Search Engine Architecture

11. Finding Similar Items
Near-Duplicate Detection

- What’s the source of the problem?
  - Mirror pages (legit)
  - Spam farms (non-legit)
  - Additional complications (e.g., nav bars)

- Naïve algorithm:
  - Compute cryptographic hash for webpage (e.g., MD5)
  - Insert hash values into a big hash table
  - Compute hash for new webpage: collision implies duplicate

- What’s the issue?
- Intuition:
  - Hash function needs to be tolerant of minor differences
  - High similarity implies higher probability of hash collision

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Minhash

• Seminal algorithm for near-duplicate detection of webpages
  • Used by AltaVista
  • For details see Broder et al. (1997)

• Setup:
  • Documents (HTML pages) represented by shingles ($n$-grams)
  • Jaccard similarity: dups are pairs with high similarity

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Representation

• Sets:
  • $A = \{e_1, e_3, e_7\}$
  • $B = \{e_3, e_5, e_7\}$
• Can be equivalently expressed as matrices:

<table>
<thead>
<tr>
<th>Element</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$e_2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e_4$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$e_5$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$e_6$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$e_7$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Jaccard Similarity

Let:

\[ M_{00} = \# \text{ rows where both elements are 0} \]
\[ M_{11} = \# \text{ rows where both elements are 1} \]
\[ M_{01} = \# \text{ rows where A=0, B=1} \]
\[ M_{10} = \# \text{ rows where A=1, B=0} \]

\[
J(A, B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}
\]

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Minhash

- **Computing minhash**
  - Start with the matrix representation of the set
  - Randomly permute the rows of the matrix
  - minhash is the first row with a “one”

- **Example:**

  \[
  \begin{array}{c|cc}
  \text{Element} & \text{A} & \text{B} \\
  \hline
  e_1 & 1 & 0 \\
  e_2 & 0 & 0 \\
  e_3 & 1 & 1 \\
  e_4 & 0 & 0 \\
  e_5 & 0 & 1 \\
  e_6 & 0 & 0 \\
  e_7 & 1 & 1 \\
  \end{array}
  \quad
  \begin{array}{c|cc}
  \text{Element} & \text{A} & \text{B} \\
  \hline
  e_6 & 0 & 0 \\
  e_2 & 0 & 0 \\
  e_5 & 0 & 1 \\
  e_3 & 1 & 1 \\
  e_7 & 1 & 1 \\
  e_4 & 0 & 0 \\
  e_1 & 1 & 0 \\
  \end{array}
  \]

  \[h(A) = e_3 \quad h(B) = e_5\]
Minhash and Jaccard

<table>
<thead>
<tr>
<th>Element</th>
<th>A</th>
<th>B</th>
<th>$M_{00}$</th>
<th>$M_{01}$</th>
<th>$M_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_6$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>$M_{00}$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>$M_{00}$</td>
</tr>
<tr>
<td>$e_5$</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>$M_{01}$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>$M_{11}$</td>
</tr>
<tr>
<td>$e_7$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>$M_{11}$</td>
</tr>
<tr>
<td>$e_4$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>$M_{00}$</td>
</tr>
<tr>
<td>$e_1$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>$M_{10}$</td>
</tr>
</tbody>
</table>

$P[h(A) = h(B)] = J(A, B)$

\[
\frac{M_{11}}{M_{01} + M_{10} + M_{11}} \quad \frac{M_{11}}{M_{01} + M_{10} + M_{11}}
\]

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
To Permute or Not to Permute?

• Permutations are expensive
• Interpret the hash value as the permutation
• Only need to keep track of the minimum hash value
  • Can keep track of multiple minhash values at once

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Extracting Similar Pairs (LSH)

- We know: \( P[h(A) = h(B)] = J(A, B) \)
- Task: discover all pairs with similarity greater than \( s \)
- Algorithm:
  - For each object, compute its minhash value
  - Group objects by their hash values
  - Output all pairs within each group
- Analysis:
  - Probability we will discovered all pairs is \( s \)
  - Probability that any pair is invalid is \((1 – s)\)

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Two Minhash Signatures

- **Task:** discover all pairs with similarity greater than $s$

- **Algorithm:**
  - For each object, compute two minhash values and concatenate together into a signature
  - Group objects by their signatures
  - Output all pairs within each group

- **Analysis:**
  - Probability we will discovered all pairs is $s^2$
  - Probability that any pair is invalid is $(1 - s)^2$

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
**k Minhash Signatures**

- Task: discover all pairs with similarity greater than $s$
- Algorithm:
  - For each object, compute $k$ minhash values and concatenate together into a signature
  - Group objects by their signatures
  - Output all pairs within each group
- Analysis:
  - Probability we will discovered all pairs is $s^k$
  - Probability that any pair is invalid is $(1 - s)^k$

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
**n different k Minhash Signatures**

- **Task:** discover all pairs with similarity greater than $s$
- **Algorithm:**
  - For each object, compute $n$ sets $k$ minhash values
  - For each set, concatenate $k$ minhash values together
  - Within each set:
    - Group objects by their signatures
    - Output all pairs within each group
  - De-dup pairs
- **Analysis:**
  - Probability we will miss a pair is $(1 - s^k)^n$
  - Probability that any pair is invalid is $n(1 - s)^k$

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Practical Notes

- In some cases, checking all candidate pairs may be possible
  - Time cost is small relative to everything else
  - Easy method to discard false positives
- Most common practical implementation:
  - Generate $M$ minhash values, randomly select $k$ of them $n$ times
  - Reduces amount of hash computations needed
- Determining “authoritative” version is non-trivial
MapReduce Implementation

• Map over objects:
  • Generate $M$ minhash values, randomly select $k$ of them $n$ times
  • Each draw yields a signature: emit as intermediate key, value is object id
• Shuffle/sort:
• Reduce:
  • Receive all object ids with same signature, emit clusters
• Second pass to de-dup and group clusters

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Offline vs. Online

- **Batch formulation of the problem:**
  - Discover all pairs with similarity greater than $s$
  - Useful for post-hoc batch processing of web crawl
- **Online formulation of the problem:**
  - Given new webpage, is it similar to one I’ve seen before?
  - Useful for incremental web crawl processing

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Online Similarity Querying

• Preparing the existing collection:
  • For each object, compute $n$ sets of $k$ minhash values
  • For each set, concatenate $k$ minhash values together
  • Keep each signature in hash table (in memory)
  • Note: can parallelize across multiple machines

• Querying and updating:
  • For new webpage, compute signatures and check for collisions
  • Collisions imply duplicate (determine which version to keep)
  • Update hash tables

Source: Lin et al. Data-Intensive Computing with MapReduce, UMD Spring 2013
Mining Bitext from Wikipedia

• Task: extract similar article pairs in different languages
  • Why is this useful?
  • Why not just use inter-wiki links?
• Use LSH!
  • Why is this non-trivial?
Machine Translation

**Tradeoffs?**

"Vector" Translation

\[
\text{tf}'(e, D) = \sum_{f \in F} p(f | e) \cdot \text{tf}(f, D)
\]

\[
\text{df}'(e) = \sum_{f \in F} p(f | e) \cdot \text{df}(f)
\]
Translations are noisy!

Question: To what extent are “ground truth” mutual translation pairs similar?
Architecture

- **German articles**
- **English articles**

**Preprocess**

**Signature generation**

**LSH**

**document vectors (English)**

**Signatures**

**Similar article pairs**
Evaluation Setup

• Collection: 3.44m English + 1.47m German Wikipedia
  • Brute force: 5.05 trillion comparisons
• Task: extract similar documents with cosine similarity > 0.3
• Representation: 1000 bit random projections
• Ground truth:
  • Sample ~1000 German Wikipedia articles
• Evaluation metrics:
  • Time
  • Relative cost
  • Recall
Results

(16 node cluster)

Time (hours) vs. Window Size (B) for 100, 200, and 300 tables. Each line represents a different number of tables, with 100 tables in green, 200 tables in red, and 300 tables in blue. The x-axis represents the window size in bytes, ranging from 0 to 2000. The y-axis represents the time in hours, ranging from 0 to 9. The graph shows the relationship between window size and time for different table sizes.

The figure indicates that as the window size increases, the time required to process the document pairs also increases. This relationship is consistent across different table sizes, with the time increasing at a slightly higher rate for larger numbers of tables.

The graph also includes a note indicating that there are 64 million document pairs, and a mention of the (16 node cluster) setup.
Sources of Error

• Hamming distance computation introduces error!
  • 1000-bit random projections yields ~0.04 absolute error
  • Maximum obtainable recall is 0.76
• Why not just compute cosine similarity?
  • 20 times slower than computing hamming distance

Let’s renormalize with respect to hamming distance upper bound...
No Free Lunch!

![Graph showing the tradeoff between relative effectiveness and cost. Points indicate that as cost increases, so does effectiveness, but there's a limit to this relationship. Labels include: 95% recall, 39% cost; 99% recall, 70% cost; for 500 tables.](image-url)
No Free Lunch!

100% recall? Don’t use LSH!