Information near-duplicates
Minimum hashing; Locality Sensitive Hashing

Web Search
Information near-duplicates

• Corpus duplicates
  • Usually, a corpus has many different topics discussed across different documents.
  • Organizing a corpus into groups of documents, unveils the diversity of topics covered by the corpus.

• Search results duplicates
  • Many search results talk about the same information facts.
  • Grouping search results by their content, enables the computation of equally relevant documents, but more informative results.
For better navigation of search results

- For grouping search results thematically
  - clusty.com / Vivisimo
Finding near-duplicates

- Typically our search space contains millions or billions of vectors.
- Data is very high dimensional. $D > 30,000$
- Finding near-duplicates has a quadratic cost on the number of documents.

- Cost:
  - $N \cdot D$ for nearest neighbor
  - $\frac{(N \cdot D)^2}{2}$ for finding near-duplicates pairs
Similarity based hash functions
Duplicate detection, min-hash, sim-hash
Web Search
Duplicate documents

• The web is full of duplicated content

• Strict duplicate detection = exact match
  • Not as common

• But many, many cases of near-duplicates
  • E.g., Last modified date the only difference between two copies of a page
Duplicate/near-duplicate detection

• Duplication: Exact match can be detected with fingerprints

• Near-Duplication: Approximate match
  • Compute syntactic similarity with an edit-distance measure
  • Use similarity threshold to detect near-duplicates
  • E.g., Similarity > 80% => Documents are “near-duplicates”
  • Not transitive though sometimes used transitively
Computing similarity

• Features:
  • Segments of a document (natural or artificial breakpoints)
  • Shingles (Word N-Grams)
  • a rose is a rose is a rose $\rightarrow$ 4-grams are
    a_rose_is_a
    rose_is_a_rose
    is_a_rose_is
    a_rose_is_a

• Similarity measure between two docs: intersection of shingles
Jaccard coefficient

- Jaccard coefficient computes the similarity between sets.

\[
Jaccard(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}
\]

- View sets as columns of a matrix A:
  - one row for each shingle in the universe
  - one column for each document
  - \( a_{ij} = 1 \) indicates presence of shingle \( i \) in document \( j \)

- Example: \( Jaccard(C_1, C_2) = \frac{3}{6} \)
Key Observation

- For columns $C_i, C_j$, four types of rows

<table>
<thead>
<tr>
<th>Shingle</th>
<th>$C_i$</th>
<th>$C_j$</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>

- Overload notation: $A = \#$ of rows of type $A$

- Claim

$$Jaccard(C_i, C_j) = \frac{A}{A + B + C}$$
Shingles + Set Intersection

• Computing exact set intersection of shingles between all pairs of documents is expensive

• Approximate using a cleverly chosen subset of shingles from each document (a sketch)

• Estimate Jaccard coefficient based on a short sketch

![Diagram showing the process of computing set intersection using sketches](image)
Sketch of a document

• Create a “sketch vector” (of size ~200) for each document

  • Documents that share $\geq t$ (say 80%) corresponding vector elements are deemed near-duplicates

• For doc D, sketchD[ i ] is as follows:
  • Let f map all shingles in the universe to $1..2^m$ (e.g., f = fingerprinting)
  • Let $p_i$ be a random permutation on $1..2^m$
  • Pick MIN {$p_i(f(s))$} over all shingles s in D
Computing Sketch[i] for Doc1

Document 1

Start with 64-bit f(shingles)

Permute on the number line

Pick the min value
Test if Doc1.Sketch[i] = Doc2.Sketch[i]

Are these equal?

Test for 200 random permutations: $\pi_1, \pi_2, \ldots, \pi_{200}$
Minimum hashing

• Random permutations are expensive
  • If we have 1 million documents and each document has 10,000 shingles... there’s ~1 billion different shingles.
  • One needs to store 200 random permutations
  • Doing all permutations is not actually needed.

• **Answer**: implement permutations as random hash functions

  • For example: \( h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N \)

  where:

  \( a, b \) ... random integers
  \( p \) ... prime number \((p > N)\)
Min-Hashing example

### Permutations $\pi$

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

### Input matrix

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

### Signature matrix $M$

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

### Jaccard:

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

Original: 1 3 2 4
Signatures: 1 2 1 2
Similarity vs probability

• A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

• This happens with probability

\[
\frac{\text{Size_of_intersection}}{\text{Size_of_union}}
\]

• In fact, we have

\[
P(\text{minhash}(a) = \text{minhash}(b)) = \text{Jaccard(}\text{minhash}(a), \text{minhash}(b))
\]

• This is a very convenient property of MinHash for LSH.
Minimum hashing - implementation

• Input: N documents
• Create n-grams shingles
• Pick 200 random permutations, as hash functions
  • Generate and store 200 random numbers, one for each hash function.
  • Hash function \( i \) can be obtained with \( .hashCode() \) XOR random number \( i \)
• For each one of the 200 hash function permutation
  • Select the hashcode of the shingle with the lowest hashcode
• Compute N sketches: 200xN matrix
  • Each document is represented by 200 hashcodes (integers)
• Compute N*{(N-1)/2} pairwise similarities
  • Each vector now has 200 integers from the hashes.
  • Each integer corresponds to the minimum shingle of a given hash permutation.
• Choose the closest ones.
Min-Hashing example with random hashing

<table>
<thead>
<tr>
<th>DocX shingles</th>
<th>hashA()</th>
<th>hashB()</th>
<th>hashC()</th>
<th>hashD()</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a rose is a</td>
<td>103</td>
<td>19032</td>
<td>09743</td>
<td>98432</td>
<td></td>
</tr>
<tr>
<td>rose is a rose</td>
<td>1098</td>
<td>3456</td>
<td>89032</td>
<td>98743</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4539</td>
<td>6578</td>
<td>89327</td>
<td>21309</td>
<td></td>
</tr>
<tr>
<td></td>
<td>243</td>
<td>2435</td>
<td>93285</td>
<td>29873</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8876</td>
<td>7746</td>
<td>9832</td>
<td>98321</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2486</td>
<td>9823</td>
<td>30984</td>
<td>30282</td>
<td></td>
</tr>
</tbody>
</table>

Doc X minHash signature: 103, 2435, 9743, 21309, ...
Discussion

• At the end, after selecting the near-duplicate candidates,
  • ... you still must do a direct comparison,
  • ... and there is a chance of retrieving false positives.

• The $N*(N-1)/2$ pairwise similarities can be computationally prohibitive for large $N$.
  • Still manageable for small $N$, e.g. for search results.

• LSH reduces the search space (the $N$ documents).
Other hashing functions

• Other similarity based hashing methods can be used to compare documents.

• Simhash is hashing technique that generates a sequence of bits.
  • Hashcodes are more compact than with minhash.
  • Based on the cosine distance.

• In 2007 Google reported to use simhash to detect near-duplicate documents.
Locality Sensitive Hashing

Web Search
Nearest Neighbor

\[ \min_{p_i \in P} \text{dist}(q, p_i) \]
$r, \varepsilon$ - Nearest Neighbor

\[
\text{dist}(q,p1) \leq R \\
\text{dist}(q,p2) \geq cR
\]
Intuition
Locality Sensitive Hashing

• **Hashing** methods to do fast **Nearest Neighbor (NN)** Search

• Sub-linear time search by hashing highly similar examples together in a hash table

  • Take random projections of data

  • Quantize each projection with few bits

  • Strong theoretical guarantees
Locality Sensitive Hashing

• The basic idea behind LSH is to project the data into a low-dimensional binary (Hamming) space; that is, each data point is mapped to a b-bit vector, called the hash key.

• Each hash function h must satisfy the locality sensitive hashing property:

\[ p(h(a) = h(b)) = \text{sim}(a, b) \]

MinHash has this property.

Where \( \text{sim}(a, b) \in [0,1] \) is the similarity function of interest.
Definition

• A family of hash functions is called \((R, cR, p_1, p_2)\)-sensitive if for any two points \(a, b\):

  • If \(\|a - b\| \leq R\) then \(p(h(a) = h(b)) \geq p_1\)

  • If \(\|a - b\| \geq cR\) then \(p(h(a) = h(b)) \leq p_2\)

• The LSH family needs to satisfy \(p_1 > p_2\)

• What is the shape of the relation between the hashes and the similarity function? MinHash satisfy these conditions.
The ideal hash function

Probability of finding correct neighbours

Real curves.

Ideal curve.

$p_1=1$ and $p_2=0$
LSH functions for dot products

- The hashing function of LSH to produce Hash Code

\[ h_r(x) = \begin{cases} 
1, & \text{if } r^T x \geq 0 \\
0, & \text{otherwise} 
\end{cases} \]

\( r^T x \geq 0 \) is a hyperplane separating the space
$L$ sets of LSH functions

- Take random projections of data
- Quantize each projection with few bits
Multiple similarity-based hash functions

- By combining a large number of similarity-based hash functions one can find different neighbours around the query vector.
- The aggregation of the different regions has a high likelihood of containing the true neighbours.
How to search with LSH?

Original vector

\[ \text{k bits hash code} \]

\[ \text{L hash tables} \]

\[ \frac{N}{2^k} \text{ instances per bucket} \]

2^k buckets

2^k buckets

2^k buckets

...
How to search with LSH?

• For the query, which buckets should be inspected?

• Each hash table returns the instances that are in each bucket.

• The total number of instances is now much smaller than the full set of data.

• However, similarities still need to be computed in the original space.
Temporal complexity

- N vectors of D dimensions
- Hash functions generate k bits hash codes
- Points per bucket is $N/2^k$
- Cost to find the bucket of the query: $D \cdot k$
- Cost of comparison with bucket data: $D \left( \frac{N}{2^k} \right)$
- Repeat for the L hashtables

**LSH search cost:**

$$L \left( D \cdot k + D \left( \frac{N}{2^k} \right) \right) = \text{constant if } k = \log N$$
Collision probability

• Prob(a and b hashcodes match in 1 bit) = s

• Prob(all bits in a and b hashcodes match) = s^k

• Prob(no hashcodes match) = 1 - s^k

• Prob(no match is found is all L hashtables) = (1 - s^k)^L

• Prob(there is a match on at least 1 hashtable) = 1 - (1 - s^k)^L

\[ P(a, b \text{ is a candidate pair}) = 1 - (1 - s^k)^L \]
Collision probability

\[ 1 - (1 - s^k)^L \]

\[ sim(a, b) \]
Picking $L$ and $k$

Given a fixed threshold $s$.

We want choose $r$ and $b$ such that the $P(\text{Candidate pair})$ has a “step” right around $s$.
Beyond LSH: Learning Hash functions

• Standard LSH uses data independent hash functions.

• Lots of research has occurred on methods that use hash codes generated by learning methods.
  • Excellent performance.
  • Performance suffers when data distribution changes over time.

• This approach defines current state-of-the-art.
Beyond LSH: Multi-probe LSH

• The idea is to inspect similar buckets.

• A similar bucket is for a bucket that differs no more than 1 or 2 bits (hamming distances).
  • This implies k buckets to inspect.

• Requires only one hash tables, leaving free memory to store more documents in memory.

• State of the art implementation: FALCONN
Multi-probe LSH

• Replace the L hashtables by a single hash tables and inspect buckets that differ a few bits (usually 1 or 2 bits) from the matching bucket.
Spectral hashing

- Data dependent hashing with multi-probe.

What’s next?

- Data structures for very large-scale data is a very active research field.
  - Facebook, Samsung, MIT, ...

![Graph](chart.png)
The big picture

MinHash

LSH
Summary

• Information near-duplicates
  • Computational complexity

• Similarity based hash functions (MinHash)

• Locality Sensitive Hashing

• References:
