Announcements

• Again, I will be out of town from April 1\textsuperscript{st} to April 16\textsuperscript{th}
  – Sporadic email access, please plan accordingly if you need to discuss your projects with me
  – I will be available for project discussion for the next three days (Tuesday through Thursday)

• Fernando will cover advanced information retrieval topics for the next four lectures

• Quizzes:
  – Last quiz is graded, please pick it up at the end of the class

• Projects:
  – Seems everything is going well 😊
Review of Last Lecture

- **Data mining** = Discover **hidden** and **useful** knowledge from **large amounts** of data
  - Customers who buy Harry Potter books often buy Twilight books
  - Users in NYC tend to search for expensive restaurants on Valentine's Day

- **Classic Studies**
  - Association Rule Mining
  - Data Cube Analysis
Association Rule Mining Review

- **Association Rule**
  - Rule: $X \Rightarrow Y$ for a transaction database $D$
  - Support: $\frac{\sigma_i(T_i \mid X \cup Y \subseteq T_i)}{|D|}$
  - Confidence: $\frac{\sigma_i(T_i \mid X \cup Y \subseteq T_i)}{\sigma_j(T_j \mid X \subseteq T_j)}$
  - And more, such as Lift and Leverage

- **Problem: Frequent Itemset Mining**
- **Problem: Maximally Frequent Itemset Mining**
Data Cube Analysis Review

<table>
<thead>
<tr>
<th>Product</th>
<th>Country</th>
<th>1Qtr</th>
<th>2Qtr</th>
<th>3Qtr</th>
<th>4Qtr</th>
</tr>
</thead>
<tbody>
<tr>
<td>TV</td>
<td>U.S.A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>Canada</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVD</td>
<td>Mexico</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Group By product, date
Group By country, date
Group By country, product
Group By country, product, data, country
Group By date
Group By product
Group By none
Data Cube Analysis Review

• **Iceberg Cube:**
  
  Cube by $d_1, d_2, ...$

  Having `count(*) > s`

• **Measure**

  – Algebraic
  
  – Monotonic

  – Sum, Average, Median, Count Unique, ...
Algorithms

- **Association rule mining**
  - Apriori
  - MaxMiner

- **Cube Analysis**
  - BUC
Outline

• Beyond item sets
  – Mining Sequential Patterns
  – Mining Graph Patterns

• Quiz + Break

• Clustering, if time permits
Alternative Data for Mining

- So far, each transaction is viewed as a set
  - The order of the items is ignored
  - The relationship between the items is ignored

- Sequential pattern mining

- Graph pattern mining
Sequential Pattern Mining

• *Mining Sequential Patterns*, Agrawal and Srikant, ICDE 1995
A General Sequence Definition

• A sequence is an order list of itemsets (more than just an item)
  • $X = \langle 3 \rangle \langle 45 \rangle \langle 8 \rangle$

• Containment: $s$ is contained in $t$ if for a list of positions, $p_1 < p_2 < \ldots < p_n$, $s_{p_1}$ is a subset of $t_{p_1}$, $s_{p_2}$ is a subset of $t_{p_2}$, ..., $s_{p_n}$ is a subset of $t_{p_n}$

  • $Y = \langle 7 \rangle \langle 38 \rangle \langle 9 \rangle \langle 456 \rangle \langle 8 \rangle$
  • $X$ is contained in $Y$
Support and Maximality

• View each customer’s *purchase history* as a sequence
  – Order the purchases (i.e., itemsets) according to time

• **Support** of a sequence $S$:
  – Fraction of customers whose purchase history contains $S$

• **Maximal sequence** $M$ for a support threshold $t$:
  – There is no other sequence $N$, such that $N$ contains $M$ and $N$ has a support greater than $t$
Examples

<table>
<thead>
<tr>
<th>Customer Id</th>
<th>Customer Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{(30) (90)}</td>
</tr>
<tr>
<td>2</td>
<td>{(10 20) (30) (40 60 70)}</td>
</tr>
<tr>
<td>3</td>
<td>{(30 50 70)}</td>
</tr>
<tr>
<td>4</td>
<td>{(30) (40 70) (90)}</td>
</tr>
<tr>
<td>5</td>
<td>{(90)}</td>
</tr>
</tbody>
</table>

Sequential Patterns with support > 25%:

- \{(30) (90)\}
- \{(30) (40 70)\}
Preparation: Transformation

• Three steps:
  – Keep only frequent items
  – Convert an itemset into a set of itemsets
    • \((40 70)\) becomes \(\{(40) \ (70) \ (40 \ 70)\}\)
  – Mapping to new item IDs

<table>
<thead>
<tr>
<th>Customer Id</th>
<th>Original Customer Sequence</th>
<th>Transformed Customer Sequence</th>
<th>After Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>({(30) \ (90)})</td>
<td>({(30)} \ {(90)})</td>
<td>({1} \ {5})</td>
</tr>
<tr>
<td>2</td>
<td>({(10 \ 20) \ (30) \ (40 \ 60 \ 70)})</td>
<td>({(30)} \ {(40), \ (70), \ (40 \ 70)})</td>
<td>({1} \ {2, 3, 4})</td>
</tr>
<tr>
<td>3</td>
<td>({(30 \ 50 \ 70)})</td>
<td>({(30), \ (70)})</td>
<td>({1, 3})</td>
</tr>
<tr>
<td>4</td>
<td>({(30) \ (40 \ 70) \ (90)})</td>
<td>({(30)} \ {(40), \ (70), \ (40 \ 70)} \ {(90)})</td>
<td>({1} \ {2, 3, 4} \ {5})</td>
</tr>
<tr>
<td>5</td>
<td>({(90)})</td>
<td>({(90)})</td>
<td>({5})</td>
</tr>
</tbody>
</table>
Algorithm Apriori for Sequence

• Starts with frequent sequences of length 1
• At each iteration k:
  – Identify frequent k-sequences
  – Self-join to generate candidate (k+1)-sequences
  – Scan database to compute support and prune away infrequent candidates
    • Hash Tree can be adapted here
• Check for Maximality
  – Top-down scan

• Familiar? Yes, it’s the same Apriori algorithm applied to a different setting
Self-Join Example

With candidate k-sequences
- Join any two k-sequences with the same (k-1)-prefix
- Permute the k-th item
- \(<1\ 2\ 3>\) join \(<1\ 2\ 4>\) \(\rightarrow\) \(<1\ 2\ 3\ 4>\) and \(<1\ 2\ 4\ 3>\)

<table>
<thead>
<tr>
<th>1-Sequences</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;2)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;3)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;4)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;5)</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2-Sequences</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1\ 2)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;1\ 3)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;1\ 4)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;1\ 5)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;2\ 3)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;2\ 4)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;3\ 4)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;3\ 5)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;4\ 5)</td>
<td>2</td>
</tr>
</tbody>
</table>

Lead to 20 2-sequences

<table>
<thead>
<tr>
<th>3-Sequences</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1\ 2\ 3)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;1\ 2\ 4)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;1\ 3\ 4)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;1\ 3\ 5)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;2\ 3\ 4)</td>
<td>2</td>
</tr>
</tbody>
</table>

Lead to 12 + 2 + 2 + 0 = 16 3-sequences

<table>
<thead>
<tr>
<th>4-Sequences</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;1\ 2\ 3\ 4)</td>
<td>2</td>
</tr>
</tbody>
</table>

Lead to 2 + 2 + 0 = 4 4-sequences
Additional Twists

- **Incorporating item hierarchy**
  - E.g., iPod is a product of Apple, which is a company of electronics, which is a category of shopping

- **Incorporating constraints**
  - E.g., the overall time span for a sequence can not be greater than a week

- **Try to think of similar twists in your projects!**
Outline

• Beyond item sets
  – Mining Sequential Patterns
  – Mining Graph Patterns

• Quiz + Break

• Clustering, if time permits
Graph Pattern Mining

- **CloseGraph: Mining Closed Frequent Graph Patterns**, Xifeng Yan and Jiawei Han, SIGKDD 2003

- **Graph pattern is even more challenging to mine**
  - For each itemset of k items
    - k! sequences
    - \(2^k\) graph patterns

- **However, it is widely encountered in practice**
  - Chemical/Molecular structure
  - Social network
Subgraph

- Intuitively: a graph $G_1$ is a subgraph of $G_2$ if all nodes and edges of $G_1$ appears in $G_2$

- Proper subgraph: there is at least one node/edge in $G_2$ that does not appear in $G_1$
Frequent Graph Patterns

• Given a graph database $D$ and a support threshold $t$:
  - A graph pattern $g$ is frequent if the fraction of graphs in $D$, of which $g$ is a subgraph of, is above $t$

• Closed Frequent Graph Patterns
  - A graph pattern $g$ is \textit{closed frequent} if there is no other graph pattern $g_1$, of which $g$ is a proper subgraph of, such that $g_1$ has the same support as $g$
  - Maximality is not considered here
Basic Steps

• **We are now very familiar with this!**
  – Start by identifying all single-node (zero-edge) frequent graph pattern
  – At each iteration where we have k-edge frequent graph pattern:
    • Generate (k+1)-edge candidate graph patterns
    • Examine the graph database to prune away infrequent candidate graph patterns

• **The main technical component is in how to extend from a k-edge graph pattern to a (k+1)-edge graph pattern**
Naïve Extension

Input: A graph $g$, a graph dataset $D$, and $\text{min\_sup}$.  
Output: The frequent graph set $S$.

1: if $g$ exists in $S$ then return;  
2: else insert $g$ to $S$;  
3: scan $D$ once, find every edge $e$ such that $g$ can be extended to $g \diamond_x e$ and it is frequent;  
4: for each frequent $g \diamond_x e$ do  
5: Call NaïveGraph($g \diamond_x e$, $D$, $\text{min\_sup}$, $S$);  
6: return;

- Serious problem: lots of redundant graph patterns being generated!
Forward and Backward Edges

- Depth-first traversal to construct a spanning tree of the graph can lead to two sets of edges
  - Those in the traversal – forward edges
  - Those not in the traversal – backward edges

- Easy to spot if the visit order of nodes are recorded
  - \((v_i, v_j)\) is a forward edge if \(i < j\)

Forward edges:
- \((x, x, a)\)
- \((x, y, a)\)
- \((x, z, b)\)

Backward edges:
- \((y, x, b)\)

DF traversal order:
- \(x, x, y, (x), z\)
Right-most Node and Edge

- Right-most node is the last node being visited in the DFS spanning tree traversal
- Right-most path is the direct path from root to the right-most node
Right-most Extension

- **Extension is needed only on the right-most path!**
  - Backward extension:
    - From the right-most vertex to any node on the right-most path
  - Forward extension:
    - From any node on the right-most path to a new node
  - See paper for proof
Challenge

• For a given graph pattern, how do we uniquely encode it and identify its right-most path?
**DFS Code**

- **DFS Code:**
  - Enforce a classic order for all the edges!
  - Forward edges appear in their natural order
  - A backward edge for a node must appear before any forward edge for that node
  - If there is no forward edge for a node, it’s backward edge must appear right after the node is first introduced

```
(v_0, v_1), (v_1, v_2), (v_1, v_3)
(v_0, v_1), (v_1, v_2), (v_2, v_0), (v_1, v_3)
```
Minimum DFS Code

• Each graph pattern can have multiple DFS codes
  – When the labels of nodes and edges are considered, we can have a unique minimum DFS code!
  – (first vertex label, edge label, second vertex label)

DFS Code 1:
- (x, a, x)
- (x, a, y)
- (y, b, x)
- (x, b, z)

DFS Code 2:
- (x, a, x)
- (x, b, y)
- (y, a, x)
- (x, b, z)
Selecting the minimum DFS code

- **Step 1:** select the nodes with the minimum label as the candidate roots
- **Step 2:** construct the DFS spanning tree from each root, always visit edges with smaller labels first; and if two edges with same label, visit the one whose second node has the smaller label
- **Step 3:** insert the backward edges
Right-most Extension

- **Ensures two things**
  - Avoid visiting patterns that have been visited before
  - No missing patterns that should have been visited

- **How do we leverage the closed property?**
Pruning

- If within the graph database D:
  - A graph $g_1$ always appears if graph $g_2$ appears, and $g_2$ is a proper subgraph of $g_1$
  - i.e., $g_1 \rightarrow g_2$ with 100% confidence

- Then, we may not have to extend from any graph pattern $g$ that contains $g_2$, but not $g_1$
  - Because those graph patterns will likely not lead to closed patterns, and we should directly extend from those patterns including $g_1$. 
Example

Graph DB

when (1) appears, (2) always appear

(3) is not closed, b/c (3) and (4) has same support
Failure Case and Detection

- **Solution:** mark \((y, b, x)\) *breakable*
  - Detecting forward extension and backward extension that creates two edges with same label
  - Remove the breakable edge for the forward extension and continue

However, this pattern can not be ignored!
Algorithm CloseGraph

Input: A DFS code $s$, its parent $p$, a graph dataset $D$, and $\text{min\_sup}$. 
Output: The closed frequent graph set $S$.

1: if $s \neq \text{min}(s)$, then 
2: return;
3: if $\exists e', g' = g_p \odot_x e'$ and $g' < g_s$ and $\mathcal{L}(g_p, D) = \mathcal{L}(g_p, g', D)$ and $g_p$ is not a failure case of early termination then
4: return;
5: set $C$ to $\emptyset$;
6: scan $D$ once, find every edge $e$ such that 
   $s$ can be extended to frequent $s \odot_x e$; 
   insert $s \odot_x e$ into $C$;
7: detect any possible failure of early termination in $s$;
8: if $\# s \odot_x e \in C$, $\text{support}(s) = \text{support}(s \odot_x e)$ then
9: insert $s$ into $S$;
10: remove $s \odot_x e$ from $C$ which cannot be 
    right-most extended from $s$;
11: sort $C$ in DFS lexicographic order;
12: for each $s \odot_r e$ in $C$ do
13: Call CloseGraph($s \odot_r e$, $s$, $D$, $\text{min\_sup}$, $S$);
14: return;
Graph Evolution Mining

  – Frequent pattern mining approach
**Input/Output**

- **Input:**
  - A sequence of snapshots of a social network
  - A minimum support threshold \( s \)
  - A minimum confidence threshold \( c \)

- **Output:**
Pattern Encoding

• Using relative time pattern to consolidate snapshots
Problem with Support

- We are now trying to determine the support of a pattern in a single graph
  - Previously, we were trying to determine the support of a pattern in a database of graphs
  - Number of occurrences is not monotonic!
Minimum Image based Support

• For each pattern node $n$, count the number of unique nodes $n$ maps to in the graph, chose the least number as the support.
Algorithm

- Any graph mining algorithm can then be applied
  - E.g., CloseGraph

- Graph mining is expensive
  - Recent research direction has been moving toward distributed processing
Quiz + 10 min Break
Outline

• Beyond item sets
  – Mining Sequential Patterns
  – Mining Graph Patterns

• Quiz + Break

• Overview of Clustering
Reading Material

• Data Clustering: 50 Years Beyond K-Means. AK Jain. *Pattern Recognition Letters, 2009*
Cluster Analysis

- **Unsupervised** classification of data items into groups
  - Minimize distances within a cluster
  - Maximize distances across clusters
What is NOT Clustering

- Simple segmentation based on known criteria
  - Grouping people according to their income

- Classification
  - Given a set of labels, predict the label of an unlabeled item
  - Usually a machine learning model is trained
It is a HARD Problem

How many clusters?

Six Clusters

Two Clusters

Four Clusters
Stages

Patterns → Feature Selection/Extraction → Pattern Representations → Interpattern Similarity → Grouping → Clusters

feedback loop
Item Representation

- A set of features that represent the data items to be clustered
  - Numerical features
    - Continuous (price)
    - Discrete (HDD capacity)
    - Interval (office hour)
  - Categorical features
    - Nominal (color)
    - Ordinal: similar to discrete numerical features

- How to represent an items is as much as an art as a science, domain expertise is required!
Similarity Measures

- **Intrinsic measure (depends on participating items only)**
  - Euclidean distance and its extensions
    \[
    d_2(x_i, x_j) = \left( \sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2 \right)^{1/2}
    \]

- **Contextual measure (e.g., incorporating neighboring items)**
  - Mutual Neighbor Distance, \( NN(x,y) \) = where y ranks as \( x \)'s nearest neighbor
    \[
    MND(x_i, x_j) = NN(x_i, x_j) + NN(x_j, x_i),
    \]
MND Example

Use intrinsic measures to compute nearest neighbors

MND(A,B) = 1 + 1 = 2
(without DEF)

MND(A,B) = 4 + 1 = 5
(with DEF)
Metric

- A similarity measure is a metric if it satisfies the triangle inequality
  \[ d(a,b) + d(b,c) > d(a,c) \]

- **Metric measures**
  - Euclidean distance
  - Cosine similarity

- **MND is not**
Clustering Techniques

• **Main approaches:**
  - Partitional Clustering
    • Each item ends up in one cluster
  - Hierarchical Clustering
    • Each item ends up in a list of hierarchical clusters

• **Other distinctions**
  - Agglomerative (bottom up) vs. Divisive (top down)
  - Hard (whole membership) vs. Fuzzy (fractional membership)
  - Incremental vs. Non-incremental
Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree-like diagram that records the sequences of merges or splits
Agglomerative Hierarchical Clustering

- Most popular clustering technique
- Basic algorithm is straightforward
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. Repeat
  4. Merge the two closest clusters
  5. Update the proximity matrix
  6. Until only a single cluster remains

- Key operation is the computation of the proximity of two clusters
  - Different approaches to defining the distance between clusters distinguish the different algorithms
Similarity between Two Clusters

- MIN (Single-Link)
- MAX (Complete-Link)
- Average
- Distance Between Centroids
- Other methods driven by an objective function
  - E.g., minimize the square error
Strength of Single-Link

- Handles different shapes when the items are separated well
Limitation of Single-Link

- Sensitive to noise and outliers

Original Points

Two Clusters
Strength of Complete-Link

- Less susceptible to noise and outliers
Limitation of Complete-Link

- Tends to break large clusters
- Biased towards globular clusters
Hierarchical Clustering: Time and Space Requirements

- **$O(N^2)$ space** since a proximity matrix needs to be maintained
  - $N$ is the number of points.

- **$O(N^3)$ time** in many cases
  - There are $N$ steps and at each step the size, $N^2$, proximity matrix must be updated and searched
  - Complexity can be reduced to $O(N^2 \log(N))$ time for some approaches
K-Means Clustering

- Number of desired clusters $K$ is pre-specified
- At each iteration, items are re-assigned to centroids
- Convergence happen for most similarity measures and often happen fast
- Computing centroids is conceptually less ambiguous than computing distance between two clusters

**Algorithm 1** Basic K-means Algorithm.

1. Select $K$ points as the initial centroids.
2. repeat
3. Form $K$ clusters by assigning all points to the closest centroid.
4. Recompute the centroid of each cluster.
5. until The centroids don’t change
Initial Centroids Selection

• Usually randomly chosen
  – Centroids may adjust themselves in the right way

• Often multiple sets initial centroids are chosen and multiple runs are done
  – Pick the run that produces the best results

• Using other clustering algorithms to determine initial centroids

• Select more than K and eliminate the ones that are redundant with others along the way

• Bisecting K-Means
Starting with two initial centroids in one cluster of each pair of clusters
Limitation of K-Means

Original Points

K-means (3 Clusters)
Summary

• We covered more data mining
  – Sequential pattern mining
  – Graph pattern mining

• Brief overview of clustering techniques
Questions?