Locality-Sensitive Hashing

Basic Technique

Hamming-LSH

Applications
Finding Similar Pairs

- Suppose we have in main memory data representing a large number of objects.
  - May be the objects themselves (e.g., summaries of faces).
  - May be signatures as in minhashing.
- We want to compare each to each, finding those pairs that are sufficiently similar.
Candidate Generation From Minhash Signatures

- Pick a similarity threshold \( s \), a fraction \(< 1 \).
- A pair of columns \( c \) and \( d \) is a \textit{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 \( i \).
Candidate Generation --- (2)

- For images, a pair of vectors is a candidate if they differ by at most a small threshold $t$ in at least $s\%$ of the components.
- For entity records, a pair is a candidate if the sum of similarity scores of corresponding components exceeds a threshold.
The Problem with Checking for 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. *Divide-Compute-Merge* (DCM) uses external sorting, merging.

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

3. *Hamming LSH* --- a variant LSH method.
Divide-Compute-Merge

- Designed for “shingles” and docs.
- At each stage, divide data into batches that fit in main memory.
- Operate on individual batches and write out partial results to disk.
- Merge partial results from disk.
DCM Steps

Invert

sort on shingleId

Invert and pair

Merge
DCM Summary

1. Start with the pairs <shingleId, docId>.
2. Sort by shingleId.
3. In a sequential scan, generate triplets <docId1, docId2, 1> for pairs of docs that share a shingle.
4. Sort on <docId1, docId2>.
5. Merge triplets with common docIds to generate triplets of the form <docId1, docId2, count>.
Some Optimizations

◆ “Invert and Pair” is the most expensive step.
◆ Speed it up by eliminating very common shingles.
  ♦ “the”, “404 not found”, “<A HREF”, etc.
◆ Also, eliminate exact-duplicate docs first.
Locality-Sensitive Hashing

- **Big idea**: hash columns of signature matrix $M$ several times.
- Arrange that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.
Partition Into Bands

Matrix $M$

$b$ bands

$r$ rows per band
Partition into Bands --- (2)

- Divide matrix $M$ into $b$ bands of $r$ rows.
- For each band, hash its portion of each column to a hash table with $k$ 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, but few nonsimilar pairs.
Matrix M with r rows and b bands connected to buckets.
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.”
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

- Probability $C_1, C_2$ identical in one particular band: $(0.8)^5 = 0.328$.
- Probability $C_1, C_2$ are *not* similar in any of the 20 bands: $(1-0.328)^{20} = 0.00035$.
  - i.e., we miss about 1/3000th 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: $\leq 20 \times 0.01 = 0.2$.
- But false positives much lower for similarities $<< 40\%$. 
LSH Involves a Tradeoff

- Pick the number of minhashes, the number of bands, and the number of rows per band to balance false positives/negatives.

- **Example**: if we had fewer than 20 bands, the number of false positives would go down, but the number of false negatives would go up.
Example Target: All pairs with $Sim > t$.

Suppose we use only one hash function:

Partition into bands gives us:

$$1 - (1 - s^r)^b$$

$$t \sim (1/b)^{1/r}$$
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.

◆ **Optional**: 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 with widely differing numbers of 1’s 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 on a random selection of rows to identify similar columns in each matrix, but restricted to columns of “medium” density.
LSH for Other Applications

1. Face recognition from 1000 measurements/face.

2. Entity resolution from name-address-phone records.

◆ General principle: find many hash functions for elements; candidate pairs share a bucket for $\geq 1$ hash.
Face-Recognition Hash Functions

1. Pick a set of $r$ of the 1000 measurements.
2. Each bucket corresponds to a range of values for each of the $r$ measurements.
3. Hash a vector to the bucket such that each of its $r$ components is in-range.
4. **Optional**: if near the edge of a range, also hash to an adjacent bucket.
Example: \( r = 2 \)

One bucket, for \((x, y)\) if \(10 \leq x \leq 16\) and \(0 \leq y \leq 4\)

<table>
<thead>
<tr>
<th></th>
<th>10-16</th>
<th>17-23</th>
<th>24-30</th>
<th>31-37</th>
<th>38-44</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15-19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(27,9) goes here.

Maybe put a copy here, too.
Many-One Face Lookup

- As for boolean matrices, use many different hash functions.
  - Each based on a different set of the 1000 measurements.
- Each bucket of each hash function points to the images that hash to that bucket.
Face Lookup --- (2)

- Given a new image (the *probe*), hash it according to all the hash functions.
- Any member of any one of its buckets is a candidate.
- For each candidate, count the number of components in which the candidate and probe are close.
- Match if \#components \(\geq\) threshold.
Hashing the Probe

Look in all these buckets
Many-Many Problem

- Make each pair of images that are in the same bucket according to any hash function be a candidate pair.
- Score each candidate pair as for the many-one problem.
Entity Resolution

◆ You don’t have the convenient multidimensional view of data that you do for “face-recognition” or “similar-columns.”

◆ We actually used an LSH-inspired simplification.
Entity Resolution --- (2)

◆ Three hash functions:
  1. One bucket for each name string.
  2. One bucket for each address string.
  3. One bucket for each phone string.

◆ A pair is a candidate iff they mapped to the same bucket for at least one of the three hashes.