Association Analysis: Apriori Algorithm

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Scenario

- The grocery store wants to get as much money from customers
- Items commonly purchased together: customers' purchasing behavior
  - This knowledge can be used for pricing, marketing promotions, inventory management,…
- "Looking for hidden relationships in large datasets"

Association Analysis

- Also called association rule learning
- Finding different combinations of items
  - Permutation problem?
  - Time-consuming task, eat a lot of computing power
  - Brute-force solutions aren't capable of solving
- A more intelligent approach is required: Apriori

Apriori

Apriori

Pros: Easy to code up

Cons: May be slow on large datasets

Works with: Numeric values, nominal values

- Two forms of interesting relationships
  - Frequent item sets
  - Association rules

Example Transactions

Transaction number	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, orange juice
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, orange juice

Figure 11.1 A simple list of transactions from a natural foods grocery store called Hole Foods

diapers → beer

The most famous example of association analysis is diapers Þ beer. It has been reported that a grocery store chain in the Midwest of the United States noticed that men bought diapers and beer on Thursdays. The store could have profited from this by placing diapers and beer close together and making sure they were full price on Thursdays, but they did not.†

† DSS News, "Ask Dan! What is the true story about data mining, beer and diapers?" http://www.dssresources.com/newsletters/66.php, retrieved March 28, 2011.

Definition of Frequent

- Two most important concepts for definition of "frequent": support and confidence
- Support: the percentage of the dataset that contains this itemset
  - support({diapers, wine}) = P(diapers ∩ wine)
- Confidence: the conditional probability form
  - {diapers} → {wine}:

    P(wine | diapers) = P(diapers ∩ wine)/P(wine)

General Approach

General approach to the Apriori algorithm

1. Collect: Any method.
2. Prepare: Any data type will work as we're storing sets.
3. Analyze: Any method.
4. Train: Use the Apriori algorithm to find frequent itemsets.
5. Test: Doesn't apply.
6. Use: This will be used to find frequent itemsets and association rules between items.

Enumeration of Itemsets

- N possible items can generate 2^N-1 possible itemsets
  - Even a store selling 100 items can generate $1.26*10^{30}$ possible itemsets
  - What if the stores sell 10,000 or more items?
    - That's common!

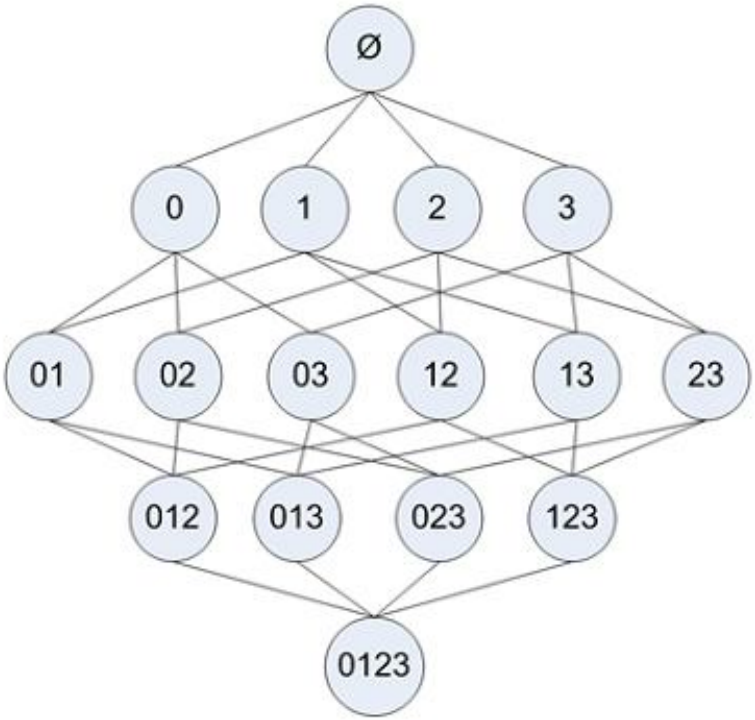

Figure 11.2 All possible itemsets from the available set {0, 1, 2, 3}

Apriori Principle

- If an itemset is frequent, then all of its subsets are frequent

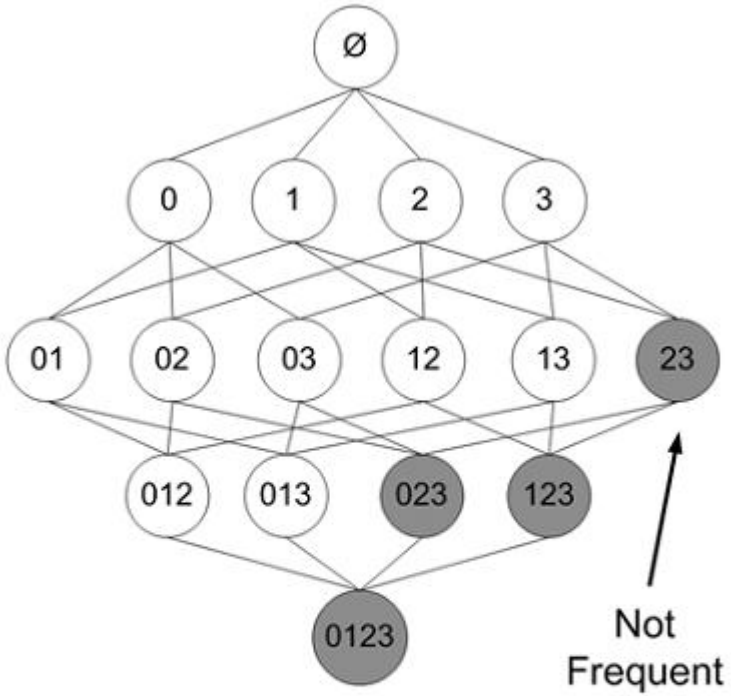

Figure 11.3 All possible itemsets shown, with infrequent itemsets shaded in gray. With the knowledge that the set {2,3} is infrequent, we can deduce that {0,2,3}, {1,2,3}, and {0,1,2,3} are also infrequent, and we don't need to compute their support.

Apriori Algorithm

- The way to find frequent itemsets
- Steps
  - Generate a list of all candidate itemsets with one item
  - The transaction data set will then be scanned to see which sets meet the minimum support level, drop those below the level
  - The remaining sets will then be combined make itemsets with longer elements
  - Repeated until all sets are tossed out

Psuedo-Code for Scanning

For each transaction in tran the dataset:
For each candidate itemset, can:
    Check to see if can is a subset of tran
    If so increment the count of can
For each candidate itemset:
If the support meets the minimum, keep this item
Return list of frequent itemsets

Helper Functions

Listing 11.1 Apriori algorithm helper functions

```
def loadDataSet():
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]

def createC1(dataSet):
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:
                C1.append([item])
    C1.sort()
    return map(frozenset, C1)

def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck:
            if can.issubset(tid):
                if not ssCnt.has_key(can): ssCnt[can]=1
                else: ssCnt[can] += 1
    numItems = float(len(D))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key]/numItems
        if support >= minSupport:
            retList.insert(0,key)
        supportData[key] = support
    return retList, supportData
```

1 Create a frozenset of each item in C1

2 Calculate support for every itemset

Psuedo-Code for Apriori

Listing 11.2 The Apriori algorithm

```
def aprioriGen(Lk, k): #creates Ck
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i])[:k-2]; L2 = list(Lk[j])[:k-2]
            L1.sort(); L2.sort()
            if L1==L2:
                retList.append(Lk[i] | Lk[j])
    return retList

def apriori(dataSet, minSupport = 0.5):
    C1 = createC1(dataSet)
    D = map(set, dataSet)
    L1, supportData = scanD(D, C1, minSupport)
    L = [L1]
    k = 2
    while (len(L[k-2]) > 0):
        Ck = aprioriGen(L[k-2], k)
        Lk, supK = scanD(D, Ck, minSupport)
        supportData.update(supK)
        L.append(Lk)
        k += 1
    return L, supportData
```

1 Join sets if first k-2 items are equal

2 Scan data set to get Lk from Ck

Association Rules, How?

- Example frequent itemset: {soy milk, lettuce}
- Example association rule: soy milk → lettuce
  - Form: *antecedent* → *consequent*
- How to generate association rules from a frequent itemset?

Confidence of Association Rules

- The confidence for a rule $P \rightarrow H$ is defined as

$$confidence(P \rightarrow H) = support(P \mid H)/support(P)$$

$$= support(P \cup H)/support(P)$$

- Use minimum confidence threshold to filter

Enumeration of Possible Rules

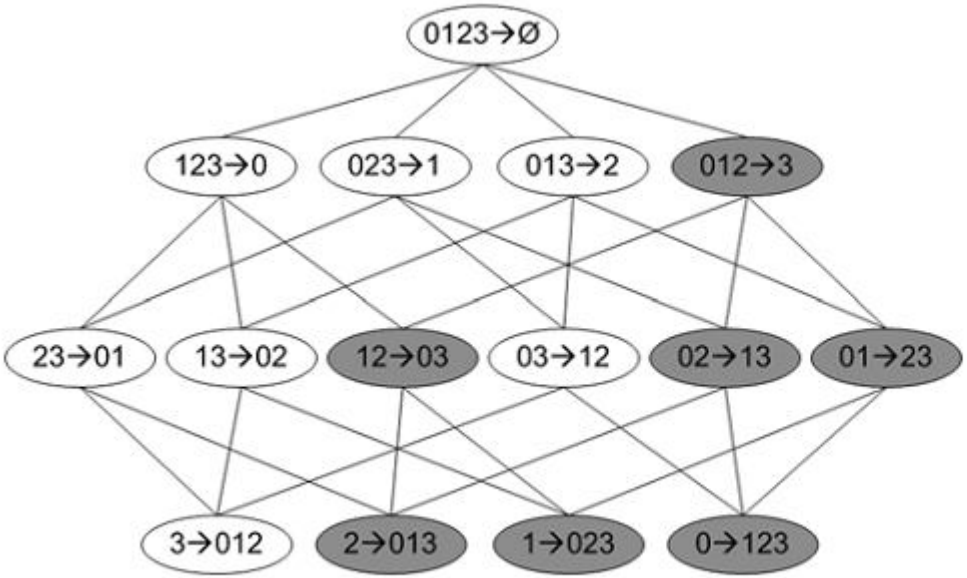

Figure 11.4 Association rule lattice for the frequent itemset {0,1,2,3}. The gray area shows rules with a low confidence. If we find that 0,1,2 → 3 is a low confidence rule, then all other rules with 3 in the consequent (shaded) will also have a low confidence.

Generation of Association Rules

Listing 11.3 Association rule-generation functions

```
def generateRules(L, supportData, minConf=0.7):
    bigRuleList = []
    for i in range(1, len(L)):
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList,\
                                minConf)
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf)
    return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = []
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq]
        if conf >= minConf:
            print freqSet-conseq,'-->',conseq,'conf:',conf
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])
    if (len(freqSet) > (m + 1)):
        Hmp1 = aprioriGen(H, m + 1)
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1):
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)
```

1 Get only sets with two or more items

2 Try further merging

3 Create Hm+1 new candidates

Summary

- Association analysis is used to find interesting relationships in a large set of data
  - Avoid brute-force enumeration of itemsets and rules
- Apriori principle: if an item is infrequent, its supersets will also be infrequent
- Support is used to measure frequent itemsets
- Confidence is used to measure association rules
- The Apriori algorithm scans over the dataset
  - What if the datasets become very large?