 4-itemsets, C_4. Al-though the join results in {{I1, I2, I3, I5}}, itemset {I1, I2, I3, I5} is pruned because its subset {I2, I3, I5} is not frequent. Thus, $C_4 = \varnothing$, and the algorithm terminates, having found all of the frequent itemsets.

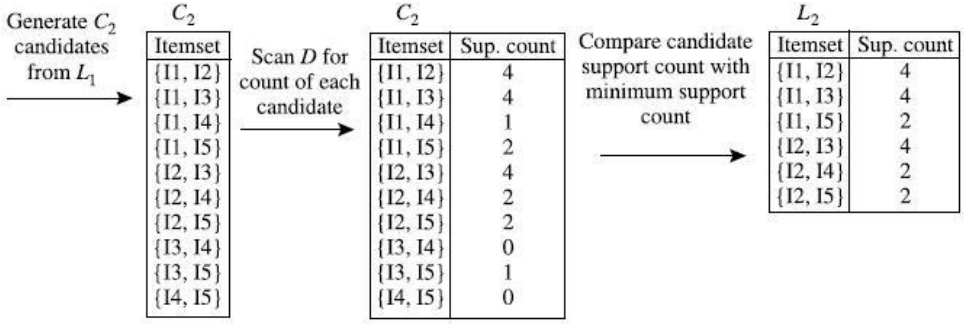
The entire illustration of the Apriori algorithm is displayed in Figures 3-5.



C_1 Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1 Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Figure 3: Initial step and creation of L_1 from C_1 generated candidate set.

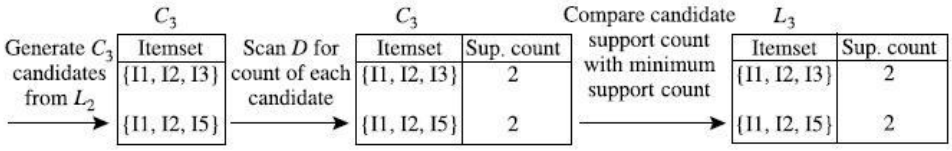


C_2 Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2 Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2 Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

Figure 4: Joining of L_1 to create C_2 and then pruning by min_sup = 2 to create L_2.



C_3 Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3 Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3 Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 5: Joining of L_2 to make huge C_3; pruning this using Apriori property; finally scanning database to get finalized frequent set of itemsets L_3; Since $C_4 = \varnothing$, hence algorithm concludes here.

3 Algorithm

Algorithm 1: Apriori algorithm for discovering frequent itemsets

Input: D, a transactional database and min_sup count
Output: L, the frequent itemsets in D

```
START
Step 1: Initialize L = empty set of itemsets
Step 2: L1 = frequent 1-itemsets(D)
Step 3: while Lk−1 = ∅ do
go to Step 5 in Algorithm 2 for Ck and return here while
each transaction t ∈ D do
    Ct = subset(Ct, t)
    while for each candidate c ∈ Ct do
        c.count++
    end
end
    Lk =c ∈ Ck|c.count ≥ min_sup
    k++
end
Step 4: algorithm ends here; return L = L ∪ Lk
END
```

Algorithm 2: Main joining and pruning step of the Apriori algorithm

Input: L_{k-1}: frequent itemsets of (k-1)-length
Output: joined and pruned C_k

```
START
Step 5: input is Lk−1
Step 6: for each itemset l1 ∈ Lk−1 do
    for each itemset l2 ∈ Lk−1 do
        if all items of l1 and l2 are equal upto (k-2) and (k-1)th item of l1 < (k-1)th item of l2 then
            c = l1 ▷◁ l2
            if c has infrequent subsets in Lk−1 then
                delete c
            else
                add c to Ck
            end
        end
    end
end
Step 7: return generated candidate set, Ck to line 5 of Step 3 of Algorithm 1
END
```

4 Implementation in python

The following implementation is for an illustration of real-world datasets. A sample Chess dataset is given as the input. Various min_sup threshold can be given as input to this implementation. In addition, this implementation can print either all the frequent itemsets, or just the total count of frequent itemsets of a certain k-length or both. The real-world datasets will be provided by your instructor in class.

Step 1: Import the required libraries

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
```

Step 2: Load and explore the data

```
data1 = pnd.read_excel('Online_Retail.xlsx')
data1.head()
# here, we will explore the columns of the data
data1.columns
# Now, we will explore the different regions of transactions
data1.Country.unique()
```

Step 3: Clean the Data

```
# here, we will strip the extra spaces in the description
data1['Description'] = data1['Description'].str.strip()

# Now, drop the rows which does not have any invoice number
data1.dropna(axis = 0, subset = ['InvoiceNo'], inplace = True)
data1['InvoiceNo'] = data1['InvoiceNo'].astype('str')

# Now, we will drop all transactions which were done on credit
data1 = data1[~data1['InvoiceNo'].str.contains('C')]
```

Step 4: Split the data according to the region of transaction

```
# Transactions done in France
basket1_France = (data1[data1['Country'] == "France"]
        .groupby(['InvoiceNo', 'Description'])['Quantity']
        .sum().unstack().reset_index().fillna(0)
        .set_index('InvoiceNo'))

# Transactions done in the United Kingdom
basket1_UK = (data1[data1['Country'] == "United Kingdom"]
        .groupby(['InvoiceNo', 'Description'])['Quantity']
        .sum().unstack().reset_index().fillna(0)
        .set_index('InvoiceNo'))

# Transactions done in Portugal
basket1_Por = (data1[data1['Country'] == "Portugal"]
        .groupby(['InvoiceNo', 'Description'])['Quantity']
        .sum().unstack().reset_index().fillna(0)
        .set_index('InvoiceNo'))

basket1_Sweden = (data1[data1['Country'] == "Sweden"]
        .groupby(['InvoiceNo', 'Description'])['Quantity']
        .sum().unstack().reset_index().fillna(0)
        .set_index('InvoiceNo'))
```

Step 5: Hot encoding the Data

```
# Here, we will define the hot encoding function
# for making the data suitable
# for the concerned libraries
def hot_encode1(P):
   if(P<= 0):
      return 0
```

```
            if(P>= 1):
                return 1

        # Here, we will encode the datasets
        basket1_encoded = basket1_France.applymap(hot_encode1)
        basket1_France = basket1_encoded

        basket1_encoded = basket1_UK.applymap(hot_encode1)
        basket1_UK = basket1_encoded

        basket1_encoded = basket1_Por.applymap(hot_encode1)
        basket1_Por = basket1_encoded

        basket1_encoded = basket1_Sweden.applymap(hot_encode1)
        basket1_Sweden = basket1_encoded
```

Step 6: Build the models and analyse the results

a) France:

```
        # Build the model
        frq_items1 = AP(basket1_France, min_support = 0.05, use_colnames = True)

        # Collect the inferred rules in a dataframe
        rules1 = AR(frq_items1, metric = "lift", min_threshold = 1)
        rules1 = rules1.sort_values(['confidence', 'lift'], ascending = [False, False])
        print(rules1.head())
```

Output:

antecedents \
45 (JUMBO BAG WOODLAND ANIMALS)
260 (PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ...
272 (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI...
302 (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...
301 (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...
consequents antecedent support consequent support \
45 (POSTAGE) 0.076531 0.765306
260 (POSTAGE) 0.051020 0.765306
272 (POSTAGE) 0.053571 0.765306

302 (SET/6 RED SPOTTY PAPER PLATES) 0.102041s 0.127551
301 (SET/6 RED SPOTTY PAPER CUPS) 0.102041 0.137755
support confidence lift leverage conviction
45 0.076531 1.000 1.306667 0.017961 inf
260 0.051020 1.000 1.306667 0.011974 inf
272 0.053571 1.000 1.306667 0.012573 inf
302 0.099490 0.975 7.644000 0.086474 34.897959
301 0.099490 0.975 7.077778 0.085433 34.489796

From the above output, it can be seen that paper cups, paper and plates are bought together in France. This is because the French has a culture of having a get-together with their friends and family at least once a week. Also, since the French government has banned the use of plastic in the country, people have to purchase paper-based alternatives.

b) United Kingdom:

```
1. frq_items = apriori(basket_UK, min_support = 0.01, use_colnames = True)
2. rules = association_rules(frq_items, metric ="lift", min_threshold = 1)
3. rules = rules.sort_values(['confidence', 'lift'], ascending =[False, False])
4. print(rules.head())
```

Output:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
116	(BEADED CRYSTAL HEART PINK ON STICK)	(DOTCOM POSTAGE)	0.011036	0.037928	0.010768	0.975728	25.725872	0.010349	39.637371
2019	(SUKI SHOULDER BAG, JAM MAKING SET PRINTED)	(DOTCOM POSTAGE)	0.011625	0.037928	0.011196	0.963134	25.393807	0.010755	26.096206
2296	(HERB MARKER THYME, HERB MARKER MINT)	(HERB MARKER ROSEMARY)	0.010714	0.012375	0.010232	0.955000	77.173095	0.010099	21.947227
2302	(HERB MARKER PARSLEY, HERB MARKER ROSEMARY)	(HERB MARKER THYME)	0.011089	0.012321	0.010553	0.951691	77.240055	0.010417	20.444951
2300	(HERB MARKER THYME, HERB MARKER PARSLEY)	(HERB MARKER ROSEMARY)	0.011089	0.012375	0.010553	0.951691	76.905682	0.010416	20.443842

6 Discussion & Conclusion

Based on the focused objective(s) to understand about simple frequent itemset mining using Apriori algorithm, the additional lab exercise will make the students more confident towards the fulfilment of the objectives(s).

7 Lab Task (Please implement yourself and show the output to the instructor)

Implement the Apriori algorithm using various other support thresholds in fo\our other datasets and record.

8 Lab Exercise (Submit as a report)

No lab report is required in this lab experiment. This will be continued in the next lab as lab report.

9 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be deducted if any such copying is detected.