جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم
قســـــــــم الانظمة الطبية الذكية

Lecture: (5)

Apply the Apriori Algorithm and mine multilevel association rules.

Subject: Clinical Data Mining

Level: Four

Lecturer: Dr. Maytham Nabeel Meqdad

Apply the Apriori Algorithm and mine multilevel association rules.

The Apriori Algorithm: Finding Frequent Itemsets Using Candidate Generation

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties, as we shall see following. Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are used to explore (*k*+1)-itemsets. First, the setof 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 *L*1.Next, *L*1 is used to find *L*2, the set of frequent 2-itemsets, which is used to find *L*3, and so on, until no more frequent *k*-itemsets can be found. The finding of each *Lk* 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, presented below, is used to reduce the search space.We will first describe this property, and then show an example illustrating its use.

Apriori property: *All nonempty subsets of a frequent itemset must also be frequent.*

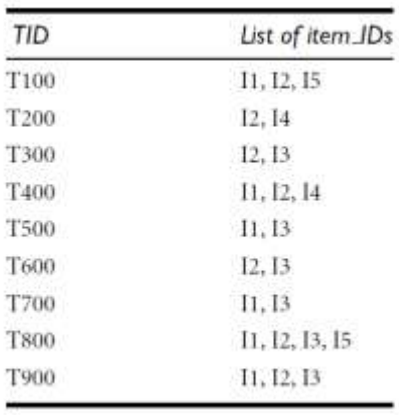
TheApriori property is based on the following observation. By definition, if an itemset *I* does not satisfy the minimum support threshold, *min sup*, then *I* is not frequent; that is, *P*(*I*) < *min sup*. If an item *A* is added to the itemset *I*, then the resulting itemset (i.e., *I* [*A*) cannot occur more frequently than *I*. Therefore, *I* [*A* is not frequent either; that is, *P*(*I* [*A*) < *min sup*.

This property belongs to a special category of properties called antimonotone in the sense that *if a set cannot pass a test, all of its supersets will fail the same test as well*. It is called *antimonotone* because the property is monotonic in the context of failing a test.7 *"How is the Apriori property used in the algorithm?"* To understand this, let us look at how *Lk*−1 is used to find *Lk* for *k* , 2. A two-step process is followed, consisting of join and prune actions.

1. The join step: To find L_k, a set of candidate *k*-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l_1 and l_2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the *j*th item in l_i (e.g., $l_1[k-2]$ refers to the second to the last item in l_1). By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order.
   For the (*k* 1)-itemset, l_i, this means that the items are sorted such that $l_i[1] < l_i[2] < \ldots < l_i[k-1]$.
   The join, L_{k-1} on L_{k-1}, is performed, where members of L_{k-1} are joinable if their first (*k*−2) items are in common. That is, members l_1 and l_2 of L_{k-1} are joined
   if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. The condition $l_1[k-1] < l_2[k-1]$ simply ensures that no duplicates are generated. The resulting itemset formed
by joining l_1 and l_2 is $l_1[1], l_1[2], \ldots, l_1[k-2], l_1[k-1], l_2[k-1]$.
2. The prune step:C_k is a superset of L_k, that is, its members may or may not be frequent, but all of the frequent *k*-itemsets are included inC_k.Ascan of the database to determine the count of each candidate in C_k would result in the determination of L_k (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to L_k). C_k, however, can be huge, and so this could

involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (k−1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k−1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed fromC_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

Table 1 Transactional data for an *All Electronics branch*.
TID List of item IDs

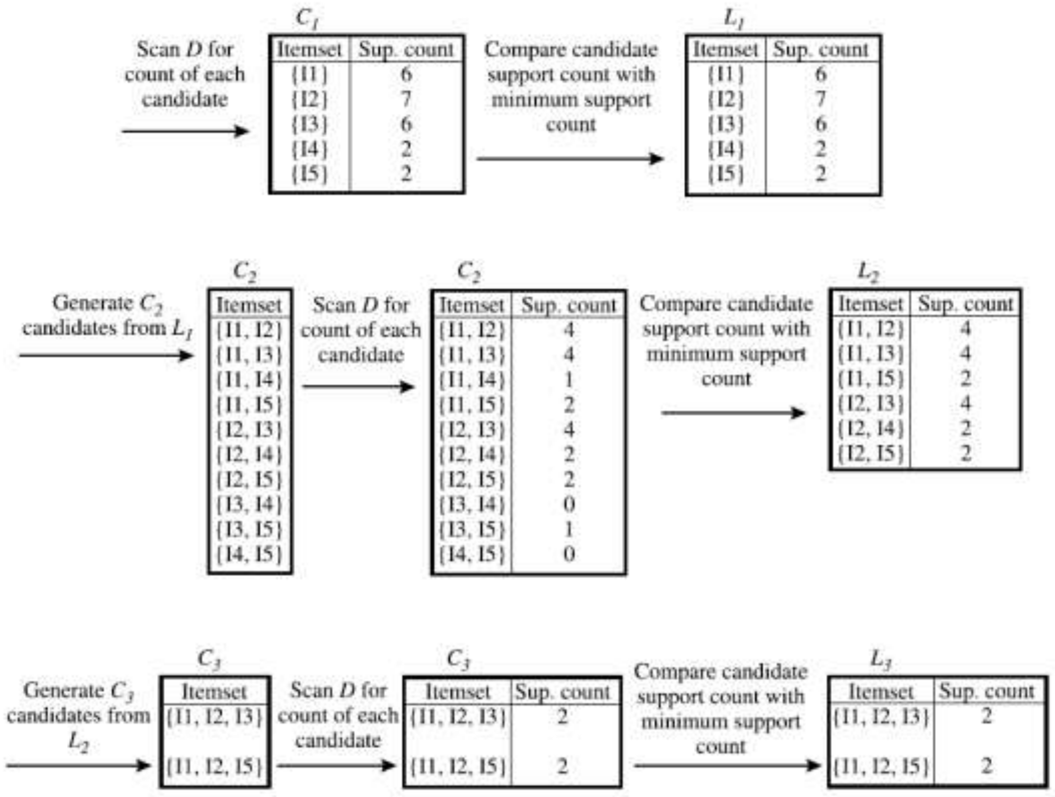


TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

involve heavy computation. To reduce the size of C_k, the Apriori property is used as follows. Any (k−1)-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any (k−1)-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed fromC_k. This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

Example 1 Apriori. Let's look at a concrete example, based on the *AllElectronics* transaction database, *D*, of Table 1. There are nine transactions in this database, that is, $|D| = 9$. We use Figure 5.2 to illustrate the Apriori algorithm for finding frequent itemsets in *D*.

1. 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 in order to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is, *min sup* = 2. (Here, we are referring to *absolute* support because we are using a support count. The corresponding relative support is 2/9 = 22%). 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.
3. To discover the set of frequent 2-itemsets, L_2, the algorithm uses the join L_1 on L_1 to generate a candidate set of 2-itemsets, C_2.[8] C_2 consists of $\binom{|L_1|}{2}$ 2-itemsets. Note that no candidates are removed fromC_2 during the prune step because each subset of the candidates is also frequent.


Scan D for count of each candidate
C_1
Compare candidate support count with minimum support count
L_1
Generate C_2 candidates from L_1
C_2
Scan D for count of each candidate
C_2
Compare candidate support count with minimum support count
L_2
Generate C_3 candidates from L_2
C_3
Scan D for count of each candidate
C_3
Compare candidate support count with minimum support count
L_3

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

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

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

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Algorithm: Apriori

Input:

- D, a database of transactions
- min_sup, the minimum support count threshold

Output:

- L, frequent itemsets in D

Method:

```
1.  L1 = find frequent 1-itemsets(D)
2.  for (k = 2; Lk−1 ≠ ∅; k++) {
       3. Ck = apriori_gen(Lk−1)
       4. for each transaction t ∈ D {
          5. Ct = subset(Ck, t)
          6. for each candidate c ∈ Ct
             7. c.count++
       8. }
       9. Lk = { c ∈ Ck | c.count ≥ min_sup }
3.  }
4.  return L = ∪ Lk
```

Procedure apriori_gen(Lk−1: frequent (k−1)-itemsets)

```
1.  for each itemset l1 ∈ Lk−1
    2.  for each itemset l2 ∈ Lk−1
    3.     if (l1[1] = l2[1]) ∧ (l1[2] = l2[2]) ∧ ... ∧ (l1[k−2] = l2[k−2]) ∧ (l1[k−1] < l2[k−1]) then {
       4.     c = l1 ∪ l2
       5.     if has_infrequent_subset(c, Lk−1) then
       6.        delete c
       7.     else add c to Ck
       8.  }
2.  return Ck
```

Procedure has_infrequent_subset(c: candidate k-itemset, Lk−1: frequent (k−1)-itemsets)

```
1.  for each (k−1)-subset s of c
    2.  if s ∉ Lk−1 then
    3.     return TRUE
2.  return FALSE
```

- **Step by step** to apply the **Apriori algorithm** and mine **multilevel association rules**. I'll first explain the concept, then give an example with calculations.

---

1. Understanding the Apriori Algorithm

The **Apriori algorithm** is used to find **frequent itemsets** and **association rules** from a transaction database. Its steps:

1. **Set minimum support (`min_sup`) and confidence (`min_conf`).**
2. **Find frequent 1-itemsets (`L1`)** – items that appear in at least `min_sup` fraction of transactions.
3. **Generate candidate k-itemsets (`Ck`)** from frequent (k-1)-itemsets (`Lk-1`).
4. **Prune candidates** that have infrequent subsets.
5. **Count support** of candidates in the database and generate frequent k-itemsets (`Lk`).
6. Repeat until no more frequent itemsets are found.
7. **Generate association rules** from frequent itemsets with confidence ≥ `min_conf`.

2. Multilevel Association Rules

Multilevel association rules consider items at **different levels of abstraction**, for example:

- Level 1: `Fruit`
- Level 2: `Apple`, `Banana`
- Level 3: `Red Apple`, `Green Apple`

You can **mine rules at different levels**:

- **Higher levels**: more general rules (e.g., `Fruit → Snack`)
- **Lower levels**: more specific rules (e.g., `Red Apple → Juice`)

To do this:

1. Encode items in a **hierarchical structure**.
2. Apply Apriori **level by level**.
3. Use **different support thresholds**: higher-level rules may need higher support, lower-level rules lower support.

3. Example Dataset

Suppose we have transactions:

TID	Items
1	Red Apple, Banana, Milk
2	Green Apple, Banana, Milk
3	Red Apple, Milk
4	Banana, Milk
5	Green Apple, Banana

Step 1: Frequent 1-itemsets

Assume `min_sup = 60%` → support count ≥ 3 (out of 5 transactions)

Item	Count	Frequent
Red Apple	2	No
Green Apple	2	No
Apple (R+G)	4	Yes
Banana	4	Yes
Milk	4	Yes

→ L1 = {Apple, Banana, Milk}

Step 2: Frequent 2-itemsets

Candidates: {Apple, Banana}, {Apple, Milk}, {Banana, Milk}

Count support:

Itemset	Count	Frequent
Apple, Banana	3	Yes
Apple, Milk	3	Yes
Banana, Milk	3	Yes

→ L2 = {Apple, Banana}, {Apple, Milk}, {Banana, Milk}

Step 3: Frequent 3-itemsets

Candidate: {Apple, Banana, Milk}

Count support:

Itemset	Count	Frequent
Apple, Banana, Milk	2	No

→ L3 = ∅

Step 4: Association Rules

From L2:

1. {Apple} → {Banana}, confidence = support({Apple, Banana}) / support({Apple}) = 3/4 = 75%
2. {Banana} → {Apple}, confidence = 3/4 = 75%
3. {Apple} → {Milk}, confidence = 3/4 = 75%
4. {Milk} → {Apple}, confidence = 3/4 = 75%
5. {Banana} → {Milk}, confidence = 3/4 = 75%
6. {Milk} → {Banana}, confidence = 3/4 = 75%

All rules with confidence ≥ 70% are **strong rules**.

4. Multilevel Mining

- Level 1: `Fruit → Dairy`
  - Support(Fruit) = 4, Support(Dairy) = 4
  - Rule: Fruit → Dairy, confidence = 4/4 = 100% □
- Level 2: {Apple, Banana} → Milk
  - As above, support = 3/5 = 60%, confidence = 75% □
- Level 3: {Red Apple, Banana} → Milk
  - Count support = 1/5 = 20%, confidence = 50% □ (below threshold)

So multilevel rules allow **gradual generalization**, with higher support at higher levels and lower support at lower levels.

Summary

1. Use **Apriori algorithm** to find frequent itemsets **level by level**.
2. Use **hierarchical item encoding** for multilevel rules.
3. Generate **association rules** from frequent itemsets, applying **min_conf** at each level.
4. Higher-level rules tend to have **higher support**, lower-level rules **more specific but lower support**.

==

Example: Apriori Algorithm

Transaction Database (D)

Transaction	Items
T1	Laptop, Mouse
T2	Laptop, Printer
T3	Mouse, Keyboard
T4	Laptop, Mouse
T5	Laptop, Keyboard, Mouse
T6	Printer, Keyboard
T7	Laptop, Keyboard
T8	Mouse, Printer
T9	Laptop, Mouse, Keyboard

Number of transactions = 9 → |D| = 9

Minimum support count = 2 (absolute support) → relative support = 2/9 ≈ 22%

Step 1: Find Frequent 1-itemsets (L1)

- Candidate 1-itemsets:
  `C1 = {Laptop, Mouse, Keyboard, Printer}`
- Count support in D:
  - Laptop = 6
  - Mouse = 6
  - Keyboard = 5
  - Printer = 3
- All ≥ min sup → **L1 = {Laptop, Mouse, Keyboard, Printer}** □

Step 2: Generate Candidate 2-itemsets (C2) using Join

- Join L1 with itself:

C2 = {Laptop+Mouse, Laptop+Keyboard, Laptop+Printer, Mouse+Keyboard, Mouse+Printer, Keyboard+Printer}

- **Prune step**: check all subsets of size 1 (all in L1, so nothing is removed)

- Count support for C2:

  - Laptop+Mouse = 4
  - Laptop+Keyboard = 3
  - Laptop+Printer = 2
  - Mouse+Keyboard = 3
  - Mouse+Printer = 2
  - Keyboard+Printer = 1 (< min sup) → remove

- **L2 = {Laptop+Mouse, Laptop+Keyboard, Laptop+Printer, Mouse+Keyboard, Mouse+Printer}** □

Step 3: Generate Candidate 3-itemsets (C3) using Join

- Join L2 with itself (only compatible pairs):

C3 = {Laptop+Mouse+Keyboard, Laptop+Mouse+Printer, Laptop+Keyboard+Printer, Mouse+Keyboard+Printer}

- **Prune step**: remove candidates if any 2-item subset is **not frequent (not in L2)**

  - Laptop+Keyboard+Printer → subset Laptop+Printer □, Laptop+Keyboard □, Keyboard+Printer □ → remove
  - Mouse+Keyboard+Printer → subset Mouse+Printer □, Mouse+Keyboard □, Keyboard+Printer □ → remove
  - Laptop+Mouse+Printer → all subsets in L2 □ → keep
  - Laptop+Mouse+Keyboard → all subsets in L2 □ → keep

- Count support for remaining candidates:

  - Laptop+Mouse+Keyboard = 2 → frequent
  - Laptop+Mouse+Printer = 1 → < min sup → remove

- **L3 = {Laptop+Mouse+Keyboard}** □

Step 4: Stop

- No candidates for 4-itemsets can be frequent (because only 3-itemset is left)
- **Final Frequent Itemsets:**

Level	Frequent Itemsets (Lk)
L1	Laptop, Mouse, Keyboard, Printer
L2	Laptop+Mouse, Laptop+Keyboard, Laptop+Printer, Mouse+Keyboard, Mouse+Printer
L3	Laptop+Mouse+Keyboard

Summary

- **Join** step: combine frequent (k-1)-itemsets to generate k-itemset candidates
- **Prune** step: remove candidates if any subset is **not frequent** (Apriori property)
- **Scan database** to count support and select frequent itemsets

```
# Online Python - IDE, Editor, Compiler, Interpreter

# Apriori Algorithm in Python
from itertools import combinations

def apriori(transactions, min_support):
    """
    transactions: list of transactions (each transaction is a list of items)
    min_support: minimum support count (absolute)
    """

    # Step 1: Find frequent 1-itemsets (L1)
    item_counts = {}
    for transaction in transactions:
        for item in transaction:
            item_counts[item] = item_counts.get(item, 0) + 1

    # Keep items that satisfy min_support
    L1 = {frozenset([item]): count for item, count in item_counts.items() if count >= min_support}
    print("L1:", L1)

    # Initialize variables
    Lk = L1
    k = 2
    frequent_itemsets = dict(L1)  # Store all frequent itemsets

    while Lk:
        # Step 2: Generate candidate k-itemsets (Ck) using Join
        items = list(Lk.keys())
        Ck = {}

        # Join step: combine itemsets to create candidates
        for i in range(len(items)):
            for j in range(i+1, len(items)):
                union_set = items[i] | items[j]
                if len(union_set) == k:
                    Ck[union_set] = 0

        # Step 3: Count support for each candidate in Ck
        for transaction in transactions:
            transaction_set = set(transaction)
            for candidate in Ck:
                if candidate.issubset(transaction_set):
                    Ck[candidate] += 1

        # Step 4: Prune candidates that do not meet min_support
        Lk = {itemset: count for itemset, count in Ck.items() if count >= min_support}

        # Add Lk to frequent itemsets
        frequent_itemsets.update(Lk)
        if Lk:
            print(f"L{k}:", Lk)

        k += 1

    return frequent_itemsets

# Example usage
transactions = [
    ['Laptop', 'Mouse'],
    ['Laptop', 'Printer'],
    ['Mouse', 'Keyboard'],
    ['Laptop', 'Mouse'],
    ['Laptop', 'Keyboard', 'Mouse'],
    ['Printer', 'Keyboard'],
    ['Laptop', 'Keyboard'],
    ['Mouse', 'Printer'],
    ['Laptop', 'Mouse', 'Keyboard']
]

min_support = 2

frequent_itemsets = apriori(transactions, min_support)
print("\nAll Frequent Itemsets:")
for itemset, count in frequent_itemsets.items():
    print(set(itemset), ":", count)
```

References

[1] Han and M. Kamber, " Data Mining Tools and Technique s", Morgan Kaufmann Publishers. Page 237

[2] .M.H. Dunham, " Data Mining Introductory and Adv anced Topics", Pear son Education