732A61/TDDD41 Data Mining - Clustering and Association Analysis

Lecture 6: Association Analysis I

Jose M. Peña
IDA, Linköping University, Sweden

Outline

- Content
  - Association Rules
  - Frequent Itemsets
  - Apriori Algorithm
  - Exercise
  - Rule Generation Algorithm
  - Exercise
  - Summary
- Literature
  - Course book. Second edition: 5.2.1-2, 5.4. Third edition: 6.2.1-2, 6.4.
  - Agrawal, R. and Srikant, R. Fast Algorithms for Mining Association Rules. In Proc. of the 20th Int. Conf. on Very Large Databases, 1994. Extended version available as IBM Research Report RJ9839, 1994.

Association Rules

- Assume that we have access to some transactional data, e.g.

Transaction id	Items bought
1	A, B, D
2	A, C, D
3	A, D, E
4	B, E, F
5	B, C, D, E, F

- We are interested in finding rules of the form

$$Item_1, \ldots, Item_m \rightarrow Item_{m+1}, \ldots, Item_n$$

  meaning that if the items in the antecedent are purchased, so are the items in the consequent, e.g.

$$milk, eggs \rightarrow bread, butter$$

- Application: Market basket analysis to support business decisions, e.g.
  - Rules with “butter” in the consequent may help to decide how to boost sales of “butter”.
  - Rules with “eggs” in the antecedent may help to determine what happens if “eggs” are sold out.
- Note however that the rules do not convey causality, i.e. forcing the antecedent does not guarantee the consequent.

Association Rules

- We are interested in finding rules of the form

$$X_1, \ldots, X_m \to Y_1, \ldots, Y_n \equiv X \to Y$$

- However, not all the rules are equally interesting.
- We are interested in finding rules with user-defined minimum support and confidence, where
  - Support = fraction of the transactions which contain X and $Y = p(X, Y)$.
  - Support = how general the rule is.
  - Confidence = fraction of the transactions that contain X which also contain $Y = p(Y|X)$.
  - Confidence = how accurate the rule is.
  - Confidence = $p(Y|X) = p(X, Y)/p(X)$ = support(X, Y) / support(X).
- Assume the following transactional data.

Transaction id	Items bought
1	A, B, D
2	A, C, D
3	A, D, E
4	B, E, F
5	B, C, D, E, F

- $A \to D$ has support 0.6 and confidence 1.
- $D \to A$ has support 0.6 and confidence 0.75.

Frequent Itemsets

- We are interested in finding rules of the form

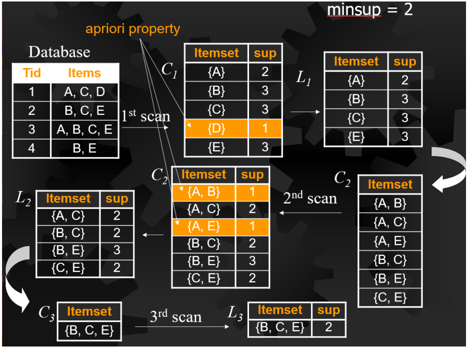
$$X_1, \ldots, X_m \rightarrow Y_1, \ldots, Y_n \equiv X \rightarrow Y$$

  with user-defined minimum support and confidence.
- We define a frequent or large itemset as a set of items that has minimum support.
  - E.g., $\{A, D\}$ is a frequent itemset in the previous example when the minimum support is 0.5.
- We will find the desired rules in two steps:
  1. Find all the frequent itemsets (via the apriori or FP grow algorithm).
  2. Generate all the rules with minimum confidence from the frequent itemsets.
- The first step above will make use of the following apriori property:
  - Every subset of a frequent itemset is frequent.
  - Or, alternatively, every superset of an infrequent itemset is infrequent.

Apriori Algorithm

Algorithm: apriori(D, $minsup$)
Input: A transactional database D and the minimum support $minsup$.
Output: All the large itemsets in D.

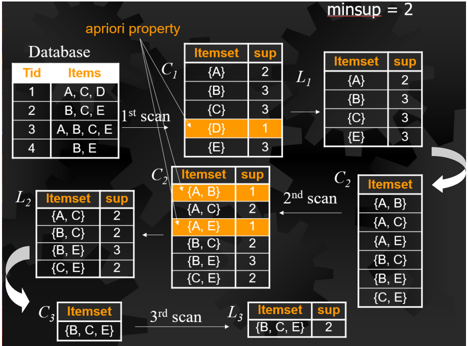
$L_1 = \{$ large 1-itemsets $\}$
for $(k = 2; L_{k-1} \neq \emptyset; k++)$ do
    C_k = apriori-gen(L_{k-1}) // Generate candidate large k-itemsets
    for all $t \in D$ do
        for all $c \in C_k$ such that $c \in t$ do
            $c.count++$
    $L_k = \{c \in C_k | c.count \geq minsup\}$
return $\bigcup_k L_k$



Apriori Algorithm

Algorithm: apriori-gen(L_{k-1})
Input: Large $(k-1)$-itemsets.
Output: A superset of L_k.

```
Ck = ∅                                            // Self-join
for all I, J ∈ L(k-1) do
    if I1 = J1, ..., I(k-2) = J(k-2) and I(k-1) < J(k-1) then
        add {I1, ..., I(k-1), J(k-1)} to Ck
for all c ∈ Ck do                                 // Prune
    for all (k-1)-subsets s of c do
        if s ∉ L(k-1) then
            remove c from Ck
return Ck
```



Apriori Algorithm

- Self-join step in MySQL:

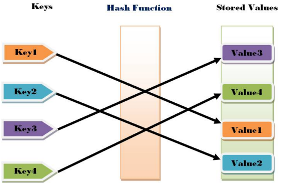
  insert into C_k
  select $I.item_1, \ldots, I.item_{k-1}, J.item_{k-1}$
  from L_{k-1} I, L_{k-1} J
  where $I.item_1 = J.item_1, \ldots, I.item_{k-2} = J.item_{k-2}, I.item_{k-1} < J.item_{k-1}$

- Self-join step in R:

  $$\text{merge}(L_{k-1},\ L_{k-1},\ \text{by=c}(L_{k-1}.item_1, \ldots, L_{k-1}.item_{k-2}))$$

  but note that duplicates will be produced because the condition $I.item_{k-1} < J.item_{k-1}$ is not enforced.

- To make the prune step fast, large itemsets are usually stored in a hash table.



- Clever data structures are also typically used for counting the support of the candidates, i.e. lines 4-6 in the apriori algorithm.

Exercise

- Run the apriori algorithm on the database below with minimum support 0.4, i.e. two transactions.

Tid	A	B	C	D	E
1	1	1	1	0	0
2	1	1	1	1	1
3	1	0	1	1	0
4	1	0	1	1	1
5	1	1	1	1	0

- Show the execution details (i.e. self-join, prune, support counting), not just the large itemsets
  $\{A, B, C, D, E,$
  $AB, AC, AD, AE, BC, BD, CD, CE, DE,$
  $ABC, ABD, ACD, ACE, ADE, BCD, CDE,$
  $ABCD, ACDE\}$.

Apriori Algorithm

Algorithm: apriori-gen(L_{k-1})
Input: Large $(k-1)$-itemsets.
Output: A superset of L_k.

1 $C_k = \varnothing$ // Self-join
2 for all $I, J \in L_{k-1}$ do
3   if $I_1 = J_1, \ldots, I_{k-2} = J_{k-2}$ and $I_{k-1} < J_{k-1}$ then
4   add $\{I_1, \ldots, I_{k-1}, J_{k-1}\}$ to C_k
5 for all $c \in C_k$ do // Prune
6   for all $(k-1)$-subsets s of c do
7   if $s \notin L_{k-1}$ then
8   remove c from C_k
9 return C_k

- We prove by induction on k that the apriori algorithm is correct.
- Trivial case: The algorithm is correct for $k = 1$.
- Induction hypothesis: Assume that the algorithm is correct up to $k-1$. We now prove that the algorithm is correct for k. It suffices to prove that $L_k \subseteq C_k$.
- Assume to the contrary that $I \in L_k$ but $I \notin C_k$. Then,
  - $\{I_1, \ldots, I_{k-2}, I_{k-1}\} \in L_{k-1}$ follows from $I \in L_k$ by the apriori property and the induction hypothesis.
  - $\{I_1, \ldots, I_{k-2}, I_k\} \in L_{k-1}$ follows from $I \in L_k$ by the apriori property and the induction hypothesis.
  - Then, $I \in C_k$ in line 5, i.e. it is generated by the self-join step.
  - Moreover, every subset of I is large by $I \in L_k$ and the apriori property.
  - Then, $I \in C_k$ in line 9, i.e. it is not removed by the prune step.
  - This contradicts our assumption and, thus, the algorithm is correct for k.

Rule Generation Algorithm

- We want to generate all the rules of the form

$$X \to L \setminus X$$

  where L is a large itemset, $X \subseteq L$, and the rule has minimum confidence.
- We will make use of the following apriori property:
  - If X does not result in a rule with minimum confidence for L, neither does any subset of X.

for all large itemsets l_k with $k \geq 2$ do
    call genrules(l_k, l_k, *minconf*)

Algorithm: genrules(l_k, a_m, *minconf*)
Input: A large itemset l_k, a set $a_m \subseteq l_k$, the minimum confidence *minconf*.
Output: All the rules of the form $a \to l_k \setminus a$ with $a \subseteq a_m$ and confidence equal or above *minconf*.

$\mathbb{A} = \{(m-1)\text{-itemsets } a_{m-1} | a_{m-1} \subseteq a_m\}$
for all $a_{m-1} \in \mathbb{A}$ do
    conf = support(l_k) / support(a_{m-1})
    if conf $\geq$ *minconf* then
        output the rule $a_{m-1} \to l_k \setminus a_{m-1}$ with confidence=conf and support=support(l_k)
        if $m-1 > 1$ then call genrules(l_k, a_{m-1}, *minconf*)

- A faster algorithm exists.

Exercise

- Run the genrule algorithm on the database below for the large itemset $\{A, B, C\}$ with minimum confidence 0.8.

Tid	A	B	C	D	E
1	1	1	1	0	0
2	1	1	1	1	1
3	1	0	1	1	0
4	1	0	1	1	1
5	1	1	1	1	0

- Show the execution details (i.e. antecedent generation, recursive calls), not just the rules $\{AB \rightarrow C, BC \rightarrow A, B \rightarrow AC\}$.

Rule Generation Algorithm

for all large itemsets l_k with $k \geq 2$ do
    call genrules(l_k, l_k, *minconf*)

Algorithm: genrules(l_k, a_m, *minconf*)
Input: A large itemset l_k, a set $a_m \subseteq l_k$, the minimum confidence *minconf*.
Output: All the rules of the form $a \rightarrow l_k \setminus a$ with $a \subseteq a_m$ and confidence equal or above *minconf*.

$\mathbb{A} = \{(m-1)\text{-itemsets } a_{m-1} | a_{m-1} \subseteq a_m\}$
for all $a_{m-1} \in \mathbb{A}$ do
    conf = support(l_k) / support(a_{m-1})
    if conf $\geq$ *minconf* then
        output the rule $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ with confidence=conf and support=support(l_k)
        if $m-1 > 1$ then call genrules(l_k, a_{m-1}, *minconf*)

- We prove by contradiction that the rule generation algorithm is correct.
- Assume to the contrary that the algorithm missed a rule. Let $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ denote one of the missing rules with the largest antecedent. Then,
  - Note that l_k has minimum support and, thus, it is outputted by the apriori algorithm since this is correct.
  - Then, the algorithm cannot have missed the rule if $m = k$.
  - Moreover if $m < k$, then
    confidence($a_m \rightarrow l_k \setminus a_m$) = support($l_k$) / support($a_m$)
    $\geq$ support(l_k) / support(a_{m-1}) = confidence($a_{m-1} \rightarrow l_k \setminus a_{m-1}$)
    $\geq$ *minconf*
  - Note that the algorithm did not miss the rule $a_m \rightarrow l_k \setminus a_m$ because, otherwise, it would contradict our assumption.
  - Then, the algorithm cannot have missed the rule $a_{m-1} \rightarrow l_k \setminus a_{m-1}$.
  - This contradicts our assumption and, thus, the algorithm is correct.

Summary

- Mining transactions to find rules of the form

$$Item_1, \ldots, Item_m \rightarrow Item_{m+1}, \ldots, Item_n$$

  with user-defined minimum support and confidence.
- Two-step solution:
  1. Find all the large itemsets (via the apriori algorithm).
  2. Generate all the rules with minimum confidence from the large itemsets.
- The two steps above make use of apriori properties.
- Drawbacks of the apriori algorithm:
  - Candidate generate-and-test.
  - Too many candidates to generate, e.g. if there are 10^4 large 1-itemsets, then more than 10^7 candidate 2-itemsets.
  - Each candidate implies expensive operations, e.g. pattern matching, subset checking, storing.
- Can candidate generation be avoided ? Yes, use the FP grow algorithm.