LECTURE NOTES
732A61/TDDD41
DATA MINING - CLUSTERING AND ASSOCIATION ANALYSIS

JOSE M. PEÑA
IDA, LINKÖPING UNIVERSITY, SE-58183 LINKÖPING, SWEDEN
JOSE.M.PENA@LIU.SE

1. Correctness of the Apriori algorithm

The proof of correctness is not unique. You can find one proof in the article by Agrawal and Srikant available from the course website. Our own alternative proof can be found below.

We prove by induction on k that the apriori algorithm is correct. That is, we prove the result for $k = 1$ and then for k under the assumption that the algorithm is correct up to $k-1$. Combining this two facts, we can conclude that the algorithm is correct for any k. First, recall the apriori algorithm.

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

```
1  L_1 = { large 1-itemsets }
2  for (k = 2; L_{k-1} ≠ ∅; k++) do
3     C_k = apriori-gen(L_{k-1})      // Generate candidate large k-itemsets
4     for all t ∈ D do
5        for all c ∈ C_k such that c ∈ t do
6           c.count++
7     L_k = {c ∈ C_k | c.count ≥ minsup}
8  return ⋃_k L_k
```

Trivial case: The algorithm is correct for $k = 1$ by line 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 $L_k \subseteq C_k$ in line 3, because lines 4-7 simply count the frequency of the candidates and, thus, nothing can go wrong there. Recall the apriori-gen function.

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

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

To prove that $L_k \subseteq C_k$, assume to the contrary that $I \in L_k$ but $I \notin C_k$. Then, the itemset $\{I_1, \ldots, I_{k-2}, I_{k-1}\}$ is in L_{k-1} due to the fact that $I \in L_k$ and the apriori property and the induction hypothesis. Likewise, $\{I_1, \ldots, I_{k-2}, I_k\} \in L_{k-1}$. Then, $I \in C_k$ in line 5 of the apriori-gen function, i.e. it is generated by the self-join step. Moreover, every subset of I is large due to the fact that

Date: 20:01, 15/02/17, Lecture6Proofs.tex.

$I \in L_k$ and the apriori property. Then, $I \in C_k$ in line 9, i.e. it is not removed by the prune step. This contradicts the assumption made at the beginning of this paragraph and, thus, the algorithm is correct for k.

2. Correctness of the rule generation algorithm

No proof is given in the article by Agrawal and Srikant. Our own proof follows. First, recall the algorithm.

```
1  for all large itemsets l_k with k ≥ 2 do
2     call genrules(l_k, l_k, minconf)

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

1  𝔸 = {(m − 1)-itemsets a_{m−1} | a_{m−1} ⊆ a_m}
2  for all a_{m−1} ∈ 𝔸 do
3     conf = support(l_k) / support(a_{m−1})        // Confidence of the rule a_{m−1} → l_k \ a_{m−1}
4     if conf ≥ minconf then
5        output the rule a_{m−1} → l_k \ a_{m−1} with confidence=conf and support=support(l_k)
6        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. Note that that we wrongly missed the rule implies that l_k has minimum support and, thus, it is outputted by the apriori algorithm since this is correct, as proven in the previous section. Then, the rule generation algorithm cannot have missed the rule when $m = k$, because $m = k$ only when we called genrules(l_k, l_k, $minconf$), and then the rule is evaluated and outputted in lines 1-5.

Therefore, we must have missed the rule in one of the subsequent calls to genrules, i.e. when $m < k$. Note that then the consequent of the rule has at least two items. Move one item from the consequent of $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ to the antecedent and so create a rule $a_m \rightarrow l_k \setminus a_m$. Evaluate the confidence of the newly created rule:

$$\text{confidence}(a_m \rightarrow l_k \setminus a_m) = \text{support}(l_k) \,/\, \text{support}(a_m) \geq \text{support}(l_k) \,/\, \text{support}(a_{m-1})$$
$$= \text{confidence}(a_{m-1} \rightarrow l_k \setminus a_{m-1}) \geq minconf$$

where the first inequality follows from the fact that $\text{support}(a_m) \leq \text{support}(a_{m-1})$, and the second inequality follows from the assumption that the algorithm wrongly missed $a_{m-1} \rightarrow l_k \setminus a_{m-1}$, which implies that the rules has confidence equal or above the threshold. Therefore, the rule generation algorithm must produce the rule $a_m \rightarrow l_k \setminus a_m$. Note that the algorithm cannot miss this rule because, otherwise, it would contradict our assumption about $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ being one of the missing rules with the largest antecedent. Then, the algorithm cannot have missed the rule $a_{m-1} \rightarrow l_k \setminus a_{m-1}$, because the rule must have been produced and evaluated in lines 1-5, i.e. a_{m-1} must have been produced from a_m in line 1. This contradicts our assumption about the algorithm missing $a_{m-1} \rightarrow l_k \setminus a_{m-1}$ and, thus, the algorithm is correct.