Low-Support, High-Correlation

Finding Rare but Similar Items
Minhashing
Locality-Sensitive Hashing
The Problem

- Rather than finding high-support item-pairs in basket data, look for items that are highly “correlated.”
  - If one appears in a basket, there is a good chance that the other does.
  - “Yachts and caviar” as itemsets: low support, but often appear together.
Correlation Versus Support

- A-Priori and similar methods are useless for low-support, high-correlation itemsets.
- When support threshold is low, too many itemsets are frequent.
  - Memory requirements too high.
- A-Priori does not address correlation.
Matrix Representation of Item/Basket Data

- **Columns** = items.
- **Rows** = baskets.
- **Entry** \((r, c)\) = 1 if item \(c\) is in basket \(r\); = 0 if not.
- Assume matrix is almost all 0’s.
### In Matrix Form

<table>
<thead>
<tr>
<th></th>
<th>m</th>
<th>c</th>
<th>p</th>
<th>b</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>{m,c,b}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{m,p,b}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{m,b}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{c,j}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{m,p,j}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{m,c,b,j}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>{c,b,j}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>{c,b}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Applications --- (1)

◆ Rows = customers; columns = items.
  ✷ \((r, c) = 1\) if and only if customer \(r\) bought item \(c\).
  ✷ Well correlated columns are items that tend to be bought by the same customers.
  ✷ Used by on-line vendors to select items to “pitch” to individual customers.
Applications --- (2)

- **Rows** = (footprints of) shingles; **columns** = documents.
  - $(r, c) = 1$ iff footprint $r$ is present in document $c$.
  - Find similar documents, as in Anand’s 10/10 lecture.
Applications --- (3)

- Rows and columns are both Web pages.
  - $(r, c) = 1$ iff page $r$ links to page $c$.
  - Correlated columns are pages with many of the same in-links.
  - These pages may be about the same topic.
Assumptions --- (1)

1. Number of items allows a small amount of main-memory/item.
   - E.g., main memory = \text{Number of items} \times 100

2. Too many items to store anything in main-memory for each \textit{pair} of items.
Assumptions --- (2)

3. Too many baskets to store anything in main memory for each basket.

4. Data is very sparse: it is rare for an item to be in a basket.
From Correlation to Similarity

◆ Statistical correlation is too hard to compute, and probably meaningless.
  ◆ Most entries are 0, so correlation of columns is always high.

◆ Substitute “similarity,” as in shingles-and-documents study.
Similarity of Columns

◆ Think of a column as the set of rows in which it has 1.

◆ The similarity of columns $C_1$ and $C_2 = Sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = Jaccard measure$. 

Jaccard measure
Example

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$\text{Sim} \ (C_1, \ C_2) = \frac{2}{5} = 0.4$
Outline of Algorithm

1. Compute “signatures” ("sketches") of columns = small summaries of columns.
   - Read from disk to main memory.

2. Examine signatures in main memory to find similar signatures.
   - Essential: similarity of signatures and columns are related.

3. Check that columns with similar signatures are really similar (optional).
Signatures

Key idea: “hash” each column $C$ to a small signature $\text{Sig} (C)$, such that:

1. $\text{Sig} (C)$ is small enough that we can fit a signature in main memory for each column.
2. $\text{Sim} (C_1, C_2)$ is the same as the “similarity” of $\text{Sig} (C_1)$ and $\text{Sig} (C_2)$. 
An Idea That Doesn’t Work

- Pick 100 rows at random, and let the signature of column \( C \) be the 100 bits of \( C \) in those rows.
- Because the matrix is sparse, many columns would have 00...0 as a signature, yet be very dissimilar because their 1’s are in different rows.
Four Types of Rows

Given columns $C_1$ and $C_2$, rows may be classified as:

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Also, $a = \# \text{ rows of type } a$, etc.

Note $Sim(C_1, C_2) = a / (a + b + c)$. 
Minhashing

- Imagine the rows permuted randomly.
- Define “hash” function $h(C) =$ the number of the first (in the permuted order) row in which column $C$ has 1.
- Use several (100?) independent hash functions to create a signature.
Minhashing Example

Input matrix

<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

1 0 1 0
1 0 0 1
0 1 0 1
0 1 0 1
0 1 0 1
1 0 1 0
1 0 1 0

Signature matrix $M$

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

19
The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.

Both are $a/(a+b+c)!$.

Why?

- Look down columns $C_1$ and $C_2$ until we see a 1.
- If it’s a type $a$ row, then $h(C_1) = h(C_2)$. If a type $b$ or $c$ row, then not.
Similarity for Signatures

The similarity of signatures is the fraction of the rows in which they agree.
- Remember, each row corresponds to a permutation or “hash function.”
**Min Hashing – Example**

### Input matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Signature matrix $M$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Similarities:

- Col.-Col.: $0.75$ $0.75$ $0$ $0$
- Sig.-Sig.: $0.67$ $1.00$ $0$ $0$
Minhash Signatures

- Pick (say) 100 random permutations of the rows.
- Think of $\text{Sig}(C)$ as a column vector.
- Let $\text{Sig}(C)[i] = \text{row number of the first row with 1 in column } C$, for $i$th permutation.
Implementation --- (1)

- Number of rows = 1 billion.
- Hard to pick a random permutation from 1...billion.
- Representing a random permutation requires billion entries.
- Accessing rows in permuted order is tough!
  - The number of passes would be prohibitive.
Implementation --- (2)

1. Pick (say) 100 hash functions.
2. For each column $c$ and each hash function $h_i$, keep a “slot” $M(i, c)$ for that minhash value.
3. for each row $r$, and for each column $c$ with 1 in row $r$, and for each hash function $h_i$ do
   if $h_i(r)$ is a smaller value than $M(i, c)$ then
   $M(i, c) := h_i(r)$.

Needs only one pass through the data.
### Example

<table>
<thead>
<tr>
<th>Row</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$h(x) = x \mod 5$

- $h(1) = 1$
- $g(1) = 3$
- $h(2) = 2$
- $g(2) = 0$
- $h(3) = 3$
- $g(3) = 2$
- $h(4) = 4$
- $g(4) = 4$
- $h(5) = 0$
- $g(5) = 1$

$g(x) = 2x+1 \mod 5$

- $g(1) = 3$
- $g(2) = 0$
- $g(3) = 2$
- $g(4) = 4$
- $g(5) = 1$
Comparison with “Shingling”

◆ The shingling paper proposed using one hash function and taking the first 100 (say) values.

◆ Almost the same, but:
  - Faster --- saves on hash-computation.
  - Admits some correlation among rows of the signatures.
Candidate Generation

- Pick a similarity threshold $s$, a fraction $< 1$.
- A pair of columns $c$ and $d$ is a candidate pair if their signatures agree in at least fraction $s$ of the rows.
  - I.e., $M(i, c) = M(i, d)$ for at least fraction $s$ values of $i$. 
The Problem with Checking Candidates

- While the signatures of all columns may fit in main memory, comparing the signatures of all pairs of columns is quadratic in the number of columns.

- **Example**: $10^6$ columns implies $5 \times 10^{11}$ comparisons.

- At 1 microsecond/comparison: 6 days.
Solutions

1. DCM method (Anand’s 10/10 slides) relies on external sorting, so several passes over the data are needed.

2. Locality-Sensitive Hashing (LSH) is a method that can be carried out in main memory, but admits some false negatives.
Locality-Sensitive Hashing

- Unrelated to “minhashing.”
- Operates on signatures.
- **Big idea**: hash columns of signature matrix $M$ several times.
- Arrange that similar columns are more likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.
Partition into Bands

Divide matrix $M$ into $b$ bands of $r$ rows.

For each band, hash its portion of each column to $k$ buckets.

Candidate column pairs are those that hash to the same bucket for $\geq 1$ band.

Tune $b$ and $r$ to catch most similar pairs, few nonsimilar pairs.
Simplifying Assumption

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band.

- Hereafter, we assume that “same bucket” means “identical.”
Matrix $M$ with $r$ rows and $b$ bands.

Buckets
Example

◆ Suppose 100,000 columns.
◆ Signatures of 100 integers.
◆ Therefore, signatures take 40Mb.
◆ But 5,000,000,000 pairs of signatures can take a while to compare.
◆ Choose 20 bands of 5 integers/band.
Suppose $C_1$, $C_2$ are 80% Similar

$\blacklozenge$ Probability $C_1$, $C_2$ identical in one particular band: $(0.8)^5 = 0.328$.

$\blacklozenge$ Probability $C_1$, $C_2$ are not similar in any of the 20 bands: $(1-0.328)^{20} = 0.00035$.

$\blacklozenge$ i.e., we miss about $1/3000$th of the 80%-similar column pairs.
Suppose $C_1, C_2$ Only 40% Similar

- Probability $C_1, C_2$ identical in any one particular band: $(0.4)^5 = 0.01$.
- Probability $C_1, C_2$ identical in $\geq 1$ of 20 bands: $\leq 20 \times 0.01 = 0.2$.
- Small probability $C_1, C_2$ not identical in a band, but hash to the same bucket.
- But false positives much lower for similarities $< 40\%$. 
Example Target: All pairs with $Sim > 60\%$.

Suppose we use only one hash function:

\[ 1 - (1 - s^r)^b \]

LSH (partition into bands) gives us:
LSH Summary

- Tune to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.
- Check in main memory that candidate pairs really do have similar signatures.
- Then, in another pass through data, check that the remaining candidate pairs really are similar *columns*. 
New Topic: Hamming LSH

- An alternative to minhash + LSH.
- Takes advantage of the fact that if columns are not sparse, random rows serve as a good signature.
- **Trick**: create data matrices of exponentially decreasing sizes, increasing densities.
Amplification of 1’s

- **Hamming LSH** constructs a series of matrices, each with half as many rows, by OR-ing together pairs of rows.
- Candidate pairs from each matrix have (say) between 20% - 80% 1’s and are similar in selected 100 rows.
  - 20%-80% OK for similarity thresholds $\geq 0.5$. Otherwise, two “similar” columns could fail to both be in range for at least one matrix.
Example
Using Hamming LSH

◆ Construct the sequence of matrices.
  ♦ If there are \( R \) rows, then \( \log_2 R \) matrices.
  ♦ Total work = twice that of reading the original matrix.

◆ Use standard LSH to identify similar columns in each matrix, but restricted to columns of “medium” density.