Reading Material

Optional Reading:
1. [RG]: Chapter 26
23,038 citations on Google Scholar in November 2017 |
• 20,610 in November 2016
• 19,496 in April 2016
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
  – 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

Related: 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 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 (mass e-mail, webpages)
  - 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
  - $X \rightarrow Y$ (not Functional Dependency!)
  - $X \subseteq I$
  - $X \cap Y = \emptyset$
Confidence and Support

• Association rule $X \rightarrow Y$

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

• Support $s = \frac{|\text{Tr. with } X \text{ and } Y|}{|\text{all Tr.}|}$
  - % 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>X</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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Support(Cereal)?
• Support(Cereal $\rightarrow$ Milk)?

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>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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Confidence(Cereal $\rightarrow$ Milk)?
• Confidence(Bananas $\rightarrow$ Bread)?

X $\rightarrow$ Y is not a Functional Dependency

For functional dependencies
- F.D. = two tuples with the same value 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 $XY \rightarrow A$
- $X \rightarrow A$ and $A \rightarrow Z$ do not mean $X \rightarrow Z$

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 $\geq$ minsup and
  - confidence $\geq$ minconf

Decomposition into two subproblems

• 1. Apriori and AprioriTID:
  - for finding “large” itemsets with support $\geq$ 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 $\geq$ minconf

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

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

Apriori vs. AprioriTID

• Both follow the basic ideas in the previous slides
• AprioriTID has the additional property that 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
• AprioriTID not covered in class (see paper)

Algorithm Apriori

\[ L_k = \text{(large } (k-1)\text{-itemsets)} \]

For \( k = 2 \), \( L_2 \neq \emptyset \), \( k++ \) do begin

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

for all transactions \( t \) do begin

\[ C_k \subset \text{subsets}(C_k) \]

end

end

\[ L_k = \{ f \in C_k | \text{support}(f) \geq \text{minsup} \} \]

end

Answer = \( \bigcup L_k \)

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

Notations

• Assume the database is of the form \( \langle \text{TID, item} \rangle \) where items are stored in lexicographic order
• \( \text{TID} = \) identifier of the transaction
• Also works when the database is "normalized": each database record is \( \langle \text{TID, item} \rangle \) pair

Apriori-Gen

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

- **Join step**
  \[ L_{k-1} \Rightarrow L_k \]

  - \( p \) and \( q \) are two large \( (k-1) \)-itemsets identical in all \( k-2 \) first items.
  - Join by adding the last item of \( q \) to \( p \).

- **Prune step**
  - Take \( \text{all subsets } s \text{ of } c \text{ do} \)
    - if \( s \not\in L_{k-1} \) then delete \( c \) from \( C_k \).
  - Check all the subsets, remove all candidate with some "small" subset
Apriori-Gen Example - 1

Step 1: Join (k = 4)
Assume numbers 1-5 correspond to individual items

\[
\begin{align*}
L_3 & \quad C_4 \\
\{1,2,3\} & \quad \{1,2,3,4\} \\
\{1,2,4\} & \quad \{1,2,3\} \\
\{1,3,4\} & \quad \{1,3,5\} \\
\{2,3,4\} & \quad \{1,2,4\} \\
\end{align*}
\]

Apriori-Gen Example - 2

Step 1: Join (k = 4)
Assume numbers 1-5 correspond to individual items

\[
\begin{align*}
L_3 & \quad C_4 \\
\{1,2,3\} & \quad \{1,2,3,4\} \\
\{1,2,4\} & \quad \{1,2,3,4\} \\
\{1,3,4\} & \quad \{1,3,4,5\} \\
\{1,3,5\} & \quad \{1,3,5,6\} \\
\{2,3,4\} & \quad \{1,2,4\} \\
\end{align*}
\]

Apriori-Gen Example - 3

Step 2: Prune (k = 4)
Remove items that can’t have the required support because there is a subset in \( t \) which doesn’t have the level of support i.e. not in the previous pass (k-1)

\[
\begin{align*}
L_3 & \quad C_4 \\
\{1,2,3\} & \quad \{1,2,3,4\} \\
\{1,2,4\} & \quad \{1,2,3\} \\
\{1,3,4\} & \quad \{1,3,4,5\} \\
\{1,3,5\} & \quad \{1,3,4\} \\
\{2,3,4\} & \quad \{1,2,4\} \\
\end{align*}
\]

Comparisons with previous algorithms (AIS, STEM)

\[
\begin{align*}
L_{k-1} \text{ to } C_k \\
- \quad \text{Read each transaction } t \text{ that are in } L_{k-1} \text{ and occur later in lexicographic order} \\
- \quad \text{Extend } p \text{ with large items in } t \text{ and } t \text{ = \{1, 2, 3, 4, 5\}} \text{ all 1-5 large items (why?)} \\
- \quad \text{5 candidates compared to 2 (after pruning 1) in Apriori} \\
\end{align*}
\]

Correctness of Apriori

Check yourself

Show that \( C_3 \supseteq L_3 \)

- Any subset of large itemset must also be large
- for each \( p \) in \( L_3 \) it has a subset \( q \) in \( L_{k-1} \)
- We are extending those subsets \( q \) in \( L_{k-1} \) with another subset \( p \) 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 \( L_3 \)

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
- Solutions
  - AprioriTID
  - Hybrid
  - Optional/not covered
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 \( AB \rightarrow D \) does not have enough confidence
then \( AB \rightarrow CD \) does not hold

\[
\text{minconf} > \frac{\text{Confidence}(ABC \rightarrow D)}{\text{Support}(ABCD)} = \frac{\text{Support}(ABCD)}{\text{Support}(ABC)} \geq \frac{\text{Confidence}(AB \rightarrow CD)}{\text{Support}(ABCD)} \frac{\text{Support}(ABCD)}{\text{Support}(AB)}
\]

Simple Algorithm

```plaintext
forall large itemsets \( I_p \) \( k \geq 2 \) do
  generate(\( I_p \))
  foreach \( A \) do
    check all the large itemsets
    conf = \( \text{support}(I_p) / \text{support}(A) \)
    if \( \text{conf} \geq \text{minconf} \)
      output the rule \( A \rightarrow (I_p - A) \)
      continue the depth-first search over the subsets.
    else
      if not enough confidence, the DFS branch cuts here.
end
end
```

More Optimizations

Example:
If \( AB \rightarrow CD \) holds
  - \( \text{conf} = \text{support}(ABCD) / \text{support}(AB) \) \( \geq \text{minconf} \)
then so do \( ABC \rightarrow D \) and \( ABD \rightarrow C \)
  - \( \text{conf} = \text{support}(ABCD) / \text{support}(ABD) \)

In general,
- If \( \text{conf} \) of \( (p-q) \rightarrow q \) holds than all the rules \( (p-q') \rightarrow q' \) must hold
  - where \( q' \subseteq q \) and is non-empty

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

Views (revisiting)

- Motivation (example)
  - Different groups of analysts within an organization are typically concerned with different aspects of a business
  - It is convenient to define “views” that give each group insight into the relevant business details
  - Other views can be defined or queries can be written using these views
  - Convenient and Efficient

View Materialization and Maintenance

[RG] Chapters 25.8-25.10
**Views and OLAP/Warehousing**

- OLAP queries are typically aggregate queries
  - Precomputation is essential for interactive response times
  - The CUBE is in fact a collection of aggregate queries, and precomputation is especially important
  - Lots of work on what is best to precompute given a limited amount of space to store precomputed results.

- Warehouses can be thought of as a collection of asynchronously replicated tables and periodically maintained views
  - Factors: size, number of tables involved, many are from external independent databases
  - Has renewed interest in (asynchronous) view maintenance (more later)

**View Materialization**

- **Query Modification may not be efficient**
  - When the underlying view is complex
  - Even with sophisticated optimization and evaluation
  - Especially when the underlying tables are in a remote database (connectivity and availability)

- **Alternative: View Materialization**
  - Precompute the view definition and store the result
  - Materialized views can be used as regular relations
  - Provides faster access, like a very high-level cache
  - Can create index on views too for further speedup
  - Drawback: maintain the consistency of the materialized view when the underlying table(s) are updated (View Maintenance)
    - Ideally, we want incremental View Maintenance algorithms (Lecture 22)

**Index on Materialized Views: Examples**

- Suppose we precompute RegionalSales and store it with a clustered B+ tree index on [category, state, sales].
  - Then, the query can be answered by an index-only scan.
  
  ```sql
  CREATE VIEW RegionalSales AS
  SELECT R.state, SUM(R.sales) FROM RegionalSales AS R
  GROUP BY R.state
  INDEX ON RegionalSales AS R
  ```

  - Index on precomputed view is great!

```sql
SELECT R.state, SUM(R.sales) FROM RegionalSales AS R
WHERE R.state="Wisconsin"
GROUP BY R.state
 INDEX ON RegionalSales AS R
```

- Index is less useful (must scan entire leaf level)

**View Maintenance**

- Two steps:
  - **Propagate:** Compute changes to view when data changes
  - **Refresh:** Apply changes to the materialized view table

- **Maintenance policy:** Controls when we do refresh
  - **Immediate:** As part of the transaction that modifies the underlying data tables
    - Materialized view is always consistent
    - Updates are slow
  - **Deferred:** Some time later, in a separate transaction
    - View becomes inconsistent
    - Can scale to maintain many views without slowing updates
Types of Deferred Maintenance

Three flavors:

- **Lazy:**
  - Delay refresh until next query on view; then refresh before answering the query (slows down queries than updates)

- **Periodic (Snapshot):**
  - Refresh periodically (e.g., once in a day). Queries possibly answered using outdated version of view tuples. Widely used, especially for asynchronous replication in distributed databases, and for warehouse applications

- **Event-based or Forced:**
  - E.g., Refresh after a fixed number of updates to underlying data tables

- e.g. **Snapshot in Oracle 7**
  - Periodically refreshed by entirely recomputing the view
  - Incremental “fast refresh” or “simple snapshots” for simpler views (no aggregate, group by, join, distinct etc.)