Today’s Reading:
- HMS 6.7, ch. 13

Today’s Lecture:
- Finding Patterns and Rules
  - Association Rules

Upcoming Due Dates:
- H3 due today
- Project Writeup due 4/30

Finding Patterns and Rules
- Examples:
  - supermarket transaction database, 10% of the customers buy wine and cheese
  - telecommunications alarms database if alarms A and B occur within 30 seconds of each other, then alarm C occurs within 60 seconds with probability 0.5
  - web log dataset if a person visits the CNN Web site, there is 60% chance the person will visit the ABC News Web site in the same month

What Is Association Mining?
- Association rule mining:
  - Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
  - Applications:
    - Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.
  - Examples.
    - Rule form: "Body $\rightarrow$ Head [support, confidence]".
    - buys(x, "diapers") $\rightarrow$ buys(x, "beers") [0.5%, 60%]
    - major(x, "CS") $\land$ takes(x, "DB") $\rightarrow$ grade(x, "A") [1%, 75%]
Market Basket Analysis

- Given some set of items
  1. database of transactions,
  2. each transaction is a list of items (purchased by a customer in a visit), called a basket
- Find: all rules that correlate the presence of one set of items with that of another set of items in a basket
  - E.g., 98% of people who purchase tires and auto accessories also get automotive services done
- Other problems with this structure:
  - baskets = documents; items = words
  - baskets = web pages; items = links

Rule Measures: Support and Confidence

Find all the rules $X \& Y \Rightarrow Z$ with minimum confidence and support

- $support$, $s$, proportion of transactions containing $X, Y, Z$
- $confidence$, $c$, proportion of transactions which have $X, Y$ and also contain $Z$

Customer buys both

Customer buys diaper

Customer buys beer

Example: Support and Confidence

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items Bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>A, B, C</td>
</tr>
<tr>
<td>1000</td>
<td>A, C</td>
</tr>
<tr>
<td>4000</td>
<td>A, D</td>
</tr>
<tr>
<td>5000</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>

Let minimum support 50%, and minimum confidence 50%

we have:

- $A \Rightarrow C$  
  $s = 50\%$  
  $c = 66.6\%$

- $C \Rightarrow A$  
  $s = 50\%$  
  $c = 100\%$

Min. support 50%

Min. confidence 50%

Finding Frequent Itemsets

- Find the frequent itemsets: the sets of items that have minimum support
  - Monotonicity principle: A subset of a frequent itemset must also be a frequent itemset
    - i.e., if $(AB)$ is a frequent itemset, both $(A)$ and $(B)$ should be a frequent itemset
  - Iteratively find frequent itemsets with cardinality from 1 to $k$ ($k$-itemset)
- Use the frequent itemsets to generate association rules.

Mining Association Rules—An Example

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Items Bought</th>
<th>Min. support 50%</th>
<th>Min. confidence 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>A, B, C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>A, C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>A, D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>B, E, F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For rule $A \Rightarrow C$

- support = support($A,C$) = 50%
- confidence = support($A,C$)/support($A$) = 66.6%

The Apriori principle:

Any subset of a frequent itemset must be frequent
Finding frequent itemsets, cont.

- Two approaches
  - Proceed levelwise, first find frequent items (sets of size 1), then frequent pairs, next frequent triples
  - Most time consuming step is often finding pairs
  - One pass over the data is made for each level
- Find all maximal frequent itemsets (i.e., sets S such that no proper superset of S is frequent)
  - One or several passes over the data

The Apriori Algorithm

- Join Step: Ck is generated by joining Lk−1 with itself
- Prune Step: Any (k-1)-itemset that is not frequent cannot be a subset of a frequent k-itemset

\[ C_k : \text{Candidate itemset of size } k \]
\[ L_k : \text{frequent itemset of size } k \]

\[ L_1 = \{\text{frequent items} \}; \]

\[
\text{for } (k = 1; L_k \neq \emptyset; k++) \text{ do begin}
\]
\[ C_{k+1} = \text{candidates generated from } L_k \]
\[ \text{for each transaction } t \text{ in database do}
\]
\[ \text{increment the count of all candidates in } C_{k+1} \text{ that are contained in } t \]
\[ L_{k+1} = \text{candidates in } C_{k+1} \text{ with min_support} \]
\[ \text{end} \]

return \( \bigcup_k L_k \);

The Apriori Algorithm — Example

Database D

<table>
<thead>
<tr>
<th>ID</th>
<th>Items</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>

L1

\[ \text{Scan D} \]
\[
\begin{array}{c|c}
\text{Items} & \text{sup} \\
\hline
1 & 2 \\
2 & 3 \\
3 & 3 \\
4 & 1 \\
5 & 3 \\
\end{array}
\]

C1

\[ \text{Scan D} \]
\[
\begin{array}{c|c}
\text{Items} & \text{sup} \\
\hline
(1) & 2 \\
(2) & 3 \\
(3) & 3 \\
(4) & 1 \\
(5) & 3 \\
\end{array}
\]

L2

\[ \text{Scan D} \]
\[
\begin{array}{c|c}
\text{Itemset} & \text{sup} \\
\hline
(1 2) & 1 \\
(2 3) & 2 \\
(2 5) & 3 \\
(3 5) & 2 \\
\end{array}
\]

C2

\[ \text{Scan D} \]
\[
\begin{array}{c|c}
\text{Itemset} & \text{sup} \\
\hline
(1 2) & 1 \\
(1 3) & 2 \\
(1 5) & 1 \\
(2 3) & 2 \\
(2 5) & 3 \\
(3 5) & 2 \\
\end{array}
\]

L3

\[ \text{Scan D} \]
\[
\begin{array}{c|c}
\text{Itemset} & \text{sup} \\
\hline
(2 3 5) & 2 \\
\end{array}
\]

C3

\[ \text{scan D} \]

How to Generate Candidates?

- Suppose the items in \( L_{k-1} \) are ordered
- Step 1: self-joining \( L_{k-1} \)
  - Insert into \( C_k \)
  - Select \( p.item_1, \ldots, p.item_{k-1}, q.item_{k-1} \)
  - From \( L_{k-1} \) \( p \rightarrow L_{k-1} q \)

- Step 2: pruning
  - For all \( \text{itemsets } c \text{ in } C_k \) do
    - For all \((k-1)\)-subsets \( s \text{ of } c \) do
      - If \( s \text{ is not in } L_{k-1} \) then delete \( c \) from \( C_k \)

Example of Generating Candidates

- \( L_3 = \{abc, abd, acd, ace, bcd\} \)
- Self-joining: \( L_2 \ast L_2 \)
  - \( abcd \) from \( abc \) and \( abd \)
  - \( ace \) from \( acd \) and \( ace \)
- Pruning:
  - \( ace \) is removed because \( ade \) is not in \( L_3 \)
  - \( C_4 = \{abcd\} \)

How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a hash-tree
  - Leaf node of hash-tree contains a list of itemsets and counts
  - Interior node contains a hash table
  - Subset function finds all the candidates contained in a transaction
Methods to Improve Apriori’s Efficiency
• Hash-based itemset counting: A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
• Transaction reduction: A transaction that does not contain any frequent k-itemset is useless in subsequent scans
• Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
• Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness
• Dynamic itemset counting: add new candidate itemsets only when all of their subsets are estimated to be frequent

Is Apriori Fast Enough? — Performance Bottlenecks
• The core of the Apriori algorithm:
  – Use frequent (k−1)-itemsets to generate candidate frequent k-itemsets
  – Use database scan and pattern matching to collect counts for the candidate itemsets
• The bottleneck of Apriori: candidate generation
  – Huge candidate sets: 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
  – To discover a frequent pattern of size 100, e.g., (a_1, a_2, …, a_100), one needs to generate 2^100 ≈ 10^30 candidates.
  – Multiple scans of database:
    • Needs (n+1) scans, n is the length of the longest pattern

Mining Frequent Patterns Without Candidate Generation
• Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  – Highly condensed, but complete for frequent pattern mining
  – Avoid costly database scans
• Develop an efficient, FP-tree-based frequent pattern mining method
  – A divide-and-conquer methodology: decompose mining tasks into smaller ones
  – Avoid candidate generation: sub-database test only

Benefits of the FP-tree Structure
• Completeness:
  – never breaks a long pattern of any transaction
  – preserves complete information for frequent pattern mining
• Compactness
  – reduce irrelevant information—in frequent items are gone
  – frequency descending ordering: more frequent items are more likely to be shared
  – never be larger than the original database
  – Example: For Connect-4 DB, compression ratio could be over 100

Construct FP-tree from a Transaction DB

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought (ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, i, m, p}</td>
</tr>
<tr>
<td>200</td>
<td>{a, b, c, f, i, m, p}</td>
</tr>
<tr>
<td>300</td>
<td>{b, f, h, i, o}</td>
</tr>
<tr>
<td>400</td>
<td>{b, c, h, j, p}</td>
</tr>
<tr>
<td>500</td>
<td>{a, f, c, i, j, p, m, n}</td>
</tr>
</tbody>
</table>

min_support = 0.5

Steps:
1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Mining Frequent Patterns Using FP-tree
• General idea (divide-and-conquer)
  – Recursively grow frequent pattern path using the FP-tree
• Method
  – For each item, construct its conditional pattern-base, and then its conditional FP-tree
  – Repeat the process on each newly created conditional FP-tree
  – Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)
Major Steps to Mine FP-tree

1) Construct conditional pattern base for each node in the FP-tree
2) Construct conditional FP-tree from each conditional pattern-base
3) Recursively mine conditional FP-trees and grow frequent patterns obtained so far
   • If the conditional FP-tree contains a single path, simply enumerate all the patterns

Properties of FP-tree for Conditional Pattern Base Construction

• Node-link property
  – For any frequent item \( a_i \), all the possible frequent patterns that contain \( a_i \) can be obtained by following \( a_i \)'s node-links, starting from \( a_i \)'s head in the FP-tree header
• Prefix path property
  – To calculate the frequent patterns for a node \( a_i \) in a path \( P \), only the prefix sub-path of \( a_i \) in \( P \) need to be accumulated, and its frequency count should carry the same count as node \( a_i \),

Mining Frequent Patterns by Creating Conditional Pattern-Bases

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern-base</th>
<th>Conditional FP-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{((fca:2), (cb:1))}</td>
<td>{((c:3)):p}</td>
</tr>
<tr>
<td>m</td>
<td>{((fca:2), (fabc:1))}</td>
<td>{((f:3, c:3, a:3)):m}</td>
</tr>
<tr>
<td>b</td>
<td>{((fca:1), (f:1), (c:1))}</td>
<td>Empty</td>
</tr>
<tr>
<td>a</td>
<td>{((f:3))}</td>
<td>{((f:3)):a}</td>
</tr>
<tr>
<td>c</td>
<td>{((f:3))}</td>
<td>{((f:3)):c}</td>
</tr>
<tr>
<td>f</td>
<td>Empty</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Step 1: From FP-tree to Conditional Pattern Base

• Starting at the frequent header table in the FP-tree
• Traverse the FP-tree by following the link of each frequent item
• Accumulate all of transformed prefix paths of that item to form a conditional pattern base

Step 2: Construct Conditional FP-tree

• For each pattern-base
  – Accumulate the count for each item in the base
  – Construct the FP-tree for the frequent items of the pattern base

Step 3: Recursively mine the conditional FP-tree

Cond. pattern base of "am": \{((f:3))\} | cond. FP-tree

Cond. pattern base of "cm": \{((f:3))\} | cond. FP-tree

Cond. pattern base of "cam": \{((f:3))\} | cond. FP-tree
Single FP-tree Path Generation

• Suppose an FP-tree T has a single path P
• The complete set of frequent patterns of T can be generated by enumeration of all the combinations of the sub-paths of P

\[
\begin{array}{c|c|c|c|c|c|c|c}
| \text{Pattern} | \text{Support} | \text{Confidence} | \text{ Lift } | \text{ Support Ratio } | \\
\hline
\{m\} & - & - & - & - & - \\
\{f\} & - & - & - & - & - \\
\{e\} & - & - & - & - & - \\
\{a\} & - & - & - & - & - \\
\{m, f, e, a\} & & & & & \\
\{m, f, e\} & & & & & \\
\{m, e, a\} & & & & & \\
\{f, e, a\} & & & & & \\
\{m, f, a\} & & & & & \\
\{f, e, a\} & & & & & \\
\{m, e, a\} & & & & & \\
\{m, f, e, a\} & & & & & \\
\end{array}
\]

m-conditional FP-tree

Principles of Frequent Pattern Growth

• Pattern growth property
  – Let \( \alpha \) be a frequent itemset in DB, B be \( \alpha \)'s conditional pattern base, and \( \beta \) be an itemset in B. Then \( \alpha \cup \beta \) is a frequent itemset in DB iff \( \beta \) is frequent in B.
  
• "abcdef" is a frequent pattern, if and only if
  – "abcde" is a frequent pattern, and
  – "f" is frequent in the set of transactions containing "abcde"

Why Is Frequent Pattern Growth Fast?

• Performance study shows
  – in some cases FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection
• Reasoning
  – No candidate generation, no candidate test
  – Use compact data structure
  – Eliminate repeated database scan
  – Basic operation is counting and FP-tree building

Presentation of Association Rules (Table Form)

Visualization of Association Rule Using Plane Graph
Visualization of Association Rule Using Rule Graph

Iceberg Queries
- **Icerberg query**: Compute aggregates over one or a set of attributes only for those whose aggregate values is above certain threshold
- **Example**: 
  ```sql
  select P.custID, P.itemID, sum(P.qty) 
  from purchase P 
  group by P.custID, P.itemID 
  having sum(P.qty) >= 10 
  ```
- **Compute iceberg queries efficiently by Apriori**:
  - First compute lower dimensions
  - Then compute higher dimensions only when all the lower ones are above the threshold

Multiple-Level Association Rules
- Items often form hierarchy.
- Items at the lower level are expected to have lower support.
- Rules regarding itemsets at appropriate levels could be quite useful.
- Transaction database can be encoded based on dimensions and levels
- We can explore shared multi-level mining

Mining Multi-Level Associations
- A top-down, progressive deepening approach:
  - First find high-level strong rules: 
    - milk → bread [20%, 60%].
  - Then find their lower-level "weaker" rules:
    - 2% milk → wheat bread [6%, 50%].
- Variations at mining multiple-level association rules.
  - Level-crossed association rules: 
    - 2% milk → Wonder wheat bread
  - Association rules with multiple, alternative hierarchies: 
    - 2% milk → Wonder bread

Multi-level Association: Uniform Support vs. Reduced Support
- Uniform Support: the same minimum support for all levels
  - One minimum support threshold. No need to examine itemsets containing any item whose ancestors do not have minimum support.
  - Lower level items do not occur as frequently. If support threshold too high ⇒ miss low level associations
  - too low ⇒ generate too many high level associations
- Reduced Support: reduced minimum support at lower levels
  - There are 4 search strategies:
    - Level-by-level independent
    - Level-cross filtering by k-itemset
    - Level-cross filtering by single item
    - Controlled level crossing filtering by single item

Uniform Support
Multi-level mining with uniform support

<table>
<thead>
<tr>
<th>Level 1</th>
<th>min_sup = 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milk</td>
<td>[support = 10%]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 2</th>
<th>min_sup = 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2% Milk</td>
<td>[support = 6%]</td>
</tr>
<tr>
<td>Skim Milk</td>
<td>[support = 4%]</td>
</tr>
</tbody>
</table>
Reduced Support
Multi-level mining with reduced support

Level 1
min_sup = 5%

- Milk [support = 10%]

Level 2
min_sup = 3%

- 2% Milk [support = 6%]
- Skim Milk [support = 4%]

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items.
- Example
  - milk ⇒ wheat bread [support = 8%, confidence = 70%]
  - 2% milk ⇒ wheat bread [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor.

Multi-Level Mining: Progressive Deepening

- A top-down, progressive deepening approach:
  - First mine high-level frequent items: milk (15%), bread (10%)
  - Then mine their lower-level “weaker” frequent itemsets: 2% milk (5%), wheat bread (4%)
- Different min_support threshold across multi-levels lead to different algorithms:
  - If adopting the same min_supp across multi-levels then toss if any of its ancestors is infrequent.
  - If adopting reduced min_support at lower levels then examine only those descendents whose ancestor’s support is frequent/non-negligible.

Progressive Refinement of Data Mining Quality

- Why progressive refinement?
  - Mining operator can be expensive or cheap, fine or rough
- Superset coverage property:
  - Preserve all the positive answers—allow a positive false test but not a false negative test.
- Two- or multi-step mining:
  - First apply rough/cheap operator (superset coverage)
  - Then apply expensive algorithm on a substantially reduced candidate set (Koperski & Han, SSD’95).

Multi-Dimensional Association: Concepts

- Single-dimensional rules:
  - buys(X, "milk") ⇒ buys(X, "bread")
- Multi-dimensional rules: ⋃ 2 dimensions or predicates
  - Inter-dimension association rules [no repeated predicates]
    - age(X,’19-25’) ∧ occupation(X,’student’) ⇒ buys(X,’coke’)
  - hybrid-dimension association rules [repeated predicates]
    - age(X,’19-25’) ∧ buys(X,’popcorn’) ⇒ buys(X,’coke’)
- Categorical Attributes
  - finite number of possible values, no ordering among values
- Quantitative Attributes
  - numeric, implicit ordering among values

Techniques for Mining MD Associations

- Search for frequent k-predicate set:
  - Example: (age, occupation, buys) is a 3-predicate set.
  - Techniques can be categorized by how age are treated.
  1. Using static discretization of quantitative attributes
     - Quantitative attributes are statically discretized by using predefined concept hierarchies.
  2. Quantitative association rules
     - Quantitative attributes are dynamically discretized into “bins”based on the distribution of the data.
  3. Distance-based association rules
     - This is a dynamic discretization process that considers the distance between data points.
Static Discretization of Quantitative Attributes

- Discretized prior to mining using concept hierarchy.
- Numeric values are replaced by ranges.
- In relational database, finding all frequent k-predicate sets will require k or k+1 table scans.
- Data cube is well suited for mining.
- The cells of an n-dimensional cuboid correspond to the predicate sets.
- Mining from data cubes can be much faster.

Quantitative Association Rules

- Numeric attributes are dynamically discretized
  - Such that the confidence or compactness of the rules mined is maximized.
- 2-D quantitative association rules: \( A_{\text{age1}} \land A_{\text{age2}} \Rightarrow A_{\text{buys}} \)
- Cluster "adjacent" association rules to form general rules using a 2-D grid.
- Example:

Clusters and Distance Measurements

- \( S[X] \) is a set of N tuples \( t_1, t_2, \ldots, t_N \), projected on the attribute set \( X \).
- The diameter of \( S[X] \):
  \[
  d(S[X]) = \frac{\sum_{i=1}^N \sum_{j=1}^N \text{dist}(t_i[X], t_j[X])}{N(N-1)}
  \]
  - \( \text{dist} \) = distance metric, e.g. Euclidean distance or Manhattan.

Clusters and Distance Measurements (Cont.)

- The diameter, \( d \), assesses the density of a cluster \( C_X \), where
  \[
  d(C_X) \leq d_s \geq \epsilon
  \]
- Finding clusters and distance-based rules
  - the density threshold, \( d_s \), replaces the notion of support
  - modified version of the BIRCH clustering algorithm

Interestingness Measurements

- Objective measures
  Two popular measurements:
  - support
  - confidence
- Subjective measures (Silberschatz & Tuzhilin, KDD95)
  A rule (pattern) is interesting if
  - it is unexpected (surprising to the user); and/or
  - actionable (the user can do something with it).

Criticism to Support and Confidence

- Example 1 (Aggarwal & Yu, PODS98)
  - Among 5000 students:
    - 3000 play basketball
    - 3750 eat cereal
    - 2000 both play basketball and eat cereal
  - play basketball \( \Rightarrow \) eat cereal (66.7%) is misleading because the overall percentage of students eating cereal is 75% which is higher than 66.7%.
  - play basketball \( \Rightarrow \) not eat cereal (33.3%) is far more accurate, although with lower support and confidence.
Criticism to Support and Confidence (Cont.)

- Example 2:
  - X and Y: positively correlated,
  - X and Z: negatively related
  - Support and confidence of X⇒Z dominates

- We need a measure of dependent or correlated events

\[
corr_{A,B} = \frac{P(A \land B)}{P(A)P(B)}
\]

- \(P(B|A)/P(B)\) is also called the lift of rule \(A \Rightarrow B\)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>X⇒Y</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>X⇒Z</td>
<td>37.50%</td>
<td>75%</td>
</tr>
</tbody>
</table>

Other Interestingness Measures: Interest

- Interest (correlation, IR)
  \[
  \frac{P(A \land B)}{P(A)P(B)}
  \]
  - taking both \(P(A)\) and \(P(B)\) in consideration
  - \(P(A \land B)/P(B)/P(A)\), if A and B are independent events
  - A and B negatively correlated, if the value is less than 1; otherwise A and B positively correlated

Itemset Support Interest

- | Itemset | Support | Interest |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y</td>
<td>25%</td>
<td></td>
</tr>
<tr>
<td>X,Z</td>
<td>37.50%</td>
<td>0.95</td>
</tr>
<tr>
<td>Y,Z</td>
<td>12.50%</td>
<td>0.57</td>
</tr>
</tbody>
</table>

Presentation of Association Rules (Table Form)

Visualization of Association Rule Using Plane Graph

Visualization of Association Rule Using Rule Graph
References

• Notes from Jeff Ullman’s Data mining course, http://www-db.stanford.edu/~ullman/mining/mining.html
• Vipin Kumar and Mahesh Joshi’s tutorial on High Performance Data Mining, http://www-users.cs.umn.edu/~mjoshi/hpdmtut/