Data Mining

CPS 216
Advanced Database Systems

Data mining

- Data \rightarrow knowledge
- DBMS meets AI and statistics
- Usually complex statistical “queries” that are difficult to answer
  » Warehousing is a must if data needs to be integrated from various sources
  » Often done using specialized algorithms outside the DBMS
    - Some recent work on pushing mining inside DBMS
      (Sarawagi et al., SIGMOD 1998)

Data mining problems

- Clustering: group together similar items and separate dissimilar ones
- Prediction: predict values of some attributes from others based on training data
  - Classification: predict the “class label”
  - Regression: predict a numeric attribute value
- Association analysis: detect attribute-value conditions that occur frequently together
- Outlier analysis, evolution analysis, etc., etc.

Data mining applications

- Business
  - Marketing, finance, investment, insurance…
    - Urban legend: WalMart discovered that people who bought diapers tended to buy beer at the same time
- Science
  - Astronomy, environmental science, genomics…
- Law enforcement
  - Fraud detection, criminal profiling…

Association rule mining

A.k.a. market-basket analysis

- A transaction (market basket) contains a set of items bought together
- Given a lot of transactions, discover rules such as “diaper \Rightarrow beer” or “digital camera, scanner \Rightarrow graphics software”

<table>
<thead>
<tr>
<th>TID</th>
<th>items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>diaper, milk, candy</td>
</tr>
<tr>
<td>1002</td>
<td>milk, egg</td>
</tr>
<tr>
<td>1003</td>
<td>milk, beer</td>
</tr>
<tr>
<td>1004</td>
<td>diaper, milk, egg</td>
</tr>
<tr>
<td>1005</td>
<td>diaper, beer</td>
</tr>
<tr>
<td>1006</td>
<td>milk, beer</td>
</tr>
<tr>
<td>1007</td>
<td>diaper, beer</td>
</tr>
<tr>
<td>1008</td>
<td>diaper, milk, beer, candy</td>
</tr>
<tr>
<td>1009</td>
<td>diaper, milk, beer</td>
</tr>
</tbody>
</table>

Association rules

- An association rule has the form $X \Rightarrow Y$, where $X$ and $Y$ are disjoint itemsets (sets of items)
  - Confidence $c\%$: $c\%$ of the transactions that contain $X$ also contain $Y$
  - Support $s\%$: $s\%$ of all transactions contain both $X$ and $Y$
  - Note: association rules are directional
    - “Diaper \Rightarrow beer” and “beer \Rightarrow diaper” mean different things
- Problem: Given a set of transactions, find all association rules with confidence and support greater than or equal to specified thresholds $c_{min}\%$ and $s_{min}\%$
Mining association rules

- Step 1: Find frequent itemsets, and count the number of times they appear in transactions
  - An itemset \( X \) is frequent if no less than \( s_{\min \%} \) of all transactions contain \( X \)
    - That is, \( \text{count}(X) \geq s_{\min \%} \cdot \text{total \# of transactions} \)
  - Step 2: Mine association rules from frequent itemsets

Finding frequent itemsets

- First try: a brute-force approach
  - Keep a running count for each possible itemset
  - For each transaction \( T \), and for each itemset \( X \), if \( T \) contains \( X \) then increment the count for \( X \)
  - Return itemsets with large enough counts
- Problem: The number of itemsets is huge!
  - \( 2^n \), where \( n \) is the number of items
  - Think: How do you prune the search space?

The Apriori property

- All subsets of a frequent itemset must also be frequent
  - Because any transaction that contains \( X \) must also contain subsets of \( X \)
  - If you have already verified that \( X \) is infrequent, there is no need to count \( X \)'s supersets because they must be infrequent too

The Apriori algorithm

Agrawal & Srikant, VLDB 1994

- Multiple passes over the transactions
- Pass \( k \) finds all frequent \( k \)-itemsets (itemset of size \( k \))
- Use the set of frequent \( (k-1) \)-itemsets found in the previous pass to narrow the search for \( k \)-itemsets

Pseudo-code for Apriori

Scan the transactions to find \( L_1 \), the set of all frequent 1-itemsets, together with their counts;
for \( k = 2; L_{k-1} \neq \emptyset; k++ \) {
  Generate \( C_k \), the set of candidate \( k \)-itemsets,
  from \( L_{k-1} \), the set of frequent \( (k-1) \)-itemsets found in the previous step;
  Scan the transactions to count the occurrences of itemsets in \( C_k \);
  Find \( L_k \), a subset of \( C_k \) containing \( k \)-itemsets with counts no less than \( s_{\min \%} \cdot \text{total \# of transactions} \);
  Return \( L_1 \cup L_2 \cup \ldots \cup L_k \);
}

Candidate generation

From \( L_{k-1} \) to \( C_k \)

- Join: combine frequent \( (k-1) \)-itemsets to form candidate \( k \)-itemsets
- Prune: ensure every size-\((k-1)\) subset of a candidate is frequent
Candidate generation: join

- Combine almost-matching pairs of frequent 
  \((k-1)\)-itemsets
  
  $$\begin{align*}
  &\text{INSERT INTO } C_k \\
  &\quad \text{SELECT } p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_{k-1} \\
  &\quad \text{FROM } L_{k-1} p, L_{k-1} q \\
  &\quad \text{WHERE } p.\text{item}_i = q.\text{item}_i \\
  &\quad \text{AND } p.\text{item}_2 = q.\text{item}_2 \text{ AND } \cdots \\
  &\quad \text{AND } p.\text{item}_{k-2} = q.\text{item}_{k-2} \\
  &\quad \text{AND } p.\text{item}_{k-1} < q.\text{item}_{k-1};
  \end{align*}$$

  » The last conjunct ensures no duplicates
  » Can you justify why \(p\) and \(q\) should almost match?

Candidate generation: prune

- Remove candidates with an infrequent 
  size-\((k-1)\) subset
  
  \(-\text{for each itemset } c \text{ in } C_k:\n  \quad \text{for each item } x \text{ in } c:\n  \quad \quad \text{if } c -\{x\} \text{ is not in } L_{k-1} \text{ then:} \\
  \quad \quad \quad \quad \text{delete } c \text{ from } C_k\)
Example: final answer

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>6</td>
</tr>
<tr>
<td>{B}</td>
<td>7</td>
</tr>
<tr>
<td>{C}</td>
<td>6</td>
</tr>
<tr>
<td>{D}</td>
<td>2</td>
</tr>
<tr>
<td>{E}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B}</td>
<td>4</td>
</tr>
<tr>
<td>{A, C}</td>
<td>4</td>
</tr>
<tr>
<td>{A, E}</td>
<td>2</td>
</tr>
<tr>
<td>{B, C}</td>
<td>4</td>
</tr>
<tr>
<td>{B, D}</td>
<td>2</td>
</tr>
<tr>
<td>{B, E}</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C}</td>
<td>2</td>
</tr>
<tr>
<td>{A, B, E}</td>
<td>2</td>
</tr>
</tbody>
</table>

Mining rules from frequent itemsets

- for each frequent itemset \( l \):
  - for each nonempty proper subset \( s \) of \( l \):
    - if confidence = \( \frac{\text{count}(l)}{\text{count}(s)} \) \( \geq c_{\min}% \) then:
      - output \( s \Rightarrow (l - s) \);

Example: rules from \( l = \{A, B, E\} \) are
- \( A \Rightarrow B, E \) (confidence 2/4 = 50%)
- \( A \Rightarrow E \) (confidence 2/2 = 100%)
- \( B \Rightarrow A, E \) (confidence 2/2 = 100%)
- \( B \Rightarrow B, E \) (confidence 2/2 = 100%)
- \( B \Rightarrow A, B \) (confidence 2/7 = 29%)
- \( E \Rightarrow A, B \) (confidence 2/7 = 29%)

Data structure for counting itemsets

- Problem: Given a transaction, determine which itemsets in \( C_k \) it contains
- Idea: Build a tree structure from \( C_k \)
- Example: \( C_2 = \{ \{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}, \{B, C\}, \{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\} \} \)

Transaction: A, D, E, F
- Remember all nodes visited
- If item matches any outgoing edge, follow it
- At a leaf node, increment count

Hash tree

- What if the tree may too big or imbalanced?
- Merge edges using a hash function
- Leaves must now store the exact itemset

Other tricks and extensions

- Transaction reduction
  - If a transaction does not contain any frequent \( k \)-itemset, remove it from further consideration
    - AprioriTid, AprioriHybrid, from the same paper
- Dynamic itemset counting
  - Why only introduce candidate itemsets at the end of a scan?
    - Start counting them whenever there is enough support from smaller itemsets
  - Fewer passes over data
    - Brin et al., SIGMOD 1997
- Parallelization, sampling, incremental mining, etc.

End-semester logistics

- Project
  - Demo: You should have received an email about your scheduled slot
  - Report: Due on the day of the final exam
- Final exam
  - Thursday, December 13, 9:00am – 12:00pm
  - In this room
  - Comprehensive, emphasis on the latter half
  - Open book, open notes
- TA and instructor office hours
  - Same as regular office hours
Some points to remember from 216

- Declarativeness is good
  - Relational model, relational algebra, SQL, …
- Redundancy is bad
  - Normal forms, decomposition, …
- Redundancy is good (for performance, as long as you can hide it)
  - Replication, warehousing, materialized views, indexes, …
- One more level of indirection solves lots of things
  - Secondary indexes, wrappers, …
- Query optimizer is really "query goodifier"
  - Assumptions and heuristics to narrow the search space…
- Think beyond tables
  - Bitmap indexes, wavelet histograms, data cube, MOLAP, …

Next semester

- CPS 296.1: Advanced topics in databases
  - Data mining
  - XML and Web data processing
  - Incremental and approximate query evaluation
  - Materialized views for caching and warehousing