Incremental Mining of Frequent Itemsets

CPS 296.1
Topics in Database Systems

Mining a growing database
- Given: DB, a database of transactions, each containing a set of items
- Find: L(DB), the set of all frequent itemsets
  - A set of items X is frequent if no less than smin% × |DB| transactions contain X
- If we add a set of transaction to the database (i.e., DB ← DB ⊔ ΔDB), what is L(DB ⊔ ΔDB)?
  - Re-computation is not optimal because it ignores the result of mining the old DB

Incorrect approaches
- L(DB ⊔ ΔDB) = L(DB) ∪ L(ΔDB)?
  - X can be frequent in DB, but it can be infrequent in ΔDB and DB ⊔ ΔDB
    - And vice versa: X can be frequent in ΔDB, but it can be infrequent in DB and DB ⊔ ΔDB
      - L(DB) is not monotone
  - L(DB ⊔ ΔDB) = L(DB) ∩ L(ΔDB)?
    - X can be infrequent in DB, but it can be frequent in ΔDB and DB ⊔ ΔDB
      - And vice versa

Facts about negative border
- Observation 1: Every 1-itemset is in either L(DB) or bd(DB)
- Observation 2: recall pass k of Apriori
  - Generate C_k (candidate itemsets of size k) from L_{k-1} (frequent itemsets of size k - 1)
  - Count C_k to determine L_k (⊆ C_k)
  - C_1 - L_1 is the negative border at level k
  - Apriori counts C_k - L_k
  - After mining DB, we know itemsets in both L(DB) and bd(DB), together with their counts
    - Remember such information to help the incremental mining algorithm

First try at an incremental algorithm
- Input: DB, ΔDB, L(DB) and bd(DB) together with their counts in DB
- Output: L(DB ⊔ ΔDB) together with their counts in DB (ε will come back later to this requirement)
- Method
  - Same as Apriori, but
  - When counting C_k, if X ∈ C_k is in L(DB) or bd(DB), do not go through DB because the count of X in DB is already known; simply go through ΔDB
  - We might save a scan over DB (but not ΔDB) if all itemsets in C_k have been counted in DB

Positive and negative border
- Positive border, bd+(DB): maximal frequent itemsets in DB
  - X is in the positive border if X is frequent and no proper superset of X is frequent
    - Example: bd+(DB) = {ab, c}
- Negative border, bd-(DB): minimal infrequent itemsets in DB
  - X is in the negative border if X is infrequent and no proper subset of X is infrequent
    - Example: bd-(DB) = {d, ac, bc}
A problem of the first try

- Each scan over $DB$ may count only a few itemsets \( \rightarrow \) insufficient computation to overlap I/O
  - Also a problem in Apriori
  - But aggravated in the incremental algorithm because some of $C_k$ may have been counted before

\[ \text{In general, a trade-off in level-wise algorithms} \]

- If we count an itemset $X$ in the next level, we risk doing useless work because a subset of $X$ (which we are counting at the same time) may turn out to be infrequent

Another problem (\?\?) (slide 1)

- Did not use the fact that $bd \subseteq DB$ and beyond are infrequent in $DB$
- Example
  - Pass 1
    - Scan of $DB$ is saved
    - Say all 1-itemsets turn out to be frequent in $DB \cup DB$
  - Pass 2
    - Scan of $DB$ is needed because $ad, bd, cd \in C_4$ but they have never been counted in $DB$
    - But at least we know they are infrequent in $DB$; perhaps their counts in $DB \cup DB$ are not high enough to make them frequent in $DB \equiv DB$, so we could have avoided scanning $DB$

Another problem (\?\?) (slide 2)

- If we also care about $bd \subseteq DB \cup DB$ with counts, then we still need to count $ad, bd, cd \in DB$
  - Counts for $bd \subseteq DB \cup DB$
  - Are needed to make the incremental algorithm ready for next $DB$

Observation 1

- If $X$ is infrequent in $DB$, then $X$ can be frequent in $DB \cup DB \cup DB$ only if $X$ is frequent in $DB$
  - Infrequent in both $DB$ and $DB \cup DB \rightarrow$ infrequent in $DB \cup DB$

\[ \text{Strategy implied} \]

- First, mine $DB \cup DB$ to find $L(DB \cup DB)$ and $bd \subseteq DB$ with counts
- When counting $C_k$, if $X \subseteq C_k$ is in $L(DB \cup DB)$ or $bd \subseteq DB$, do not go through $DB \cup DB$ because the count of $X$ in $DB \cup DB$ is already known
- Add the following pruning condition: For any $X \subseteq C_k$, if we already know $X \not\subseteq L(DB)$ and $X \not\subseteq L(DB \cup DB)$, remove $X$ from $C_k$

Observation 2

- If none of the itemsets in $bd \subseteq DB$ becomes frequent in $DB \cup DB$, then no new itemset will be introduced (i.e., $L(DB \cup DB \cup DB) \subseteq L(DB \cup DB)$)
  - Say $X$ is infrequent in $DB$
  - Then there exists $Y \subseteq X$ s.t. $Y \subseteq bd \subseteq DB$
  - Since none of the itemsets in $bd \subseteq DB$ is frequent in $DB \cup DB$, $Y$ is infrequent in $DB \cup DB$
  - That means $X \supseteq Y$ is infrequent in $DB \cup DB$

\[ \text{Strategy implied} \]

- In $DB \cup DB$, count itemsets in $bd \subseteq DB$ to find their counts in $DB \cup DB$
- If none of these itemsets are frequent in $DB \cup DB$, there is no need to scan $DB$ at all

Second try (slide 1)

\[ \text{Strategy implied} \]

- Mine $DB \cup DB$ to obtain $L(DB \cup DB)$ and $bd \subseteq DB$ with counts
- While mining $DB \cup DB$, also count itemsets in $L(DB \cup DB)$ and $bd \subseteq DB$
- For each itemset in $L(DB \cup DB)$ and $bd \subseteq DB$, calculate its count in $DB \cup DB$

(Continue on the next slide)
Second try (slide 2)

(Continued from the previous slide)

- If none of the itemsets in $bd(DB)$ is frequent in $DB \triangle DB$, stop and output itemsets in $L(DB)$ and $bd(DB)$ that are in $L(DB \triangle DB)$ or $bd(DB \triangle DB)$, together with their counts.
- Otherwise, scan $DB$ once
  - Count all itemsets in $C = L(DB) \cup bd(DB) - L(DB) - bd(DB) - \{ X \mid \exists Y \in L(DB) \cup bd(DB) \text{ s.t. } Y \text{ is known to be infrequent in } DB \cup \triangle DB \text{ and } Y \subseteq X \}$
  - Output itemsets in $L(DB)$, $bd(DB)$, and $C$ that are in $L(DB \cup \triangle DB)$ or $bd(DB \cup \triangle DB)$, together with their counts.

Experiments

- Not nearly close to the ideal speed-up
  - Incremental algorithm does not replace Apriori
- Smaller $\triangle DB$ means bigger speed-up (usually)
- Speed-up is lower for very high support threshold
  - Apriori makes very few passes anyway
- Speed-up is lower for very low support threshold
  - Probability of the negative border expanding is higher

First vs. second try

- Second try (Thomas et al.) scans $DB$ at most once
  - May need to count lots of itemsets in the same pass
  - Some of these itemset may not need to be counted
    - Example?
  - Also, complete mining of $\triangle DB$ may be unnecessary
    - Example?
- First try scans $DB$ multiple times (up to the number of scans required by Apriori minus one)
  - Will not scan $DB$ if the second try does not
  - May count very few itemsets in one pass
  - Every itemset counted is necessary
  - Fundamental trade-off in play again!