Finding Similar Items

Juliana Freire

Modified from Jeff Ullman, Jure Lescovek
Goals

• Many Web-mining problems can be expressed as finding “similar” sets
  
• Find near-neighbors in high-dimensional space

• Examples
  1. Pages with similar words, e.g., for duplicate detection, classification by topic.
  2. Customers who purchased similar products, e.g., similar tastes in movies, for recommendation systems.
  3. Dual: products bought with similar customer sets
  4. Images with similar features.
Similarity Algorithms

• The best techniques depend on whether you are looking for items that are very similar or only somewhat similar.

• We will cover the “somewhat” case.

• The textbook also covers the “very similar” case
Problem Definition

- **Given:** High dimensional data points $x_1, x_2, \ldots$
  - **For example:** Image is a long vector of pixel colors
    \[
    \begin{bmatrix}
    1 & 2 & 1 \\
    0 & 2 & 1 \\
    0 & 1 & 0
    \end{bmatrix} \rightarrow [1 2 1 0 2 1 0 1 0]
    \]
- **And some distance function** $d(x_1, x_2)$
  - Which quantifies the “distance” between $x_1$ and $x_2$
- **Goal:** Find all pairs of data points $(x_i, x_j)$ that are within some distance threshold $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take $O(N^2)$ ☹
  - where $N$ is the number of data points
- **MAGIC:** This can be done in $O(N)$!! How?
Goal: Find near-neighbors in high-dimension space

- We formally define “near neighbors” as points that are a “small distance” apart

- For each application, we first need to define what “distance” means

- For sets, we can use Jaccard distance/similarity
  - The Jaccard similarity of two sets is the size of their intersection divided by the size of their union:
    \[ sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \]
  - Jaccard distance: \( d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \)

\[ \begin{array}{c}
\text{3 in intersection} \\
\text{8 in union} \\
\text{Jaccard similarity} = \frac{3}{8} \\
\text{Jaccard distance} = \frac{5}{8}
\end{array} \]
Example: Comparing Documents

• **Goal**: *common text*, not common topic or meaning.
  • For the latter, other techniques can be applied, e.g., NLP

• Special cases are easy, e.g., identical documents, or one document contained character-by-character in another.

• General case, where many small pieces of one doc appear out of order in another, is very hard.
Finding Similar Documents

• Given a large number of documents, e.g., the Web, find pairs of documents with a lot of text in common, e.g.:
  • Mirror sites, or approximate mirrors.
    • Application: Don’t want to show both in a search.
  • Plagiarism, including large quotations.
  • Similar news articles at many news sites.
    • Application: Cluster articles by “same story.”
    • Application: Understand news propagation.

• Challenges:
  • Many pieces of a document can appear out of order in another
  • Too many documents to compare all pairs
  • Documents are so large or so many that they cannot fit in memory
Exploring a 'Deep Web' That Google Can't Grasp

One day last summer, Google's search engine trundled quietly past a milestone. It added the one trillionth address to the list of Web pages it knows about. But as impossibly big as that number may seem, it represents only a fraction of the entire Web.

Beyond those trillion pages lies an even vaster Web of hidden data: financial information, shopping catalogs, flight schedules, medical research and all kinds of other material stored in databases that remain largely invisible to search engines.

The challenges that the major search engines face in penetrating this so-called Deep Web are enormous. The search engines knew how to find them.

Beyond those trillion pages lie even more pages: hidden data that they cannot see. The search engines can't find them. Beyond those trillion pages lies an even vaster Web of hidden data: financial information, shopping catalogs, flight schedules, medical research and all kinds of other material stored in databases that remain largely invisible to search engines.

The challenges that the major search engines face in penetrating this so-called Deep Web are enormous. The search engines knew how to find them.

Beyond those trillion pages lie even more pages: hidden data that they cannot see. The search engines can't find them.

Emerging search technologies aim for Web's hidden depths

By Alex Wright

NEW YORK: One day last summer, Google's search engine trundled quietly past a milestone. It added the one trillionth address to the list of Web pages it knows about. But as impossibly big as that number may seem, it represents only a fraction of the entire Web.

Beyond those trillion pages lies an even vaster Web of hidden data: financial information, shopping catalogs, flight schedules, medical research and all kinds of other material stored in databases that remain largely invisible to search engines.

The challenges that the major search engines face in penetrating this so-called Deep Web are enormous. The search engines knew how to find them.

Now a new breed of technologies is taking shape that will extend the reach of search engines into the Web's hidden corners. That, in turn, will extend their usefulness.

Search engines rely on programs known as crawlers (or spiders) that gather information by following the trails of hyperlinks that are the Web's highways. While that approach works well for the pages that make up the surface Web, these programs have a harder time penetrating databases that are set up to respond to typed queries.

Many search engines include a powerful feature: the ability to search databases most likely to yield relevant information, then return an overview of the topic drawn from multiple sources.

"The best search engines try to help you find a needle in a haystack," says Anand Rajaraman, co-founder of Kosmia (www.kosmia.com), a Deep Web search startup whose investors include Jeffrey Bezos, chief executive of Amazon.com. Kosmia has developed software that matches searches with the databases most likely to yield relevant information, then returns an overview of the topic drawn from multiple sources.

Rajaraman says, "But what we're trying to do is help you explore the haystack." That haystack is infinitely large. With millions of databases connected to the Web, and endless possible permutations of search terms, there is simply no way for any search engine -- no matter how powerful -- to sift through every possible combination of data on the fly.

To extract meaningful data from the Deep Web, search engines have to analyze users' search terms and figure out how to break those queries into particular databases. For example, if a user types "Rambler" into an search engine, the search engine needs to know which databases are most likely to contain information about the car (like, say, museum catalogs or auction houses), and what kinds of queries those databases will accept..."
How to map this problem into a nearest-neighbor search?
Three Essential Techniques for Similar Documents

1. **Shingling**: convert documents, emails, etc., to sets.

2. **Minhashing**: convert large sets to short signatures, while preserving similarity.

3. **Locality-sensitive hashing**: focus on pairs of signatures likely to be similar.
   - Candidate pairs
The Big Picture

The set of strings of length $k$ that appear in the document

Signatures: short integer vectors that represent the sets, and reflect their similarity

Candidate pairs: those pairs of signatures that we need to test for similarity.
Documents as Sets

• Step 1: convert documents to sets

• Some approaches:
  • Doc = set of words in document
  • Doc = set of important words

• Need to account for ordering of words!
Shingles

• A $k$-shingle (or $k$-gram) for a document is a sequence of $k$ tokens that appears in the document.
  • Tokens can be characters, words, or something else depending on the application

• Example: $k=2$; $doc = abcab$. Set of 2-shingles $S(doc) = \{ab, bc, ca\}$.
  • Option: regard shingles as a bag, and count ab twice, $S'(doc) = \{ab, bc, ca. ab\}$.

• Represent a doc by its set of $k$-shingles.
Working Assumption

• Documents that have lots of shingles in common have similar text, even if the text appears in different order.
Shingle Size

• Is k=2 a good choice for size?
• **Example:** k=2;
• doc1 = abcab. 2-shingles = \{ab, bc, ca\}.
• doc2 = cabc. 2-shingles = \{ab, bc, ca\}.
Shingle Size

- **Careful**: you must pick $k$ large enough, or most documents will have most shingles.

- The probability of any given shingle appearing in any given document should be low

- $k = 5$ is OK for short documents; $k = 9$ is better for long documents.

- Assume $k=5$, then there are $27^5 = 14,348,907$ possible shingles
Compressing Shingles

• If k=9, to compare shingles we need to compare 9 bytes.
• To improve efficiency, we can compress long shingles: hash them to (say) 4 bytes – 1 word, and
• Represent a doc by the set of hash values of its $k$-shingles.

$(aaabbbccc)(abcabcabc) \rightarrow h(aaabbbccc)h(abcabcabc)$

18 bytes $\rightarrow$ 8 bytes

• Example: $k=2$; doc = abcab.
  • Set of 2-shingles $S(doc) = \{ab, bc, ca\}$.
  • Hash of the shingles: $h(doc) = \{1,5,7\}$.
Thought Question

• Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?

• **Hint:** How random are the 32-bit sequences that result from 4-shingling?
Thought Question

• Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?

• **Hint**: How random are the 32-bit sequences that result from 4-shingling?

• Assuming 20 characters are common in English, there are $(20)^4 = 160,000$ 4-shingles

• Using 9 shingles there are $(20)^9 \gg 2^{32}$
Motivation for Minhash/LSH

• Suppose we need to find near-duplicate documents among N = 1 million documents

• Naïvely, we would have to compute similarities for each pair of documents
  • N(N-1)/2 = ~ 5 * 10^{11} comparisons
  • At 10^5 sec/day and 10^6 comparisons/sec, it would take 5 days!
  • For N=10 million, it takes more than a year

• You can use more hardware, or you can be smarter ;-)}
MinHashing

- Idea: convert large sets to short signatures while preserving similarities
- Data as Sparse Matrices
- Jaccard Similarity Measure
- Constructing Signatures
Basic Data Model: Sets

• Many similarity problems can be couched as finding subsets of some universal set that have significant intersection.

• **Examples** include:
  1. Documents represented by their sets of shingles (or hashes of those shingles).
  2. Similar customers or products.
Jaccard Similarity of Sets

- The *Jaccard similarity* of two sets is the size of their intersection divided by the size of their union.

- \( \text{Sim} (C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}. \)
Example: Jaccard Similarity

3 in intersection.
8 in union.
Jaccard similarity = 3/8
From Sets to Boolean Matrices

- **Rows** = elements of the universal set.
- **Columns** = sets.
- 1 in row $e$ and column $S$ if and only if $e$ is a member of $S$.
- Column similarity is the Jaccard similarity of the sets of their rows with 1.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**
- Example: $C_1 = 10111$; $C_2 = 10011$
  - Size of intersection = 3 ($C_1$ AND $C_2$); size of union = 4 ($C_1$ OR $C_2$)
- Jaccard similarity (not distance) = $3/4$
- Distance: $d(C_1, C_2) = 1 - \text{(Jaccard similarity)} = 1/4$
From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row \( e \) and column \( s \) if and only if \( e \) is a member of \( s \)
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**
- **Each document is a column:**
  - **Example:** \( \text{sim}(C_1, C_2) = ? \)
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = 3/6
    - \( d(C_1, C_2) = 1 - \text{(Jaccard similarity)} = 3/6 \)

<table>
<thead>
<tr>
<th></th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<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></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Shingles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
• We might not really represent the data by a boolean matrix.

• Sparse matrices are usually better represented by the list of places where there is a non-zero value.

• But the matrix picture is conceptually useful.
Finding Similar Columns

So far:

- Documents → Sets of shingles
- Represent sets as boolean vectors in a matrix

Next goal: Find similar columns while computing small signatures

- Similarity of columns == similarity of signatures
Finding Similar Columns

- **Next Goal:** Find similar columns, Small signatures

- **Naïve approach:**
  1. **Signatures of columns:** small summaries of columns
  2. **Examine pairs of signatures** to find similar columns
     - **Essential:** Similarities of signatures and columns are related
  3. **Optional:** Check that columns with similar signatures are really similar

- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)
Hashing Columns: Signatures

- **Key idea:** “hash” each column \( C \) to a small *signature* \( h(C) \), such that:
  1. \( h(C) \) 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 \( h(C_1) \) and \( h(C_2) \).

- **Goal:** find a hash function such that
  - if \( \text{sim}(C_1, C_2) \) is high, then with high prob \( h(C_1) = h(C_2) \)
  - if \( \text{sim}(C_1, C_2) \) is low, then with high prob \( h(C_1) \neq h(C_2) \)

- Hash docs into buckets and expect that most pairs of near duplicate docs hash into the same bucket
Minhashing

• History: invented by Andrei Broder in 1997 to detect duplicate pages

• Imagine the rows of a set matrix are permuted randomly.

• Define “hash” function $h(C) = \text{the number of the first (in the permuted order) row in which column } C \text{ has 1.}$

• Use several (e.g., 100) independent hash functions to create a signature.
Minhashing Example

**Input matrix**

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Permuted matrix**

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Signature matrix $M$**

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
</tbody>
</table>
Minhashing Example

Input matrix

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Permuted matrix

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Signature matrix $M$

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

P1

P2

beadc

aebdc
Four Types of Rows

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

  $\begin{array}{ccc}
  & C_1 & C_2 \\
  a & 1 & 1 & 1 \text{ in both columns} \\
  b & 1 & 0 & \text{columns are different} \\
  c & 0 & 1 \\
  d & 0 & 0 & 0 \text{ in both columns} \\
  \end{array}$

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

• 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 \( a / (a + b + c)! \)

• Matrix is sparse – most rows are of type d

• The ratio of type a, b, and c that determine the similarity and the probability that \( h(C_1) = h(C_2) \)
Minhashing Property

- Probability that \( h(C_1) = h(C_2) \) is the same as \( Sim(C_1, C_2) = a/(a+b+c)! \)

- **Why?**
  - Look down the permuted columns \( C_1 \) and \( C_2 \) until we see a 1.
  - If its a type-\( a \) (1,1) row, then \( h(C_1) = h(C_2) \). If a type-\( b \) (1,0) or type-\( c \) (0,0) row, then not.
Similarity for Signatures

• The similarity of signatures is the fraction of the hash functions in which they agree.
Min Hashing – Example

Input matrix

<table>
<thead>
<tr>
<th></th>
<th>S₁</th>
<th>S₂</th>
<th>S₃</th>
<th>S₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</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>4</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>S₁</th>
<th>S₂</th>
<th>S₃</th>
<th>S₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Similarities:

<table>
<thead>
<tr>
<th></th>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
<th>3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col/Col</td>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sig/Sig</td>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Minhash Signatures

• Pick (say) 100 random permutations of the rows.

• Think of $\text{Sig}(C)$ as a column vector.

• Let $\text{Sig}(C)[i] =$
  according to the $i$th permutation, the number of the first row that
  has a 1 in column $C$. 
Implementation – (1)

- Suppose 1 billion rows.
- Hard to pick a random permutation from 1…billion.
- Sorting would take a long time
- Representing a random permutation requires 1 billion entries.
Implementation – (2)

- **A good approximation to permuting rows**: pick 100 (?) hash functions.

- For each column \( c \) and each hash function \( h_i \), keep a “slot” \( M(i, c) \).

- **Intent**: \( M(i, c) \) will become the smallest value of \( h_i(r) \) for which column \( c \) has 1 in row \( r \).
  - I.e., \( h_i(r) \) gives order of rows for \( i \)th permutation.
for each row $r$

for each column $c$

if $c$ has 1 in row $r$

for each hash function $h_i$ do

if $h_i(r)$ is a smaller value than $M(i, c)$ then

$M(i, c) := h_i(r)$;
**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>

Row  

<table>
<thead>
<tr>
<th>row</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(1)$</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$g(1)$</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>$h(2)$</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$g(2)$</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$h(3)$</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$g(3)$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$h(4)$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$g(4)$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$h(5)$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$g(5)$</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$h(x) = x \mod 5$

$g(x) = 2x+1 \mod 5$
Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures
Other Applications Will Follow
Finding Similar Pairs

- Suppose we have, in main memory, data representing a large number of objects.
  - May be the objects themselves.
  - May be signatures as in minhashing.

- We want to compare each to each, finding those pairs that are sufficiently similar.
Checking All Pairs is Hard

• 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 \((10^6 \cdot 2) \approx 5 \cdot 10^{11}\) column-comparisons.

• At 1 microsecond/comparison: 6 days.

\[
\frac{n!}{k! (n-k)!}
\]
Locality-Sensitive Hashing

• **General idea**: Use a function $f(x,y)$ that tells whether or not $x$ and $y$ is a *candidate pair*: a pair of elements *whose similarity must be evaluated*.

• **For minhash matrices**: Hash columns to many buckets, and make elements of the same bucket candidate pairs.
Candidate Generation From Minhash Signatures

• 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 $i$. 
LSH for Minhash Signatures

- **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

One signature
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.
  - Make $k$ as large as possible.

- *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.
Columns 2 and 6 are probably identical.

Columns 6 and 7 are surely different.

Matrix $M$

Buckets

Columns 2 and 6

Columns 6 and 7

$1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7$

$r$ rows

$b$ bands
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 in that band.”
Example: Effect of Bands

• Suppose 100,000 columns.
• Signatures of 100 integers.
• Therefore, signatures take 40Mb.
• Want all 80%-similar pairs.
• 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., about 1/3000th of the 80%-similar column pairs are false negatives.
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 only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up.
What $b$ Bands of $r$ Rows Gives You

$\text{Probability of sharing a bucket} \quad \uparrow$

$\begin{align*}
    t & \sim (1/b)^{1/r} \\
    1 - (1 - s^r)^b & \quad \text{(At least one band identical)} \\
    1 - (1 - s^r)^b & \quad \text{(No bands identical)} \\
    \text{Some row of a band unequal} \\
    \text{All rows of a band are equal}
\end{align*}$

$\text{Similarity } s \text{ of two sets}$

$t$
S-Curves
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 represent similar sets.
Applications of LSH

Entity Resolution
Similar News Articles
Desiderata

- Whatever form we use for LSH, we want:
  1. The time spent performing the LSH should be linear in the number of objects.
  2. The number of candidate pairs should be proportional to the number of truly similar pairs.
Entity Resolution

- The *entity-resolution* problem is to examine a collection of records and determine which refer to the same entity.
  - *Entities* could be people, events, etc.

- Typically, we want to merge records if their values in corresponding fields are similar.
I once took a consulting job solving the following problem:

- Company A agreed to solicit customers for Company B, for a fee.
- They then argued over how many customers.
- Neither recorded exactly which customers were involved.
Customer Records – (2)

- Company B had about 1 million records of all its customers.
- Company A had about 1 million records describing customers, some of whom it had signed up for B.
- Records had name, address, and phone, but for various reasons, they could be different for the same person.
Customer Records – (3)

- **Step 1**: Design a measure ("score") of how similar records are:
  - E.g., deduct points for small misspellings ("Jeffrey" vs. "Jeffery") or same phone with different area code.

- **Step 2**: Score all pairs of records; report high scores as matches.
Customer Records – (4)

- **Problem**: \((1 \text{ million})^2\) is too many pairs of records to score.
- **Solution**: A simple LSH.
  - Three hash functions: exact values of name, address, phone.
    - Compare iff records are identical in at least one.
    - Misses similar records with a small differences in all three fields.
Aside: Hashing Names, Etc.

• How do we hash strings such as names so there is one bucket for each string?

• **Possibility**: Sort the strings instead.
  • Used in this story.

• **Possibility**: Hash to a few million buckets, and deal with buckets that contain several different strings.
Aside: Validation of Results

• We were able to tell what values of the scoring function were reliable in an interesting way.
  • Identical records had a creation date difference of 10 days.
  • We only looked for records created within 90 days, so bogus matches had a 45-day average.
Validation – (2)

• By looking at the pool of matches with a fixed score, we could compute the average time-difference, say $x$, and deduce that fraction $(45-x)/35$ of them were valid matches.

• Alas, the lawyers didn’t think the jury would understand.
Validation – Generalized

• Any field not used in the LSH could have been used to validate, provided corresponding values were closer for true matches than false.

• Example: if records had a *height* field, we would expect true matches to be close, false matches to have the average difference for random people.
Application: Same News Article

• Recently, the Political Science Dept. asked a team from CS to help them with the problem of identifying duplicate, on-line news articles.

• Problem: the same article, say from the Associated Press, appears on the Web site of many newspapers, but looks quite different.
News Articles – (2)

• Each newspaper surrounds the text of the article with:
  • It’s own logo and text.
  • Ads.
  • Perhaps links to other articles.

• A newspaper may also “crop” the article (delete parts).
News Articles – (3)

- The team came up with its own solution, that included shingling, but not minhashing or LSH.
  - A special way of shingling that appears quite good for this application.
  - **LSH substitute**: candidates are articles of similar length.
LSH

• They implemented minhashing + LSH and found it faster for similarities below 80%.
  • Aside: That’s no surprise. When similarity is high, there are better methods, as we shall see.
New Shingling Technique

• The team observed that news articles have a lot of stop words, while ads do not.
  • “Buy Sudzo” vs. “I recommend that you buy Sudzo for your laundry.”

• They defined a shingle to be a stop word and the next two following words.

• To find an efficient solution, you need to torture the data until it talks
  • You need to be creative!
Why it Works

• By requiring each shingle to have a stop word, they biased the mapping from documents to shingles so it picked more shingles from the article than from the ads.

• Pages with the same article, but different ads, have higher Jaccard similarity than those with the same ads, different articles.
Another Application
Collaborative Filtering

- Recommend to users items that were liked by other users who have exhibited similar tastes
- Online purchases
- Movie ratings
Online Purchases

• Two customers are similar if their sets of purchased items have a high Jaccard similarity
  John: laptop, hard-disk, memory, ipad, guitar
  Mary: desktop, hard disk, ipad, piano

• Two items that have sets of purchasers with high Jaccard similarity will be deemed similar

• High is relative!
  • Likely, a meaningful similarity threshold here is lower than for similar documents

• Clustering can help:
  John: computer, computer peripheral, memory, ipad, musical instrument
  Mary: computer, computer peripheral, musical instrument, ipad
Movie Ratings

• Netflix records
  • Movies seen by customers
  • Ratings assigned to movies

• Movie1 is similar to Movie2 is they were rented or rated highly by many of the same customers

• Customer1 is similar to Customer2 if they rented or rated highly many of the same movies

• Ratings need to be treated differently than binary decisions
  • Ignore low-rated customer/movie pairs
  • Map ratings to categories and look for similarity in these sets
    • high score->liked, low score->hated,
Distance Measures
Distance Measures

- Generalized LSH is based on some kind of “distance” between points.
  - Similar points are “close.”

- Two major classes of distance measure:
  1. *Euclidean*
  2. *Non-Euclidean*
A *Euclidean space* has some number of real-valued dimensions and “dense” points.

- There is a notion of “average” of two points.
- A *Euclidean distance* is based on the locations of points in such a space.

A *Non-Euclidean distance* is based on properties of points, but not their “location” in a space.
Axioms of a Distance Measure

• $d$ is a distance measure if it is a function from pairs of points to real numbers such that:
  1. $d(x,y) > 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality).
Some Euclidean Distances

- $L_2$ norm: $d(x,y) = \sqrt{\text{sum of the squares of the differences between } x \text{ and } y \text{ in each dimension}}$.
  - The most common notion of “distance.”

- $L_1$ norm: sum of the differences in each dimension.
  - *Manhattan distance* = distance if you had to travel along coordinates only.
Examples of Euclidean Distances

$L_2$-norm:
\[ \text{dist}(a,b) = \sqrt{4^2 + 3^2} \]
\[ = 5 \]

$L_1$-norm:
\[ \text{dist}(a,b) = 4 + 3 = 7 \]
Another Euclidean Distance

- \( L_\infty \) norm: \( d(x,y) = \) the maximum of the differences between \( x \) and \( y \) in any dimension.

- **Note**: the maximum is the limit as \( n \) goes to \( \infty \) of the \( L_n \) norm: what you get by taking the \( n \)th power of the differences, summing and taking the \( n \)th root.
Examples of Euclidean Distances

a = (5,5)  

b = (9,8)  

$L_\infty$-norm:  
Max(|9-5|,|8-5|) = Max(4,3) = 4

What is the $L_\infty$-norm for a and b?
Non-Euclidean Distances

- **Jaccard distance** for sets = 1 minus Jaccard similarity.
- **Cosine distance** = angle between vectors from the origin to the points in question.
- **Edit distance** = number of inserts and deletes to change one string into another.
- **Hamming Distance** = number of positions in which bit vectors differ.
Jaccard Distance for Sets

• **Example:** $p_1 = 10111; p_2 = 10011$.

• Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = 3/4.

• $d(x,y) = 1 - \text{(Jaccard similarity)} = 1/4$. 
Why J.D. Is a Distance Measure

• $d(x,x) = 0$ because $x \cap x = x \cup x$.
• $d(x,y) = d(y,x)$ because union and intersection are symmetric.
• $d(x,y) \geq 0$ because $|x \cap y| \leq |x \cup y|$.
• $d(x,y) \leq d(x,z) + d(z,y)$ trickier (see textbook)
Triangle Inequality for J.D.

\[
1 - |x \cap z| + 1 - |y \cap z| \geq 1 - |x \cap y| \\
\frac{|x \cup z|}{|x \cup y|} = \frac{|y \cup z|}{|x \cup y|}
\]

- **Remember**: \(|a \cap b|/|a \cup b| = \text{probability that } \text{minhash}(a) = \text{minhash}(b)\).

- **Thus**, \(1 - |a \cap b|/|a \cup b| = \text{probability that } \text{minhash}(a) \neq \text{minhash}(b)\).
Triangle Inequality – (2)

- **Claim**: $\text{prob}[\text{minhash}(x) \neq \text{minhash}(y)] \leq \text{prob}[\text{minhash}(x) \neq \text{minhash}(z)] + \text{prob}[\text{minhash}(z) \neq \text{minhash}(y)]$

- **Proof**: whenever $\text{minhash}(x) \neq \text{minhash}(y)$, at least one of $\text{minhash}(x) \neq \text{minhash}(z)$ and $\text{minhash}(z) \neq \text{minhash}(y)$ must be true.
Cosine Distance

- Think of a point as a vector from the origin \((0,0,\ldots,0)\) to its location.

- Two points’ vectors make an angle, whose cosine is the normalized dot-product of the vectors: \(p_1.p_2/\|p_2\|\|p_1\|\).
  - Example: \(p_1 = 00111; p_2 = 10011\).
  - \(p_1.p_2 = 2; \|p_1\| = \|p_2\| = \sqrt{3}\).
  - \(\cos(\theta) = 2/3; \theta\) is about 48 degrees.
Cosine-Measure Diagram

$$d(p_1, p_2) = \theta = \arccos\left(\frac{p_1 \cdot p_2}{|p_2||p_1|}\right)$$
Why C.D. Is a Distance Measure

- $d(x,x) = 0$ because $\arccos(1) = 0$.
- $d(x,y) = d(y,x)$ by symmetry.
- $d(x,y) \geq 0$ because angles are chosen to be in the range 0 to 180 degrees.
- **Triangle inequality**: physical reasoning. If I rotate an angle from $x$ to $z$ and then from $z$ to $y$, I can’t rotate less than from $x$ to $y$. 

Edit Distance

• The *edit distance* of two strings is the number of inserts and deletes of characters needed to turn one into the other. Equivalently:

  • \( d(x,y) = |x| + |y| - 2|\text{LCS}(x,y)|. \)

    • \( \text{LCS} = \textit{longest common subsequence} = \) any longest string obtained both by deleting from \( x \) and deleting from \( y \).
Example: LCS

- $x = abcde$ ; $y = bcduve$.
- Turn $x$ into $y$ by deleting $a$, then inserting $u$ and $v$ after $d$.
  - Edit distance = 3.
- Or, LCS($x,y$) = $bcde$.
- Note: $|x| + |y| - 2|LCS(x,y)| = 5 + 6 - 2*4 = 3 = \text{edit distance.}$
Why Edit Distance Is a Distance Measure

- $d(x,x) = 0$ because 0 edits suffice.
- $d(x,y) = d(y,x)$ because insert/delete are inverses of each other.
- $d(x,y) \geq 0$: no notion of negative edits.
- **Triangle inequality**: changing $x$ to $z$ and then to $y$ is one way to change $x$ to $y$. 
Variant Edit Distances

• Allow insert, delete, and *mutate*.
  • Change one character into another.

• Minimum number of inserts, deletes, and mutates also forms a distance measure.

• Ditto for any set of operations on strings.
  • Example: substring reversal OK for DNA sequences
Hamming Distance

• **Hamming distance** is the number of positions in which bit-vectors differ.

• **Example:** $p_1 = 10101$; $p_2 = 10011$.

• $d(p_1, p_2) = 2$ because the bit-vectors differ in the 3$^{rd}$ and 4$^{th}$ positions.
Why Hamming Distance Is a Distance Measure

• $d(x, x) = 0$ since no positions differ.

• $d(x, y) = d(y, x)$ by symmetry of “different from.”

• $d(x, y) \geq 0$ since strings cannot differ in a negative number of positions.

• **Triangle inequality:** changing $x$ to $z$ and then to $y$ is one way to change $x$ to $y$. 