Finding Similar Items: Locality Sensitive Hashing

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu
Scene Completion Problem

[Hays and Efros, SIGGRAPH 2007]
Scene Completion Problem
Scene Completion Problem

10 nearest neighbors from a collection of 20,000 images

[Hays and Efros, SIGGRAPH 2007]
Scene Completion Problem

10 nearest neighbors from a collection of 2 million images
Many problems can be expressed as finding “similar” sets:

- Find near-neighbors in high-dimensional space

Examples:

- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
- Users who visited the similar websites
Last time: Finding frequent pairs

Naïve solution:
Single pass but requires space quadratic in the number of items

N … number of distinct items
K … number of items with support $\geq s$

A-priori:
First pass: Find frequent singletons
For a pair to be a candidate for a frequent pair, its singletons have to be frequent!
Second pass:
Count only candidate pairs!
Relation to Previous Lecture

- **Last time:** Finding frequent pairs
- **Further improvement:** PCY

**Pass 1:**
- Count exact frequency of each item:
- Take pairs of items \{i,j\}, hash them into \(B\) buckets and count of the number of pairs that hashed to each bucket:

Basket 1: \{1,2,3\}
Pairs: \{1,2\} \{1,3\} \{2,3\}
Relation to Previous Lecture

- **Last time:** Finding frequent pairs
- **Further improvement:** PCY

### Pass 1:
- Count exact frequency of each item:
- Take pairs of items \{i,j\}, hash them into $B$ buckets and count of the number of pairs that hashed to each bucket:

### Pass 2:
- For a pair \{i,j\} to be a **candidate for a frequent pair**, its singletons have to be frequent and it has to hash to a frequent bucket!
Last time: Finding frequent pairs

Further improvement: PCY

- **Pass 1:** Count exact frequency of each item.
  - Take pairs of items \{i, j\}, hash them into buckets and count of the number of pairs that hashed to each bucket.

- **Pass 2:** For a pair \{i, j\} to be a candidate for a frequent pair, its singletons have to be frequent and its hash has to hash to a frequent bucket!

**Previous lecture: A-priori**

**Main idea:** Candidates

Instead of keeping a count of each pair, only keep a count for candidate pairs!

**Today’s lecture: Find pairs of similar docs**

**Main idea:** Candidates

- **Pass 1:** Take documents and hash them to buckets such that documents that are similar hash to the same bucket.
- **Pass 2:** Only compare documents that are candidates (i.e., they hashed to a same bucket)

**Benefits:** Instead of \(N^2\) comparisons, we need \(O(N)\) comparisons to find similar documents.

Basket 1: \{1, 2, 3\}
- Pairs: \{1, 2\} \{1, 3\} \{2, 3\}

Basket 2: \{1, 2, 4\}
- Pairs: \{1, 2\} \{1, 4\} \{2, 4\}

Buckets: 1 \(\rightarrow\) 3

1/15/2013

Jure Leskovec, Stanford C246: Mining Massive Datasets
Goal: Find near-neighbors in high-dim. 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

Today: Jaccard distance (/similarity)

- The Jaccard Similarity/Distance of two sets is the size of their intersection / the size of their union:
  - \( \text{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \)
  - \( d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \)

3 in intersection
8 in union
Jaccard similarity = 3/8
Jaccard distance = 5/8
Goal: Given a large number ($N$ in the millions or billions) of text documents, find pairs that are “near duplicates”

Applications:
- Mirror websites, or approximate mirrors
  - Don’t want to show both in a search
- Similar news articles at many news sites
  - Cluster articles by “same story”

Problems:
- Many small pieces of one 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 main memory
3 Essential Steps for Similar Docs

1. **Shingling**: Convert documents 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 from similar documents
   - Candidate pairs!
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
Step 1: **Shingling**: Convert documents to sets

**Document**

Shingling

The set of strings of length $k$ that appear in the document
Step 1: Shingling: Convert documents to sets

Simple approaches:
- Document = set of words appearing in document
- Document = set of “important” words
- Don’t work well for this application. Why?

Need to account for ordering of words!
A different way: 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.
- Assume tokens = characters for examples.

**Example:** $k=2$; document $D_1 = \text{abcab}$

Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$

- **Option:** Shingles as a bag (multiset), count ab twice: $S'(D_1) = \{ab, bc, ca, ab\}$
To compress long shingles, we can hash them to (say) 4 bytes.

Represent a doc by the set of hash values of its $k$-shingles.

**Idea:** Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

**Example:** $k=2$; document $D_1 = \text{abcab}$

Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$

Hash the singles: $h(D_1) = \{1, 5, 7\}$
Similarity Metric for Shingles

- **Document** $D_1 = \text{set of } k\text{-shingles } C_1 = S(D_1)$
- Equivalently, each document is a 0/1 vector in the space of $k$-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- A natural similarity measure is the Jaccard similarity:
  \[
  \text{Sim}(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}
  \]
Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Caveat: 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
Suppose we need to find near-duplicate documents among N=1 million documents.

Naïvely, we’d have to compute pairwise Jaccard similarities for every pair of docs:
- i.e, $N(N-1)/2 \approx 5 \times 10^{11}$ comparisons
- At $10^5$ secs/day and $10^6$ comparisons/sec, it would take 5 days

For $N = 10$ million, it takes more than a year...
MinHashing

Step 2: **Minhashing**: Convert large sets to short signatures, while preserving similarity.
Many similarity problems can be formalized as **finding subsets that have significant intersection**

- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- **Interpret set intersection as bitwise AND, and set union as bitwise OR**

**Example:** $C_1 = 10111; C_2 = 10011$
- Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = $3/4$
- $d(C_1, C_2) = 1 - \text{(Jaccard similarity)} = 1/4$
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
- \( d(C_1, C_2) = 1 - \text{(Jaccard similarity)} = 3/6 \)
Outline: Finding Similar Columns

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix

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

- **Approach:**
  1) **Signatures of columns:** small summaries of columns
  2) Examine pairs of signatures to find similar columns
     - **Essential:** Similarities of signatures & 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)
Key idea: “hash” each column $C$ to a small signature $h(C)$, such that:

1. $h(C)$ is small enough that the signature fits in RAM
2. $\text{sim}(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$

Goal: Find a hash function $h(\cdot)$ such that:

1. if $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
2. if $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Hash documents into buckets, and expect that “most” pairs of near duplicate docs hash into the same bucket!
Goal: Find a hash function $h(\cdot)$ 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. $h(C_1) \neq h(C_2)$

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 Jaccard similarity: Min-hashing
Imagine the rows of the boolean matrix permuted under random permutation $\pi$

Define a “hash” function $h_{\pi}(C)$ = the number of the first (in the permuted order $\pi$) row in which column $C$ has value 1:

$$h_{\pi}(C) = \min_{\pi} \pi(C)$$

Use several (e.g., 100) independent hash functions to create a signature of a column
Min-Hashing Example

Permutation $\pi$

2\textsuperscript{nd} element of the permutation is the first to map to a 1

Input matrix (Shingles x Documents)

Signature matrix $M$

2\textsuperscript{nd} element of the permutation is the first to map to a 1

4\textsuperscript{th} element of the permutation is the first to map to a 1

Note: Another (equivalent) way is to store row indexes:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

1/15/2013

Jure Leskovec, Stanford C246: Mining Massive Datasets
Surprising Property

- Choose a random permutation $\pi$
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- **Why?**
  - Let $X$ be a document (set of shingles)
  - Then: $\Pr[\pi(x) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $x \in X$ is mapped to the $\min$ element
  - Let $x$ be s.t. $\pi(x) = \min(\pi(C_1 \cup C_2))$
  - Then either: $\pi(x) = \min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = \min(\pi(C_2))$ if $x \in C_2$
  - So the prob. that both are true is the prob. $x \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2|/|C_1 \cup C_2| = \text{sim}(C_1, C_2)$
Four Types of Rows

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

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</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>

- $a = \#$ rows of type A, etc.
- **Note:** $\text{sim}(C_1, C_2) = \frac{a}{a + b + c}$
- **Then:** $\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)$

  - Look down the cols $C_1$ and $C_2$ until we see a 1
  - If it’s a type-A row, then $h(C_1) = h(C_2)$
  - If a type-B or type-C row, then not
Similarity for Signatures

- We know: \( \Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = sim(C_1, C_2) \)
- Now generalize to multiple hash functions

- The *similarity of two signatures* is the fraction of the hash functions in which they agree

- **Note:** Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures
Min-Hashing Example

Permutation $\pi$

<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 (Shingles x Documents)

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

Signature matrix $M$

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></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>0.67</td>
<td>1.00</td>
</tr>
<tr>
<td>Sig/Sig</td>
<td>0.75</td>
<td>0.75</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Note: Another (equivalent) way is to store row indexes:

\[ \begin{array}{cccc}
1 & 5 & 1 & 5 \\
2 & 3 & 1 & 3 \\
6 & 4 & 6 & 4
\end{array} \]
Pick K=100 random permutations of the rows

Think of $\text{sig}(C)$ as a column vector

$\text{sig}(C)[i] = \text{ according to the } i\text{-th permutation, the index of the first row that has a 1 in column } C$

$\text{sig}(C)[i] = \min (\pi_i(C))$

Note: The sketch (signature) of document C is small -- ~100 bytes!

We achieved our goal! We “compressed” long bit vectors into short signatures
Implementational Trick

- Permuting rows even once is prohibitive
- **Row hashing!**
  - Pick $K = 100$ hash functions $k_i$
  - Ordering under $k_i$ gives a random row permutation!
- **One-pass implementation**
  - For each column $C$ and hash-func. $k_i$ keep a “slot” for the min-hash value
  - Initialize all $\text{sig}(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row $j$ has 1 in column $C$
    - Then for each $k_i$:
      - If $k_i(j) < \text{sig}(C)[i]$, then $\text{sig}(C)[i] \leftarrow k_i(j)$

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 ($p > N$)
Step 3: **Locality-sensitive hashing:**
Focus on pairs of signatures likely to be from similar documents

**Locality Sensitive Hashing**

- **Docum-*...*nt**
- **Shingling**
  - The set of strings of length $k$ that appear in the document
- **MinHashing**
- **Locality-sensitive Hashing**
  - **Candidate pairs:** those pairs of signatures that we need to test for similarity

**Signatures:** short integer vectors that represent the sets, and reflect their similarity
### LSH: First Cut

- **Goal:** Find documents with Jaccard similarity at least $s$ (for some similarity threshold, e.g., $s=0.8$)

- **LSH – General idea:** Use a function $f(x,y)$ that tells whether $x$ and $y$ is a **candidate pair:** a pair of elements whose similarity must be evaluated

- **For minhash matrices:**
  - Hash columns of **signature matrix $M$** to many buckets
  - Each pair of documents that hashes into the same bucket is a **candidate pair**
Candidates from Minhash

- Pick a similarity threshold $s$ (0 < $s$ < 1)

- Columns $x$ and $y$ of $M$ are a candidate pair if their signatures agree on at least fraction $s$ of their rows:
  $$M(i, x) = M(i, y)$$ for at least frac. $s$ values of $i$

  - We expect documents $x$ and $y$ to have the same (Jaccard) similarity as is the similarity of their signatures
LSH for Minhash

- **Big idea:** Hash columns of signature matrix $M$ several times

- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability

- **Candidate pairs** are those that hash to the same bucket
Partition $M$ into $b$ Bands

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

$M$ is a signature matrix with $r$ rows per band.
b bands

One signature

Signature matrix $M$
Partition M into Bands

- 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 non-similar pairs
Hashing Bands

Columns 2 and 6 are probably identical (candidate pair)

Columns 6 and 7 are surely different.

Matrix $M$

$B$ bands

$r$ rows
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.
Assume the following case:

- Suppose 100,000 columns of $M$ (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take 40Mb
- Choose $b = 20$ bands of $r = 5$ integers/band

**Goal:** Find pairs of documents that are at least $s = 0.8$ similar
C₁, C₂ are 80% Similar

- **Find pairs of** ≥ s=0.8 similarity, set b=20, r=5
- **Assume:** sim(C₁, C₂) = 0.8
  - Since sim(C₁, C₂) ≥ s, we want C₁, C₂ to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)
- **Probability C₁, C₂ identical in one particular band:** (0.8)⁵ = 0.328
- **Probability C₁, C₂ 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**
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 $s$
Pick:
- the number of minhashes (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 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up
Analysis of LSH – What We Want

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

Probability

Probability of sharing a bucket

No chance if $t < s$

Similarity threshold $s$

Probability = 1 if $t > s$
What 1 Band of 1 Row Gives You

Remember:

Probability of equal hash-values = similarity

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

Pick any band ($r$ rows)
  - Prob. that all rows in band equal = $t^r$
  - Prob. that some row in band unequal = $1 - t^r$

Prob. that no band identical = $(1 - t^r)^b$

Prob. that at least 1 band identical = $1 - (1 - t^r)^b$
What $b$ Bands of $r$ Rows Gives You

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

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

- All rows of a band are equal
- Some row of a band unequal
- No bands identical
- At least one band identical

Probability of sharing a bucket

Similarity $t = \text{sim}(C_1, C_2)$ of two sets
Example: $b = 20; r = 5$

- **Similarity threshold $s$**
- **Prob. that at least 1 band is identical:**

<table>
<thead>
<tr>
<th>$s$</th>
<th>$1-(1-s^r)^b$</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>
Picking $r$ and $b$: The S-curve

- **Picking $r$ and $b$ to get the best S-curve**
  - 50 hash-functions ($r=5$, $b=10$)

![Graph showing the S-curve with blue and green areas representing False Negative and False Positive rates, respectively.](image-url)
Tune $M, b, r$ 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 documents.
Summary: 3 Steps

- **Shingling**: Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-hashing**: Convert large sets to short signatures, while preserving similarity
  - We used *similarity preserving hashing* to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-sensitive hashing**: Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find *candidate pairs* of similarity $\geq s$