CompSci 516
Data Intensive Computing Systems

Lecture 25
Data Mining
and
Mining Association Rules

Instructor: Sudeepa Roy
Announcements

• HW5 due tomorrow 04/20 (Wednesday), 11:55 pm

• Additional office hour – Sudeepa - Thursdays – 3 – 4 pm – D325 (until the exam)

• Review session a few days before the final
Reading Material

Optional Reading:

1. [RG]: Chapter 26

2. “Fast Algorithms for Mining Association Rules”
   Agrawal and Srikant, VLDB 1994

19,496 citations on Google Scholar (as of April, 2016 - ~800 increase in eight months)!
One of the most cited papers in CS

• Acknowledgement:
The following slides have been prepared adapting the slides provided by the authors of [RG] and using several presentations of this paper available on the internet (esp. by Ofer Pasternak and Brian Chase)
Data Mining - 1

• Find interesting trends or patterns in large datasets
  – to guide decisions about future activities
  – ideally, with minimal user input
  – the identified patterns should give a data analyst useful and unexpected insights
  – can be explored further with other decision support tools (like data cube)
Data Mining - 2

• Related to
  – exploratory data analysis (Statistics)
  – Knowledge Discovery (KD)
  – Machine Learning

• Scalability is important and a new criterion
  – w.r.t. main memory and CPU

• Additional criteria
  – Noisy and incomplete data (Lecture 24)
  – Iterative process (improve reliability and reduce missing patterns with user inputs)
OLAP vs. Data Mining

• Both analyze and explore data
  – SQL queries (relational algebra)
  – OLAP (multidimensional model)
  – Data mining (most abstract analysis operations)

• Data mining has more flexibility
  – assume complex high level “queries”
  – few parameters are user-definable
  – specialized algorithms are needed
Four Main Steps in KD and DM (KDD)

• Data Selection
  – Identify target subset of data and attributes of interest

• Data Cleaning
  – Remove noise and outliers, unify units, create new fields, use denormalization if needed

• Data Mining
  – extract interesting patterns

• Evaluation
  – present the patterns to the end users in a suitable form, e.g. through visualization
Several DM/KD (Research) Problems

• Discovery of causal rules
• Learning of logical definitions
• Fitting of functions to data
• Clustering
• Classification
• Inferring functional dependencies from data
• Finding “usefulness” or “interestingness” of a rule

– See the citations in the Agarwal-Srikant paper
– Some discussed in [RG] Chapter 27
More: Iceberg Queries

SELECT P.custid, P.item, SUM(P.qty)
FROM Purchases P
GROUP BY P.custid, P.item
HAVING SUM(P.qty) > 5

- Output is much smaller than the original relation or full query answer
- Computing the full answer and post-processing may not be a good idea
- Try to find efficient algorithms with full "recall" and high "precision"

ref. "Computing Iceberg Queries Efficiently"
Fang et al.
VLDB 1998
Our Focus in this Lecture

• Frequent Itemset Counting
• Mining Association Rules
  – using frequent itemsets
  – Both from the Agarwal-Srikant paper

• Many of the “rule-discovery systems” can use the association rule mining ideas
Mining Association Rules

• Retailers can collect and store massive amounts of sales data
  – transaction date and list of items

• Association rules:
  – e.g. 98% customers who purchase “tires” and “auto accessories” also get “automotive services” done
  – Customers who buy mustard and ketchup also buy burgers
  – Goal: find these rules from just transactional data (transaction id + list of items)
Applications

• Can be used for
  – marketing program and strategies
  – cross-marketing
  – catalog design
  – add-on sales
  – store layout
  – customer segmentation
Notations

- Items $I = \{i_1, i_2, \ldots, i_m\}$
- $D$: a set of transactions
- Each transaction $T \subseteq I$
  - has an identifier $TID$
- Association Rule
  - $X \Rightarrow Y$
  - $X, Y \subset I$
  - $X \cap Y = \emptyset$
Confidence and Support

- **Association rule** $X \rightarrow Y$

- **Confidence** $c = \frac{|\text{Tr. with } X \text{ and } Y|}{|\text{Tr. with } X|} \times 100$
  - $c\%$ of transactions in $D$ that contain $X$ also contain $Y$

- **Support** $s = \frac{|\text{Tr. with } X \text{ and } Y|}{|\text{all Tr.}|} \times 100$
  - $s\%$ of transactions in $D$ contain $X$ and $Y$. 
## Support Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Cereal</th>
<th>Beer</th>
<th>Bread</th>
<th>Bananas</th>
<th>Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- **Support(Cereal)**
  - $4/8 = .5$
- **Support(Cereal → Milk)**
  - $3/8 = .375$
# Confidence Example

<table>
<thead>
<tr>
<th>TID</th>
<th>Cereal</th>
<th>Beer</th>
<th>Bread</th>
<th>Bananas</th>
<th>Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- Confidence(Cereal $\rightarrow$ Milk)
  - $3/4 = 0.75$
- Confidence(Bananas $\rightarrow$ Bread)
  - $1/3 = 0.33333\ldots$
X $\rightarrow$ Y is not a Functional Dependency

For functional dependencies

• F.D. = two tuples with the same value of of X must have the same value of Y
  – $X \rightarrow Y \Rightarrow XZ \rightarrow Y$ (concatenation)
  – $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$ (transitivity)

For association rules

• $X \rightarrow A$ does not mean $XY \rightarrow A$
  – May not have the minimum support
  – Assume one transaction {AX}

• $X \rightarrow A$ and $A \rightarrow Z$ do not mean $X \rightarrow Z$
  – May not have the minimum confidence
  – Assume two transactions {XA}, {AZ}
Problem Definition

• **Input**
  – a set of transactions D
    • Can be in any form – a file, relational table, etc.
  – min support (minsup)
  – min confidence (minconf)

• **Goal: generate all association rules that have**
  – support >= minsup and
  – confidence >= minconf
Decomposition into two subproblems

• **1. Apriori and AprioriTID:**
  – for finding “large” itemsets with support $\geq \text{minsup}$
  – all other itemsets are “small”

• **2. Then use another algorithm to find rules $X \rightarrow Y$ such that**
  – Both itemsets $X \cup Y$ and $X$ are large
  – $X \rightarrow Y$ has confidence $\geq \text{minconf}$

• **Paper focuses on subproblem 1**
  – if support is low, confidence may not say much
  – subproblem 2 in full version
Basic Ideas - 1

• Q. Which itemset can possibly have larger support: ABCD or AB
  – i.e. when one is a subset of the other?

• Ans: AB
  – any subset of a large itemset must be large
  – So if AB is small, no need to investigate ABC, ABCD etc.
Basic Ideas - 2

• Start with individual (singleton) items \{A\}, \{B\}, ...
• In subsequent passes, extend the “large itemsets” of the previous pass as “seed”
• Generate new potentially large itemsets (candidate itemsets)
• Then count their actual support from the data
• At the end of the pass, determine which of the candidate itemsets are actually large
  – becomes seed for the next pass
• Continue until no new large itemsets are found

• Benefit: candidate itemsets are generated using the previous pass, without looking at the transactions in the database
  – Much smaller number of candidate itemsets are generated
Apriori vs. AprioriTID

• Both follow the basic ideas in the previous slides

• AprioriTID has the additional property that the database is not used at all for counting the support of candidate itemsets after the first pass
  – An “encoding” of the itemsets used in the previous pass is employed
  – Size of the encoding becomes smaller in subsequent passes – saves reading efforts

• More later
### Notations

- Assume the database is of the form `<TID, i1, i2, ...>` where items are stored in lexicographic order.
- TID = identifier of the transaction.
- Also works when the database is “normalized”: each database record is `<TID, item>` pair.

<table>
<thead>
<tr>
<th><strong>k-itemset</strong></th>
<th>An itemset having <em>k</em> items.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>L_k</em></td>
<td>Set of large <em>k</em>-itemsets (those with minimum support). Each member of this set has two fields: i) itemset and ii) support count.</td>
</tr>
<tr>
<td><em>C_k</em></td>
<td>Set of candidate <em>k</em>-itemsets (potentially large itemsets). Each member of this set has two fields: i) itemset and ii) support count.</td>
</tr>
<tr>
<td>$\overline{C}_k$</td>
<td>Set of candidate <em>k</em>-itemsets when the TIDs of the generating transactions are kept associated with the candidates.</td>
</tr>
</tbody>
</table>

**ACTUAL**

**POTENTIAL**

Used in both Apriori and AprioriTID

Used in AprioriTID
Algorithm Apriori

\[ L_1 = \{\text{large 1-itemsets}\} \]

For ( \( k = 2; \ L_{k-1} \neq \emptyset; \ k ++ \) ) do begin
\[ C_k = \text{apriori-gen}(L_{k-1}); \]
for all transactions \( t \in D \) do begin
\[ C_t = \text{subset}(C_k, t); \]
for all candidates \( c \in C_t \) do
\[ c.\text{count}++; \]
end
end
\[ L_k = \{ c \in C_k | c.\text{count} \geq \text{minsup}\} \]
end
Answer = \( \bigcup_k L_k \);
Apriori-Gen

- Takes as argument $L_{k-1}$ (the set of all large $k$-itemsets)
- Returns a superset of the set of all large $k$-itemsets by augmenting $L_{k-1}$

**Join step**

- $L_{k-1} \bowtie L_{k-1}$
- $p$ and $q$ are two large $(k-1)$-itemsets identical in all $k-2$ first items.

**Prune step**

- $p, q$ are two large $(k-1)$-itemsets identical in all $k-2$ first items.
- Check all the subsets, remove all candidate with some “small” subset

- Join by adding the last item of $q$ to $p$
- for all itemsets $c \in C_k$ do
  - for all $(k-1)$-subsets $s$ of $c$ do
    - if $(s \notin L_{k-1})$ then
      - delete $c$ from $C_k$
Apriori-Gen Example - 1

Step 1: Join \((k = 4)\)

Assume numbers 1-5 correspond to individual items

\(L_3\) \hspace{2cm} \(C_4\)

- \{1,2,3\}
- \{1,2,4\} \rightarrow \{1,2,3,4\}
- \{1,3,4\}
- \{1,3,5\}
- \{2,3,4\}
Apriori-Gen Example - 2

Step 1: Join (k = 4)

Assume numbers 1-5 correspond to individual items

$L_3$

- \{1,2,3\}
- \{1,2,4\}
- \{1,3,4\}
- \{1,3,5\}
- \{2,3,4\}

$C_4$

- \{1,2,3,4\}
- \{1,3,4,5\}
Apriori-Gen Example - 3

Step 2: Prune (k = 4)

- Remove itemsets that can’t have the required support because there is a subset in it which doesn’t have the level of support i.e. not in the previous pass (k-1)

$L_3$

- {1,2,3}
- {1,2,4}
- {1,3,4}
- {1,3,5}
- {2,3,4}

$C_4$

- {1,2,3,4}
- {1,3,4,5}

No {1,4,5} exists in $L_3$

Rules out {1, 3, 4, 5}
Comparisons with previous algorithms (AIS, STEM)

$L_{k-1}$ to $C_k$
- Read each transaction $t$
- Find itemsets $p$ in $L_{k-1}$ that are in $t$
- Extend $p$ with large items in $t$ and occur later in lexicographic order

$L_3$               $C_4$
- $\{1,2,3\}$      - $\{1,2,3,4\}$
- $\{1,2,4\}$      - $\{1,2,3,5\}$
- $\{1,3,4\}$      - $\{1,2,4,5\}$
- $\{1,3,5\}$      - $\{1,3,4,5\}$
- $\{2,3,4\}$      - $\{2,3,4,5\}$

$t = \{1, 2, 3, 4, 5\}$
all 1-5 large items (why?)

5 candidates compared to 2 (after pruning 1) in Apriori
Correctness of Apriori

Show that $C_k \supseteq L_k$

- Any subset of large itemset must also be large
- for each $p$ in $L_k$, it has a subset $q$ in $L_{k-1}$
- We are extending those subsets $q$ in Join with another subset $q'$ of $p$, which must also be large
  - equivalent to extending $L_{k-1}$ with all items and removing those whose $(k-1)$ subsets are not in $L_{k-1}$
- Prune is not deleting anything from $L_k$

insert into $C_k$

select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$

from $L_{k-1}p, L_{k-1}q$

where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$

forall itemsets $c \in C_k$ do

forall $(k-1)$-subsets $s$ of $c$ do

if ($s \notin L_{k-1}$) then

delete $c$ from $C_k$

Check yourself
Variations of Apriori

- Counting candidates of multiple sizes in one pass
- In the k-th pass
  - Not only update counts of $C_k$
  - update counts of candidates $C'_{k+1}$
  - $C'_{k+1} \supseteq C_{k+1}$ since it is generated from $L_k$
  - Can help when the cost of updating and keeping in memory $C'_{k+1} - C_{k+1}$ additional candidates is less than scanning the database
Problem with Apriori

- Every pass goes over the entire dataset
  - Database of transactions is massive
    – Can be millions of transactions added an hour
  - Scanning database is expensive
    – In later passes transactions are likely NOT to contain large itemsets
    – Don’t need to check those transactions

\[
L_1 = \{\text{large 1-itemsets}\} \\
\text{For } (k = 2; L_{k-1} \neq \phi; k++) \text{ do begin} \\
\quad C_k = \text{apriori-gen}(L_{k-1}); \\
\quad \text{forall transactions } t \in D \text{ do begin} \\
\quad \quad C_t = \text{subset}(C_k, t) \\
\quad \quad \text{forall candidates } c \in C_t \text{ do} \\
\quad \quad \quad c.count++; \\
\quad \text{end} \\
\quad \text{end} \\
\quad L_k = \{c \in C_k | c.count \geq \text{minsup}\} \\
\text{end} \\
\text{Answer} = \bigcup_k L_k;
\]
AprioriTid

• Also uses Apriori-Gen
• But scans the database D only once.
• Builds a storage set $C^*_k$
  – “bar” in the paper instead of *
• Members of $C^*_k$ are of the form $< \text{TID}, \{X_k\}>$
  – each $X_k$ is a potentially large $k$-itemset present in the transaction TID
  – For $k=1$, $C^*_1$ is the database D
  – items i as \{i\}
• If a transaction does not have a candidate $k$-itemset, $C^*_k$ will not contain anything for that TID
• $C^*_k$ may be smaller than #transactions, esp. for large values of $k$
  – For smaller values of $k$, it may be large
Algorithm AprioriTid

$L_1 = \{\text{large 1-itemsets}\}$

$C^\wedge_l = \text{database } D$;

For $(k = 2; L_{k-1} \neq \emptyset; k++)$ do begin

\[ C_k = \text{apriori-gen}(L_{k-1}); \]

\[ C^\wedge_k = \emptyset; \]

for all entries $t \in C^\wedge_{k-1}$ do begin

\[ C_t = \{c \in C_k | (c - c[k] \in t.\text{set-of-items} \land (c - c[k-1] \in t.\text{set-of-items})\}; \]

for all candidates $c \in C_t$ do

\[ c.\text{count}++; \]

if ($C_t \neq \emptyset$) then $C^\wedge_k = \langle t.\text{TID}, C_t \rangle$;

end

end

$L_k = \{c \in C_k | c.\text{count} \geq \text{minsup}\}$

end

Answer $= \bigcup_k L_k$;

- Count item occurrences
- The storage set is initialized with the database
- Generate new k-itemsets candidates
- Build a new storage set
- Determine candidate itemsets which are contained in transaction TID
- Find the support of all the candidates
- Remove empty entries
- Take only those with support over minsup

See the examples in the following slides and then come back to the algorithm.
AprioriTid Example

Min support = 2

<table>
<thead>
<tr>
<th>Database</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TID</td>
<td>Items</td>
</tr>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>TID</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>${{1}, {3}, {4}}$</td>
</tr>
<tr>
<td>200</td>
<td>${{2}, {3}, {5}}$</td>
</tr>
<tr>
<td>300</td>
<td>${{1}, {2}, {3}, {5}}$</td>
</tr>
<tr>
<td>400</td>
<td>${{2}, {5}}$</td>
</tr>
</tbody>
</table>

$L_1$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1}$</td>
<td>2</td>
</tr>
<tr>
<td>${2}$</td>
<td>3</td>
</tr>
<tr>
<td>${3}$</td>
<td>3</td>
</tr>
<tr>
<td>${5}$</td>
<td>3</td>
</tr>
</tbody>
</table>
Now we need to compute the supports of $C_2$ without looking at the database $D$ from $C^*_1$
AprioriTid Example

Min support = 2

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td></td>
</tr>
<tr>
<td>{1 5}</td>
<td></td>
</tr>
<tr>
<td>{2 3}</td>
<td></td>
</tr>
<tr>
<td>{2 5}</td>
<td></td>
</tr>
<tr>
<td>{3 5}</td>
<td></td>
</tr>
</tbody>
</table>

\[ C_2 \]

\[ C_1 \]

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1}, {3}, {4} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2}, {3}, {5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1}, {2}, {3}, {5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2}, {5} }</td>
</tr>
</tbody>
</table>

C_{100} = \{\{1, 3\}\}
C_{200} = \{\{2, 3\}, \{2, 5\}, \{3, 5\}\}
C_{300} = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 5\}\}
C_{400} = \{\{2, 5\}\}

Only 300 has both \{1\} and \{2\}
Support = 1
AprioriTid Example

Min support = 2

$k = 2$

### Candidate Itemsets

<table>
<thead>
<tr>
<th>$C_2$</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>1</td>
</tr>
<tr>
<td>{2 5}</td>
<td>1</td>
</tr>
<tr>
<td>{3 5}</td>
<td>1</td>
</tr>
</tbody>
</table>

### Frequent Itemsets

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1}, {3}, {4} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2}, {3}, {5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1}, {2}, {3}, {5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2}, {5} }</td>
</tr>
</tbody>
</table>

```python
for all entries $t \in \overline{C}_{k-1}$ do begin
  // determine candidate itemsets in $C_k$ contained
  // in the transaction with identifier $t$.TID
  $C_t = \{ c \in C_k \mid (c - c[k]) \in t.set-of-itemsets \wedge$
  $(c - c[k-1]) \in t.set-of-itemsets\};$
  for all candidates $c \in C_t$ do
    $c.count++;$
    if ($C_t \neq \emptyset$) then $\overline{C}_k += <t.TID, C_t>;$
  end
end
```

\[ C_{100} = \{\{1, 3\}\} \]
\[ C_{200} = \{\{2, 3\}, \{2, 5\}, \{3, 5\}\} \]
\[ C_{300} = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 5\}\} \]
\[ C_{400} = \{\{2, 5\}\} \]
AprioriTid Example

Min support = 2

$k = 2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

$C_2$

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{ {1}, {3}, {4} }</td>
</tr>
<tr>
<td>200</td>
<td>{ {2}, {3}, {5} }</td>
</tr>
<tr>
<td>300</td>
<td>{ {1}, {2}, {3}, {5} }</td>
</tr>
<tr>
<td>400</td>
<td>{ {2}, {5} }</td>
</tr>
</tbody>
</table>

$C_1$

\[ C_{100} = \{1, 3\} \]
\[ C_{200} = \{2, 3\}, \{2, 5\}, \{3, 5\} \]
\[ C_{300} = \{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 5\} \]
\[ C_{400} = \{2, 5\} \]

forall entries $t \in C_{k-1}$ do begin
  // determine candidate itemsets in $C_k$ contained in the transaction with identifier $t.TID$
  $C_t = \{ c \in C_k \mid (c - c[k]) \in t.set-of-itemsets \land (c - c[k-1]) \in t.set-of-itemsets \}$;
  forall candidates $c \in C_t$ do
    $c.count++$;
    if ($C_t \neq \emptyset$) then $C_k += <t.TID, C_t>$;
  end
AprioriTid Example

Min support = 2

How $C^*_2$ looks

$k = 2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>${1 \ 2}$</td>
<td>1</td>
</tr>
<tr>
<td>${1 \ 3}$</td>
<td>2</td>
</tr>
<tr>
<td>${1 \ 5}$</td>
<td>1</td>
</tr>
<tr>
<td>${2 \ 3}$</td>
<td>2</td>
</tr>
<tr>
<td>${2 \ 5}$</td>
<td>3</td>
</tr>
<tr>
<td>${3 \ 5}$</td>
<td>2</td>
</tr>
</tbody>
</table>

$C_2$

$C_{100} = \{\{1, 3\}\}$
$C_{200} = \{\{2, 3\}, \{2, 5\}, \{3, 5\}\}$
$C_{300} = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 5\}\}$
$C_{400} = \{\{2, 5\}\}$

$\overline{C}_2$

<table>
<thead>
<tr>
<th>TID</th>
<th>Set-of-Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>${{1, 3}}$</td>
</tr>
<tr>
<td>200</td>
<td>${{2, 3}, {2, 5}, {3, 5}}$</td>
</tr>
<tr>
<td>300</td>
<td>${{1, 2}, {1, 3}, {1, 5}, {2, 3}, {2, 5}, {3, 5}}$</td>
</tr>
<tr>
<td>400</td>
<td>${{2, 5}}$</td>
</tr>
</tbody>
</table>

forall entries $t \in \overline{C}_{k-1}$ do begin

// determine candidate itemsets in $C_k$ contained
// in the transaction with identifier $t$.TID
$C_t = \{c \in C_k \mid (c - c[k]) \in t.set-of-itemsets \land\ (c - c[k-1]) \in t.set-of-itemsets\}$;

forall candidates $c \in C_t$ do

c.count++; //

if ($C_t \neq \emptyset$) then $\overline{C}_k += <t.TID, C_t>$;

end
AprioriTid Example

Min support = 2

The supports are in place
Can compute $L_2$ from $C_2$

How $L_2$ looks
(entries above threshold)

$C_2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

$L_2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>
AprioriTid Example

Min support = 2

Next step

$L_2$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

$C_3$

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td></td>
</tr>
</tbody>
</table>
AprioriTid Example

Min support = 2

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2,3,5}</td>
<td></td>
</tr>
</tbody>
</table>

Look for transactions containing \{2, 3\} and \{2, 5\}

Add <200, \{2,3,5\}> and <300, \{2,3,5\}> to \(C^*_3\)

```
forall entries t ∈ \(\overline{C}_{k-1}\) do begin
  // determine candidate itemsets in \(C_k\) contained
  // in the transaction with identifier t.TID
  \(C_t = \{c ∈ C_k \mid (c - c[k]) ∈ t.set-of-itemsets ∧ \)
  \((c - c[k-1]) ∈ t.set-of-itemsets\};
  forall candidates c ∈ C_t do
    c.count++;
  if (C_t ≠ ∅) then \(\overline{C}_k += <t.TID, C_t>\);
end
```
### AprioriTid Example

**Min support = 2**

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

$C_3$ has only two transactions (we started with 4)

$L_3$ has the largest itemset

$C_4$ is empty

Stop

Optional: read the correctness proof, buffer managements, data structure from the paper
Discovering Rules
(from the full version of the paper)

Naïve algorithm:

• For every large itemset $p$
  
  – Find all non-empty subsets of $p$
  
  – For every subset $q$
    
    • Produce rule $q \rightarrow (p-q)$
    
    • Accept if $\text{support}(p) / \text{support}(q) \geq \text{minconf}$
Checking the subsets

• For efficiency, generate subsets using recursive DFS. If a subset q does not produce a rule, we do not need to check for subsets of q

Example
Given itemset : ABCD
If ABC $\rightarrow$ D does not have enough confidence
then AB $\rightarrow$ CD does not hold
Reason

- For efficiency, generate subsets using recursive DFS. If a subset q does not produce a rule, we do not need to check for subsets of q

For any subset q’ of q:
  Support(q’) \geq support(q)

  \text{confidence (} q’ \rightarrow (p-q’) \text{)}
  = support(p) / support(q’)
  \leq support(p) / support(q)
  = \text{confidence (} q \rightarrow (p-q))
Simple Algorithm

forall large itemsets \( l_k, k \geq 2 \) do

\[
\text{genrules}(l_k, l_k)
\]

procedure genrules \((l_k: \text{large } k\text{-itemset, } a_m: \text{large } m\text{-itemset})\)

\[
A = \{(m-1)\text{-itemset } a_{m-1} \mid a_{m-1} \subseteq a_m\};
\]

forall \( a_{m-1} \in A \) do begin

\[
\text{conf} = \text{support}(l_k)/\text{support}(a_{m-1})
\]

if \( \text{conf} \geq \text{minconf} \) then begin

output the rule \( a_{m-1} \Rightarrow (l_k - a_{m-1}) \);

if \( m - 1 > 1 \) then

call genrules \((l_k, a_{m-1})\);

end

end

Check all the large itemsets

Check all the subsets

Check confidence of new rule

Output the rule

Continue the depth-first search over the subsets.

If not enough confidence, the DFS branch cuts here
Faster Algorithm

- If \((p-q) \rightarrow q\) holds than all the rules \\
\((p-q') \rightarrow q'\) must hold \\
  - where \(q' \subseteq q\) and is non-empty

Example:
If \(AB \rightarrow CD\) holds, \\
then so do \(ABC \rightarrow D\) and \(ABD \rightarrow C\)

Idea
- Start with 1-item consequent and generate larger consequents
- If a consequent does not hold, do not look for bigger ones
- The candidate set will be a subset of the simple algorithm
Performance

• Support decreases => time increases

• AprioriTID is “almost” as good as Apriori, BUT Slower for larger problems
  – $C^*_k$ does not fit in memory and increases with #transactions
Performance

- AprioriTid is effective in later passes
  - Scans $C^*_k$ instead of the original dataset
  - becomes small compared to original dataset

- When fits in memory, AprioriTid is faster than Apriori
AprioriHybrid

- Use Apriori in initial passes
- Switch to AprioriTid when it can fit in memory
- Switch happens at the end of the pass
  - Has some overhead to switch
- Still mostly better or as good as apriori
From Apriori Algorithm

Subset Function - 1

- Candidate itemsets in $C_k$ are stored in a hash-tree (like a B-tree)
  - interior node = hash table
  - each bucket points to another node at the level below
  - leaf node = itemsets
  - recall that the itemsets are ordered
  - root at level 1 (top-most)
  - All nodes are initially leaves
  - When the number of itemsets in a leaf-node exceeds a threshold, convert it into an interior node

- To add an itemset $c$, start from the root and go down the tree until reach a leaf

Given a transaction $t$ and a candidate set $C_k$, compute the candidates in $C_k$ contained in $t$

$L_1 = \{\text{large 1-itemsets}\}$

For $(k = 2; L_{k-1} \neq \emptyset; k++)$ do begin
  $C_k = \text{apriori-gen}(L_{k-1});$
  forall transactions $t \in D$ do begin
    $C_t = \text{subset}(C_k, t)$
    forall candidates $c \in C_t$ do
      $c\text{-count}++;$
    end
  end
  $L_k = \{c \in C_k \mid c\text{-count} \geq \text{minsup}\}$
end

$Answer = \bigcup_k L_k;$
 Subset Function - 2

• To find all candidates contained in a transaction \( t \)
  – if we are at a leaf
    • find which itemsets are contained in \( t \)
    • add references to them in the answer set
  – if we are at an interior node
    • we have reached it by hashing an item \( i \)
    • hash on each item that comes after \( i \) in \( t \)
    • repair
  – if we are at the root, hash on every item in \( t \)

\[ L_1 = \{ \text{large 1-itemsets} \} \]

For \( k = 2; L_{k-1} \neq \emptyset; k++ \) do begin
\[ C_k = \text{apriori-gen}(L_{k-1}); \]
for all transactions \( t \in D \) do begin
\[ C_t = \text{subset}(C_k, t) \]
for all candidates \( c \in C_t \) do
\[ c.count ++; \]
end
end
\[ L_k = \{ c \in C_k | c.count \geq \text{minsup} \} \]
end
\[ \text{Answer} = \bigcup_k L_k; \]
Subset Function - 3

Why does it work?

For any itemset \( c \) in a transaction \( t \)

- the first item of \( c \) must be in \( t \)
- by hashing on each item in \( t \), we ensure that we only ignore itemsets that start with an item not in \( t \)
- similarly for lower depths
- since the itemset is ordered, if we reach by hashing on \( i \), we only need to consider items that occur after \( i \)

\[
L_1 = \{\text{large 1-itemsets}\}
\]

For \( (k = 2; L_{k,1} \neq \emptyset; k++) \) do begin

\[
C_k = \text{apriori-gen}(L_{k,1});
\]

forall transactions \( t \in D \) do begin

\[
C_t = \text{subset}(C_k,t)
\]

forall candidates \( c \in C_t \) do

\[
c\text{.count}++;\]

end

end

\[
L_k = \{c \in C_k | c\text{.count} \geq \text{minsup}\}
\]

end

\[
Answer = \bigcup_k L_k;
\]
Conclusions
(of 516, Spring 2016)
Take-Aways

• DBMS Basics

• DBMS Internals

• Overview of Research Areas

• Hands-on Experience in DB systems
DB Systems

• Traditional DBMS
  – PostGres, SQL

• Large-scale Data Processing Systems
  – Spark/Scala, AWS

• New DBMS/NOSQL
  – MongoDB

• In addition
  – XML, JSON, JDBC, Python/Java
DB Basics

• SQL
• RA/Logical Plans
• RC
• Datalog
  – Why we needed each of these languages

• Normal Forms
DB Internals and Algorithms

• Storage
• Indexing
• Operator Algorithms
  – External Sort
  – Join Algorithms
• Cost-based Query Optimization
• Transactions
  – Concurrency Control
  – Recovery
Large-scale Processing and New Approaches

• Parallel DBMS
• Distributed DBMS
• Map Reduce
• NOSQL
Advanced/Research Topics

(In various levels of details)

• Data Warehouse/OLAP/Data Cube
• Data Privacy
• View Selection
• Data Provenance
• Probabilistic Databases
• Crowdsourcing
• Could not cover many more....
Hope some of you will further explore Database Systems/Data Management/Data Analysis/Big Data...

Thank you 😊