Data Mining Part I: “Fast Algorithms for Mining Association Rules”  
Agrawal R., Srikant R.  
(Thanks to previous student Dan Li for many of the slide points.)

The Problem: Discovering Associated Purchases  
► You have a database of customer transactions  
► You want to find out which products customers buy at the same time  
► Store owners might use this info to:  
  ▪ Organizing items together in catalogs/flyers  
  ▪ Figure out best arrangement of products on the shelves  
  ▪ etc.

The Problem: Notation  
► You have a set of items: A, B, C, D, ....  
► You have a set of transactions:  
  1) {A, B, D}  
  2) {B, C, D}  
  3) {C, D, J, R, V}  
  4) etc.  
► You want to find association rules, e.g. {A, B} => {D, R}

Support & Confidence  
► Consider a rule {A, B} => {D, R}  
► The support $s$ is for this rule is:  
  ▪ The percentage of transactions that contain {A, B, D, R}  
► The confidence $c$ for this rule is:  
  ▪ The percentage of transactions containing {A, B} that also contain {D, R}  
► Sets with support > $s$ are called large sets

The Problem  
► Given a list of transactions  
► Find all rules with support > $s$ and confidence > $c$

Discussion  
► When generating association rules, we can set a desired support level and a desired confidence level.  
► What considerations are necessary when setting values for both?  
► For what applications would you choose a high confidence value? A high support value?
Important Observation about "Support"

- If a set X has support > s, then every subset of X has support > s

Example:
- Suppose there are 3 large items in the transaction list: \{A\}, \{B\}, \{C\}
- Only possible sets of size 2 are \{A, B\}, \{A, C\}, \{B, C\}
- Only possible set of size 3 is \{A, B, C\}

Problem Decomposition

- Paper breaks the problem into 2 parts:
  - Part 1: Find: All sets with support > s
  - Part 2: Given solution to Part 1
    - Find all rules with support > s and confidence > c
- This paper solves Part 1 only.
  - But Part 2 is much easier than Part 1.

The Apriori Algorithm

- Basic outline is easy to understand. The hard part is generating "candidates" (apriori-gen)

Generating Candidate Sets: apriori-gen

- Insert into \(C_k\)
- Select \(p_i, \ldots, p_i, q_i, \ldots, q_i\) from \(L_{k-1}\)
- Where \(p_i = q_i = \ldots = q_i\)
- \(p_i < q_i\) and \(q_i < q_i\)

- To make a candidate of size k, join two large sets of size \((k - 1)\) which differ only in their last element
- “But why?”, you ask.

Explanation of apriori-gen

- Suppose you want to generate \(C_3\) from

```
\begin{array}{c|c}
L_1 & L_2 \\
\hline
\{A\} & \{A, B\} \\
\{B\} & \{A, E\} \\
\{O\} & \{B, D\} \\
\{E\} & \\
\end{array}
```

- Could generate \(C_3\) by combining each set from \(L_1\) with each set from \(L_2\)
- e.g. \(\{A, B\} \cup \{D\} = \{A, B, D\}\)
- However, notice that in order for \(\{A, B, D\}\) to be large, \(\{A, D\}\) must also be large.

Explanation of apriori-gen

In general, suppose we have a set
\(a = \{i_1, i_2, \ldots, i_{k-1}\}\)
and we extend it with an item X:
\(a' = \{i_1, i_2, \ldots, i_{k-1}, X\}\)
\(a'\) cannot be large unless \(\{i_1, i_2, \ldots, i_{k-2}, X\}\) is large.

Therefore, generate candidates of size k by merging all pairs \(\{i_1, i_2, \ldots, i_{k-2}, X\}\) and \(\{i_1, i_2, \ldots, i_{k-1}, Y\}\) from \(L_{k-1}\).
Apriori-gen: The Prune Step

- Look at each candidate of size k generated by the join
- Check that each subset of size k-1 is large (if not, throw it away)

Apriori Algorithm: Example

Suppose the user specifies a minimum support of 20% and we have the transaction table:

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{A, C, D, E}</td>
</tr>
<tr>
<td>2</td>
<td>{A, B, C}</td>
</tr>
<tr>
<td>3</td>
<td>{B, C, E}</td>
</tr>
<tr>
<td>4</td>
<td>{A, B, D, E}</td>
</tr>
<tr>
<td>5</td>
<td>{C, E}</td>
</tr>
</tbody>
</table>

Since there are 5 transactions, support of 20% means 2 or more occurrences.
Discussion

This paper has spun off more similar algorithms in the database world than any other data mining algorithm.

Why do you think this paper is so influential?
- Is it the context of association rule mining?
- The way they approach the problem?
- The algorithm itself?
- Its performance?

What is Data Clustering?

You are given:
- n points in d-dimensional space
- a distance function f(a,b)
- a desired number of clusters, k

You want to find:
- a partitioning that minimizes the "size" of the clusters
- several ways to measure "size" (e.g., average distance between pairs of points in a cluster)
Discussion

The BIRCH paper gave the example of clustering 2D image features into five clusters.

Can you think of other large datasets where discovering clusters would be useful? What constraints does this data pose on the resources required?

The Goal of the BIRCH Algorithm

Efficient clustering of large datasets (larger than memory, that is)

Minimize disk I/O’s

BIRCH can be seen as a "helper" algorithm enables standard clustering algorithms to run on very large datasets

Advantages of BIRCH vs. Other Clustering Algorithms

1) It is "local".
   - i.e. each time a new point is added, it is only compared against a subset of the other points in the dataset
2) There is a mechanism for removing outliers.
3) BIRCH minimizes I/O costs. Also, adjusts the quality of results to the amount of available memory.
4) It only scans the dataset once (If phase 4 is omitted).

Clustering Feature (CF)

Compact - no need to store the individual points belonging to a cluster.

Three parts:
- \(N\), the number of points in the cluster
- \(LS\), the sum of the points in the cluster
- \(SS\), the sum of the points squared

This info is sufficient to compute the distance between two clusters

When merging two clusters, can just add CFs

CF TREE

The CF Tree is a hierarchy of clusters
Each node contains a list of CFs
\(T\) is the threshold for the diameter of the leaf nodes
Data items are scanned and inserted into the CF tree, one at a time.

CF Tree Insertion

To identify the appropriate leaf:
- Starting with CF list at the root node, find the closest cluster (by using the CF values)
- Look at all the children of this cluster, find the closest.
- And so on, until you reach a leaf node.

Once the point has been added, must update the CF of all ancestors
Leaves have a max size, so they must sometimes must be split
The BIRCH Algorithm: 4 Phases

**Phase 1:** Scan all data and build an initial in-memory CF tree.

**Phase 2:** Shrink the tree as required for Phase 3.

**Phase 3:** Run a standard ("global") clustering algorithm on the leaf clusters.

**Phase 4:** Reassign individual data points to the clusters.

BIRCH Phase 1

- Start with initial threshold \( T \) and insert points into the tree
- If we run out of memory, increase \( T \), and rebuild
  - Take leaf entries from original tree and re-insert into new tree
  - This is an opportunity to remove outliers
- Methods for initializing and adjusting \( T \) are ad hoc

Important Point:

- After Phase 1, the data has been "shrunk" to fit in memory.
- Subsequent phases of processing happen entirely in memory (no disk I/Os)

BIRCH Phase 2

- Optional.
- Number of clusters produced in Phase 1 may be larger than Phase 3 can handle.
- Shrink tree as necessary.

BIRCH Phase 3

- Use the leaf nodes of the CF tree as input to a standard ("global") clustering algorithm.
- Phase 1 has reduced the size of the input dataset enough so that the standard algorithm can work entirely in memory.

BIRCH Phase 4

- Optional.
- Scan through the data again, assign each data point to a cluster
  - Choose the cluster whose centroid is closest.
- This redistributes the points among clusters, in a more accurate fashion than the original CF tree

Discussion

- If you had to design a data mining algorithm for your data, which of the following criteria would you consider most important?
  - Average running time?
  - I/O cost?
  - Memory efficiency?
  - Scalability?
  - Robustness to noise?
  - Parameter tuning?
- What are the trade-offs between your choice and the other factors? How much accuracy are you willing to sacrifice?
Applications of Data Clustering

Helps understand the natural groupings that exist inside a dataset.

Examples:
- Market analysis: determining groups of customers with similar tastes
- Bioinformatics: determining groups of molecules with similar functions in the cell
- Insurance: identifying high-risk groups of policy holders