Finding Similar Sets

Vassilis Christophides
christop@csd.uoc.gr
http://www.csd.uoc.gr/~hy562
University of Crete, Fall 2019

Motivation

- Many Web-mining problems can be expressed as finding “similar” sets:
  - Pages with similar words, e.g., for classification by topic
  - NetFlix users with similar tastes in movies for recommendation systems
    - Dual: movies with similar sets of fans
  - Images of related things

- The best techniques depend on whether you are looking for items that are very similar or only somewhat similar
  - 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 document appear out of order in another, is very hard
Comparing Documents for Near Duplicates

- **Applications**: Given a body of documents, find pairs of documents with a lot of text in common, e.g.:
  - Mirror Web 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”

- Simple IR approaches are not suited:
  - Document = set of words appearing in document
  - Document = set of “important” words
  - Why? we need to account for ordering of words!

Main Issues

- What is the right representation of the document when we check for similarity?
  - E.g., representing a document as a set of characters will not do (why?)

- When we have billions of documents, keeping the full text in memory is not an option
  - We need to find a shorter representation

- How do we do pairwise comparisons of billions of documents?
  - If exact match was the issue it would be ok, can we replicate this idea?
Three Essential Techniques for Detecting Similar Documents

- **Shingling**: convert documents, emails, etc., to *sets*
- **Minhashing**: convert *large sets to short signatures*, while preserving similarity
- **Locality-sensitive hashing**: focus on *pairs of signatures likely to be similar*

**Shingles**

- A $k$-shingle (or k-gram) for a document is a sequence of $k$ characters that appears in the document
  - Represent a document by its set of $k$-shingles

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

- **Working Assumption**: Documents that have lots of shingles in common have similar text, even if the text appears in different order
  - What if two documents differ by a word?
    - Affects only $k$-shingles within distance $k$ from the word
  - What if we reorder paragraphs?
    - Affects only the $2k$ shingles that cross paragraph boundaries
Shingle Size

Is $k=2$ a good choice for size?

- Example: $k=2$;
  - $\text{doc1} = \text{abcab}$. 2-shingles = \{ab, bc, ca\}
  - $\text{doc2} = \text{cabc}$. 2-shingles = \{ab, bc, ca\}

- Careful decision: you must pick $k$ large enough, or most documents will have most shingles
  - $k = 5$ is OK for short documents
  - $k = 10$ is better for long documents

Shingles: Compression Option

- How about space overhead?
  - Each character can be represented as a byte
  - $k$-shingle requires $k$ bytes

- 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, and
  - represent a document by the set of hash values of its $k$-shingles
  - $(\text{aaabbbccc})(\text{abcabcabc}) \rightarrow h(\text{aaabbbccc})h(\text{abcabcabc})$
    - 18 bytes $\rightarrow$ 8 bytes

- Working Assumption: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared
Thought Question

- Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?
  - There are many more possible shingles, this reduces the likelihood that documents that share many shingles are not similar

- Hint: How random are the 32-bit sequences that result from 4-shingling?
  - Assuming 20 characters are common in English, there are \((20)^4 = 160000\) 4-shingles \(< 2^{32}\)
  - Using 9-shingles there are \((20)^9 >> 2^{32}\)

MinHashing
Basic Data Model: Sets

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

- Examples include:
  - Documents represented by their sets of shingles (or hashes of those shingles): \( C_i = S(D_i) \)
  - Similar customers or products

- Equivalently, each document is a 0/1 vector in the space of k-shingles:
  - Each unique shingle is a dimension
  - Vectors are very sparse

- Interpret set intersection as bitwise AND, and set union as bitwise OR

Jaccard Similarity of Sets

- The Jaccard similarity of two sets is the size of their intersection divided by the size of their union:
  \[
  Sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}
  \]

- Example:

  ![Venn Diagram](image)

  - 3 in intersection
  - 8 in union
  - Jaccard similarity = 3/8
Motivation for Min-Hash

- Suppose we need to find near-duplicate documents among $N = 1$ million ($10^6$) documents

- Naïvely, we would have to compute pairwise Jaccard similarities for every pair of docs
  - $N(N-1)/2 \approx 5 \times 10^{11}$ comparisons
  - At $= 10^5$ secs/day and $10^8$ comparisons/sec, it would take 5 days

- For $N = 10$ million ($10^7$), it takes more than a year...

From Sets to Boolean Matrices

- **Rows** = elements (shingles) of the universal set
- **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 sets of their rows with 1

- Typical matrix is sparse (most rows are of type d, see later)
  - Sparse matrices are usually better represented by the list of places where there is a non-zero value
  - But the boolean matrix picture is conceptually useful
Example: Jaccard Similarity of Columns

<table>
<thead>
<tr>
<th></th>
<th>C₁</th>
<th>C₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Sim \((C₁, C₂) = \frac{2}{5} = 0.4\)

d\((C₁, C₂) = 1 - \text{(Jaccard similarity)} = 0.6\)

Outline: Finding Similar Columns

- Naïve approach:
  1. Compute signatures of columns = small summaries of columns
  2. Examine pairs of signatures to find similar columns
     - Requirement: similarities of signatures and columns are related
  3. Optional: check that columns with similar signatures are really similar
- This scheme works but …
  - What if the set of signatures (or k-shingles) is too large to fit in the memory?
  - Or the number of documents are too large?
- Idea: Find a way to hash a document (column) to a single (small size) value! and similar documents to the same value!
  - Warning: These methods can produce false negatives, and even false positives (if the above optional check is not made)
Signatures

- Key idea: “hash” $h(\cdot)$ each column $C$ to a small signature, such that:
  1. $h(C)$ is small enough that we can fit a signature in main memory for each column
  2. $Sim(C_1, C_2)$ is the same as the “similarity” of $h(C_1)$ and $h(C_2)$
- By hashing columns into buckets we expect that “most” pairs of near duplicate documents hash into the same bucket!
- **Goal**: Find a hash function $h(\cdot)$ such that:
  - If $sim(C_1, C_2)$ is high, then with high probability $h(C_1) = h(C_2)$
  - If $sim(C_1, C_2)$ is low, then with high probability $h(C_1) \neq h(C_2)$
- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function!
  - There is a suitable hash function for the Jaccard similarity:
    - It is called Min-Hashing!

Minhashing

- History: invented by Andrei Broder in 1997 (AltaVista) to detect near duplicate web pages
- Imagine the rows of the Boolean matrix permuted under random permutation $\pi$
- Define a “hash” function $h_{\pi}(C) = \text{index of the first} (\text{in the permuted order } \pi) \text{ row in which column } C \text{ has value 1}$:
  - $h_{\pi}(C) = \min_{\pi} \pi(C)$
Min-hashing Example

2nd element of the permutation is the first to map to a 1 in col 1

Input matrix

<table>
<thead>
<tr>
<th>Permutations</th>
<th>Input matrix</th>
<th>Signature matrix M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 3</td>
<td>1 0 1 0</td>
<td>2 1 2 1</td>
</tr>
<tr>
<td>3 2 4</td>
<td>1 0 0 1</td>
<td>2 1 4 1</td>
</tr>
<tr>
<td>7 1 7</td>
<td>0 1 0 1</td>
<td>1 2 1 2</td>
</tr>
<tr>
<td>6 3 6</td>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>2 6 1</td>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>5 7 2</td>
<td>1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>4 5 5</td>
<td>1 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

$h_2(3) = 4$ (permutation 2, column 3)

4th element of the permutation is the first to map to a 1 in col 3

Surprising Property

- The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $\text{Sim}(C_1, C_2)$:
  - $\Pr[h_n(C_1) = h_n(C_2)] = \text{sim}(C_1, C_2)$

- With multiple signatures we get a good approximation

- Use several independent hash functions to create a signature of a column
  - The similarity of signatures is the fraction of the hash functions in which they agree
  - Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures
Why?

Let \( X \) be a column (set of shingles), \( y \in X \) is a shingle

Then: \( \Pr[\pi(y) = \min(\pi(X))] = 1/|X| \)

\( \Rightarrow \) It is equally likely that any shingle \( y \in X \) is mapped to the \textit{min} element

Let \( y \) be s.t. \( \pi(y) = \min(\pi(C_1 \cup C_2)) \)

Then either: \( \pi(y) = \min(\pi(C_1)) \) if \( y \in C_1 \), or \( \pi(y) = \min(\pi(C_2)) \) if \( y \in C_2 \)

So the prob. that \textit{both} are true is the prob. \( y \in C_1 \cap C_2 \)

\[ \Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2|/|C_1 \cup C_2| = \text{Sim}(C_1, C_2) \]

Four Types of Rows

Given columns \( C_1 \) and \( C_2 \), rows may be classified as:

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

Also, \( a = \# \) rows of type \( a \), etc.

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

\( \Rightarrow \) Note \( \text{Sim}(C_1, C_2) = a / (a + b + c) \)

Then: \( \Pr[h(C_1)=h(C_2)] = \text{Sim}(C_1, C_2) \)

Look down the permuted columns \( C_1 \) and \( C_2 \) until we see a 1

\( \Rightarrow \) If it’s a type-\( a \) row, then \( h(C_1)=h(C_2) \)

\( \Rightarrow \) If a type-\( b \) or type-\( c \) row, then not
Min Hashing – Example

Input matrix

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

Signature matrix $M$

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</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>

Similarities:

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

MinHash – False Positive/Negative

- Instead of comparing sets, we now compare only 1 bit!

- False positive?
  - False positive can be easily dealt with by doing an additional layer of checking (treat minhash as a filtering mechanism)

- False negative?
  - Requiring full match of signature is strict, some similar sets will be lost

- High error rate! Can we do better?
Minhash Signatures

- Pick (say) 100 random permutations of the rows
- Think of $\text{Sig}(C)$ as a column vector
- Let $\text{Sig}(C)[i] = \min(\pi_i(C))$
  according to the $i$ th permutation, the number of the first row that has a 1 in column $C$
- **Note:** The sketch (signature) of column $C$ is small ~400 bytes!
  - We achieved our goal! We “compressed” long bit vectors into short signatures

---

Implementation Trick

- **Permuting rows even once is prohibitive**
  - 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
- **A good approximation to permuting rows:** pick 100 (?) hash functions $h_i$
  - Simulate the effect of a random permutation by a random hash function that maps row numbers to as many buckets as there are rows
  - **Row hashing:** ordering under $h_i$ gives a random row permutation!
- **One-pass implementation**
  - For each column $C$ and each hash function $h_i$, keep a “slot” $M(i,C)$ for the min-hash value
  - **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
Implementation

\[ M(i, C) = \infty \]

for each row \( r \)
for each column \( C \)
if \( C \) has 1 in row \( r \) // Scan rows looking for 1s
for each hash function \( h_i \) do
// Suppose row \( r \) has 1 in column \( C \)
if \( h_i(r) \) is a smaller value than \( M(i, C) \) then
\[ M(i, C) := h_i(r); \]

How to pick a random hash function \( h(x) \)?
Universal hashing:
\[ h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N \]
where:
\( a, b \) ... random integers
\( p \) ... prime number (\( p > N \))

Example

<table>
<thead>
<tr>
<th>Row</th>
<th>Sig1</th>
<th>Sig2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>C2</td>
<td></td>
</tr>
<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>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ h(x) = x \mod 5 \]
\[ g(x) = 2x + 1 \mod 5 \]

Jaccard = 1/5
So far …

- Represent a document as a set of hash values (of its k-shingles)
- Transform set of k-shingles to a set of minhash signatures
- Use Jaccard to compare two documents by comparing their signatures
- Is this method (i.e., transforming sets to signature) necessarily “better”??

Locality-Sensitive Hashing

<table>
<thead>
<tr>
<th>general hashing</th>
<th>locality-sensitive hashing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Suppose, in **main memory**, a **representation** of a large number of objects
- May be signatures of documents as in minhashing
- We want to **pair-wise compare each** for finding those pairs that are sufficiently similar

### Finding Similar Pairs

- 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

- **Naïve solution**
  - For each document, compare with the other **N-1** documents
    - Takes **N-1** comparisons
    - Can be optimized using *filter-and-refine* mechanisms
  - Requires **N*(N-1)/2** comparisons

- **Example**:
  - **10^7** columns implies ~ **10^{14}** column-comparisons
  - At 1 μs/comparison **10^8** (~ 3 years!)
Locality-Sensitive Hashing

- 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.

- With only one hash function on one entire column of signature, likely to have many false negatives.

- **Key idea**: Apply the hash function on the columns of signature matrix $M$ multiple times, each on a partition of the column.
  - Arrange that (only) similar columns are likely to hash (i.e., with high probability) to the same bucket.
  - Each pair of columns that hashes at least once into the same bucket is a candidate pair.

---

Candidate Generation from Minhash Signatures

- Pick a similarity threshold $s$, a fraction $0 < s < 1$.
- A pair of columns $x$ and $y$ is a candidate pair if their signatures agree in at least fraction $s$ of the rows.
  - i.e., $M(i, x) = M(i, y)$ for at least fraction $s$ values of $i$.
  - We expect documents $x$ and $y$ to have the same (Jaccard) similarity as their signatures.
Partition Into Bands

- Divide matrix $M$ into $b$ bands of $r$ rows
  - For each document, compute $b$ sets of $r$ minhash values
  - Each set is a mini-signature with $r$ minhash functions (or a concatenation of the $r$ minhash values together)

$\sum b$ mini-signatures

Matrix $M$

Partition into Bands

- For each band, hash its portion of each column (i.e., the concatenated values) to a hash table with $k$ buckets
  - this has the "same" effect as ensuring all columns have the same values
  - make $k$ as large as possible to minimize collision
- 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 non-similar pairs

Columns 2 and 6 are probably identical
Columns 6 and 7 are surely different
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”
  - Assumption needed only to simplify analysis, not for correctness of algorithm
- **Finding all pairs within a bucket become computationally cheaper!**
  - Declare all pairs within a bucket to be “matching” OR
  - Perform pair-wise comparisons for those documents that fall into the same bucket
    - Much smaller than pair-wise over all documents

Example: Effect of Bands

- Suppose $10^5$ columns of M (100k docs)
- Signatures of $10^2$ integers (rows)
- If each signature is represented as a 4 byte integer value, we need only $10^2 \times 4 \times 10^5 = 40$ Mb of memory!
- $5 \times 10^9$ pairs of signatures can take a while to compare
- Choose 20 bands of 5 integers/band
- Goal: Find pairs of documents that are *at least* $s = 0.8$ similar
Suppose $C_1, C_2$ are 80% Similar

- Find pairs of $\geq s=0.8$ similarity, set $b=20$, $r=5$
- Assume: $\text{sim}(C_1, C_2) = 0.8$
  - Since $\text{sim}(C_1, C_2) \geq s$, we want $C_1, C_2$ to be a candidate pair
  - We want them to hash to at least 1 common bucket (at least one band is identical)

- Probability $C_1, C_2$ identical in one particular band: $(0.8)^5 = 0.328$
- Probability $C_1, C_2$ are not similar in all of the 20 bands: $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)

- We would find 99.965% pairs of truly similar documents

Suppose $C_1, C_2$ are 30% Similar

- Find pairs of $\geq s=0.8$ similarity, set $b=20$, $r=5$
- Assume: $\text{sim}(C_1, C_2) = 0.3$
  - Since $\text{sim}(C_1, C_2) < s$, we want $C_1, C_2$ to hash to no common buckets (all bands should be different)

- Probability $C_1, C_2$ identical in one particular band: $(0.3)^5 = 0.00243$
  - Probability $C_1, C_2$ identical in at least 1 of 20 bands: $1-(1-0.00243)^{20} = 0.0474$
  - In other words, approximately 4.74% pairs of docs with similarity 0.3 end up becoming candidate pairs
  - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold $s$
LSH Involves a Tradeoff

- How to get a step-function?
- Pick:
  - The number of Min-Hashes (rows of $M$
  - The number of bands $b$, and
  - The number of rows $r$ per band to balance false positives/negatives

*Example:* if we had only 20 bands of 5 rows, the number of false negatives would go down, but the number of false positives would go up.

Analysis of LSH – What We Want

- Probability of sharing a bucket
- Similarity threshold $s$
- Probability $= 1$ if $t > s$
- No chance if $t < s$

Similarity $t = \text{sim}(C_1, C_2)$ of two sets

---

What One Band Gives You

- Probability of sharing a bucket
- Similarity threshold $s$
- Single hash signature
- Remember: probability of equal hash-values = similarity

Similarity $t = \text{sim}(C_1, C_2)$ of two sets

- This is what 1 hash-code gives you
  $\Pr[h_r(C_1) = h_r(C_2)] = \text{sim}(C_1, C_2)$
What \( b \) Bands of \( r \) Rows Gives You

- The S-curve is where the “magic” happens

The S-curve is where the “magic” happens

\[ s \sim (1/b)^{1/r} \]

At least one band identical

No bands identical

Some row of a band unequal

All rows of a band are equal

Example: \( b = 20; r = 5 \)

<table>
<thead>
<tr>
<th>( t )</th>
<th>( 1-(1-t^r)^b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>.2</td>
<td>.006</td>
</tr>
<tr>
<td>.3</td>
<td>.047</td>
</tr>
<tr>
<td>.4</td>
<td>.186</td>
</tr>
<tr>
<td>.5</td>
<td>.470</td>
</tr>
<tr>
<td>.6</td>
<td>.802</td>
</tr>
<tr>
<td>.7</td>
<td>.975</td>
</tr>
<tr>
<td>.8</td>
<td>.9996</td>
</tr>
</tbody>
</table>

\[ s = 0.5 \ (\sim 1/20)^{1/5} \]

Probability of sharing a bucket

Similarity \( t = \text{sim}(C_1, C_2) \) of two sets

Figure 3.7: The S-curve
S-curves as a Function of $b$ and $r$

- Given a fixed threshold $s$
- We want choose $r$ and $b$ such that the $Pr(\text{Candidate pair})$ has a "step" right around $s$

Picking $r$ and $b$: The S-Curve

- Picking $r$ and $b$ to get the best S-curve

Blue area: False Negative rate
These are pairs with $\text{sim} > s$ but the X fraction won't share a band and then will never become candidates. This means we will never consider these pairs for (slow/exact) similarity calculation!

Green area: False Positive rate
These are pairs with $\text{sim} < s$ but we will consider them as candidates. This is not too bad, we will consider them for (slow/exact) similarity computation and discard them.
Picking $r$ and $b$ to Get Desired Performance

- 50 hash-functions ($r \cdot b = 50$)

Limitations of Minhash

- Minhash is great for near-duplicate detection
  - Set high threshold for Jaccard similarity

- Limitations:
  - Jaccard similarity only
  - Set-based representation, no way to assign weights to features

- Random projections:
  - Works with arbitrary vectors using cosine similarity
  - Same basic idea, but details differ
  - Slower but more accurate: no free lunch!
LSH Generalizations

Multiple Hash Functions

- For Min-Hashing signatures, we got a Min-Hash function for each permutation of rows

- So far, we have assumed only one hash function (even applied multiple times)
  - Shorthand: $h(x) = h(y)$ implies “$h$ says $x$ and $y$ are equal”

- We could have used a family of hash functions
  - A (large) set of related hash functions generated by some mechanism
  - We should be able to efficiently pick a hash function at random from such a family
Locality-Sensitive (LS) Families

- Consider a space $S$ of points with a distance measure $d$
- A family $H$ of hash functions is said to be $(d_1, d_2, p_1, p_2)$-sensitive if for any $x$ and $y$ in $S$:
  - If $d(x, y) \leq d_1$, then prob over all $h$ in $H$ that $h(x)=h(y)$ is at least $p_1$
  - If $d(x, y) \geq d_2$, then prob over all $h$ in $H$ that $h(x)=h(y)$ is at most $p_2$

Small distance, high probability of hashing to the same value

Large distance, low probability of hashing to the same value

Example of LS Family: MinHash

- Let
  - $S =$ space of all sets,
  - $d =$ Jaccard distance,
  - $H$ is family of Min-Hash functions for all permutations of rows

- Minhashing gives a $(d_1, d_2, p_1, p_2)$-sensitive family for any $d_1 < d_2$
  - E.g., $H$ is a $(1/3, 2/3, 2/3, 1/3)$-sensitive family for $S$ and $d$
  - If distance $\leq 1/3$ (i.e., similarity $\geq 2/3$), then probability that minhash values agree is $\geq 2/3$
  - This is because for any hash function $h \in H$
    $\Pr(h(x)=h(y))=1-d(x, y)$

- Simply restates theorem about Min-Hashing in terms of distances rather than similarities!
Example of LS Family: MinHash

- **Claim**: Min-hash $H$ is a $(1/3, 2/3, 2/3, 1/3)$-sensitive family for $S$ and $d$

  If distance < 1/3 (so similarity ≥ 2/3)  
  Then probability that Min-Hash values agree ≥ 2/3

- For Jaccard similarity, Min-Hashing gives a $(d_1, d_2, (1-d_1), (1-d_2))$-sensitive family for any $d_1 < d_2$

- Theory leaves unknown what happens to pairs that are at distance between $d_1$ and $d_2$
  - **Consequence**: No guarantees about fraction of false positives in that range

Amplifying a LS-family

- Can we reproduce the “S-curve” effect we saw before for any LS family?

- The “bands” technique we learned for signature matrices carries over to this more general setting
  - So we can do LSH with any $(d_1, d_2, p_1, p_2)$-sensitive family

- Two constructions:
  - **AND** construction like “rows in a band”
  - **OR** construction like “many bands”
**AND Construction of Hash Functions**

- Given family $H$, construct family $H'$ consisting of $r$ functions from $H$

- For $h=[h_1, ..., h_r]$ in $H'$, $h(x)=h(y)$ if and only if $h_i(x)=h_i(y)$ for all $i$, $1 \leq i \leq r$

- Note this has the same effect as "r signatures":
  - $x$ and $y$ are considered a candidate pair if every one of the $r$ rows say that $x$ and $y$ are equal

**Theorem:** If $H$ is $(d_1, d_2, p_1, p_2)$-sensitive, then $H'$ is $(d_1, d_2, p_1^r, p_2^r)$-sensitive

- That is, for any $p$, if $p$ is the probability that a member of $H$ will declare $(x, y)$ to be a candidate pair, then the probability that a member of $H'$ will so declare is $p^r$

- **Proof:** Use the fact that $h_i$'s are independent

---

**OR Construction of Hash Functions**

- Given family $H$, construct family $H'$ consisting of $b$ functions from $H$

- For $h=[h_1, ..., h_b]$ in $H'$, $h(x)=h(y)$ if and only if $h_i(x)=h_i(y)$ for at least one $i$, $1 \leq i \leq b$

- Mirrors the effect of combining "b bands":
  - $x$ and $y$ become a candidate pair if any set makes them a candidate pair

**Theorem:** If $H$ is $(d_1, d_2, p_1, p_2)$-sensitive, then $H'$ is $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive

- That is, for any $p$, if $p$ is the probability that a member of $H$ will declare $(x, y)$ to be a candidate pair, then $(1-p)$ is the probability that it will not declare so

- $(1-p)^b$ is the probability that none of the family $h_1, h_b$ will declare $(x, y)$ a candidate pair

- $1-(1-p)^b$ is the probability that at least one $h_i$ will declare $(x, y)$ a candidate pair, and therefore that $H'$ will declare $(x, y)$ to be a candidate pair
Effect of AND & OR Constructions

- **AND** makes all probabilities shrink, but by choosing $r$ correctly, we can make the lower probability approach 0 while the higher does not.
- **OR** makes all probabilities grow, but by choosing $b$ correctly, we can make the upper probability approach 1 while the lower does not.

![Graphs showing the effect of AND and OR on probabilities](image)

Composing Constructions: AND-OR Composition

- $r$-way **AND** construction followed by $b$-way **OR** construction
  - Exactly what we did with minhashing
    - If $b$ bands match in all $r$ values hash to same bucket
    - Columns that are hashed into $\geq 1$ common bucket $\rightarrow$ candidate
  - Take points $x$ and $y$ s.t. $\Pr[h(x) = h(y)] = p$
    - $H$ will make $(x, y)$ a candidate pair with probability $p$
  - Construction makes $(x, y)$ a candidate pair with probability $1 - (1 - p^r)^b$
    - The S-Curve!
Example

- **Example:** Take $H$ and construct $H'$ by the AND construction with $r = 4$. Then, from $H'$, construct $H''$ by the OR construction with $b = 4$.

- E.g., transform a $(0.2, 0.8, 0.8, 0.2)$-sensitive family into a $(0.2, 0.8, 0.8785, 0.0064)$-sensitive family.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$1-(1-p^4)^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.2</td>
<td>.0064</td>
</tr>
<tr>
<td>.3</td>
<td>.0320</td>
</tr>
<tr>
<td>.4</td>
<td>.0985</td>
</tr>
<tr>
<td>.5</td>
<td>.2275</td>
</tr>
<tr>
<td>.6</td>
<td>.4260</td>
</tr>
<tr>
<td>.7</td>
<td>.6666</td>
</tr>
<tr>
<td>.8</td>
<td>.8785</td>
</tr>
<tr>
<td>.9</td>
<td>.9860</td>
</tr>
</tbody>
</table>

Composing Constructions: OR-AND Composition

- b-way OR construction followed by r-way AND construction
- Transforms probability $p$ into $(1-(1-p)^b)^r$
  - The same S-curve, mirrored horizontally and vertically
**Example**

- **Example**: Take $H$ and construct $H'$ by the **OR** construction with $b = 4$. Then, from $H'$, construct $H''$ by the **AND** construction with $r = 4$.

- E.g., transform a $(0.2, 0.8, 0.8, 0.2)$-sensitive family into a $(0.2, 0.8, 0.9936, 0.1215)$-sensitive family.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$(1-p)^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>.0140</td>
</tr>
<tr>
<td>.2</td>
<td>.1215</td>
</tr>
<tr>
<td>.3</td>
<td>.3334</td>
</tr>
<tr>
<td>.4</td>
<td>.5740</td>
</tr>
<tr>
<td>.5</td>
<td>.7725</td>
</tr>
<tr>
<td>.6</td>
<td>.9015</td>
</tr>
<tr>
<td>.7</td>
<td>.9680</td>
</tr>
<tr>
<td>.8</td>
<td>.9936</td>
</tr>
</tbody>
</table>

**Cascading Constructions**

- **Example**: Apply the $(4,4)$ **OR-AND** construction followed by the $(4,4)$ **AND-OR** construction.

- Transforms a $(.2,.8,.8,.2)$-sensitive family into a $(.2,.8,.9999996,.0008715)$-sensitive family.
  - Note this family uses $256 (= 4^4)$ of the original hash functions.
Applications of LSH

Application 2: A LHS Family for Fingerprint Matching

- Fingerprint can be uniquely defined by its minutiae
- By overlaying a grid on the fingerprint image, we can extract the grid squares where the minutiae are located
- Two fingerprints are similar if the set of grid squares significantly overlap
  - Jaccard distance and minhash can be used, but …
- Let $F$ be a family of functions
  - $f \in F$ is defined by, say 3, grid squares such that $f$ returns the same bucket whenever the fingerprint has minutiae in all three grid squares
  - $f$ sends all fingerprints that have minutiae in all three of $f$’s grid points to the same bucket
  - Two fingerprints match if they are in the same bucket
LSH for Fingerprint Matching

- Suppose probability of finding a minutiae in a random grid square of a random finger is 0.2
- And probability of finding one in the same grid square of the same finger (different fingerprint) is 0.8
- Prob two fingerprints from different fingers match = \( (0.2)^2 \times (0.2)^2 \times (0.2)^2 \times (0.2)^2 = 0.000064 \)
- Prob two fingerprints from the same finger match = \( (0.2)^2 \times (0.8)^2 \times (0.2)^2 \times (0.8)^2 = 0.004096 \)
- Use more functions from F!
- Take 1024 functions and do an OR construction
  - Prob putting the fingerprints from the same finger in at least one bucket is \( 1 - (1 - 0.004096)^{1024} = 0.985 \)
  - Prob two fingerprints from different fingers falling into the same bucket is \( 1 - (1 - 0.000064)^{1024} = 0.063 \)
  - We have 1.5% false negatives and 6.3% false positives
- Using AND construction will
  - Greatly reduce the prob of a false positive
  - Small increase in false-negative rate

References

- CS9223 – Massive Data Analysis J. Freire & J. Simeon New York University Course 2013
- CS246: Mining Massive Datasets Jure Leskovec, Stanford University, 2014
- CS5344: Big Data Analytics Technology, TAN Kian-Lee, National University of Singapore 2014