Finding Similar Items: Locality Sensitive Hashing
New thread: High dimension data

- High dim. data: Locality sensitive hashing, Clustering, Dimensionality reduction
- Graph data: PageRank, SimRank, Network Analysis
- Infinite data: Filtering data streams, Web advertising, Queries on streams
- Machine learning: SVM, Decision Trees, Perceptron, kNN
- Apps: Recommender systems, Association Rules, Duplicate document detection
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 similar websites
Problem for today’s lecture

- **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?
Relation to Previous Lecture

- **Last time:** Finding frequent pairs

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

Items 1…N

- Count of pair \{i,j\} in the data

\[ N \ldots \text{number of distinct items} \]
\[ K \ldots \text{number of items with support } \geq s \]

**A-Priori:**
First pass: Find frequent singletons
For a pair to be a frequent pair candidate, its singletons have to be frequent!
Second pass:
Count only candidate pairs!
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\}
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 \{i\}, \{j\} have to be frequent and the pair has to hash to a frequent bucket!

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

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

Items 1…N

Buckets 1…B
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 \(i\) 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 of 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 \(O(N^2)\) comparisons, we need \(O(N)\) comparisons to find similar documents
Finding Similar Items
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 of two sets is the size of their intersection divided by the size of their union:
  \[ \text{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \]
- Jaccard distance: \[ d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \]

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 documents, find “near duplicate” pairs

Applications:
- Mirror websites, or approximate mirrors
  - Don’t want to show both in search results
- 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. **Min-Hashing**: 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

Shingling

The set of strings of length $k$ that appear in the document
Documents as High-Dim Data

- **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 \textit{k-shingle} (or \textit{k-gram}) for a document is a sequence of \textit{k} tokens that appears in the doc

- Tokens can be characters, words or something else, depending on the application
- Assume tokens = characters for examples

\textbf{Example:} \texttt{k=2}; document \texttt{D_1} = abcab
Set of 2-shingles: \texttt{S(D_1)} = \{ab, bc, ca\}

- \textbf{Option:} Shingles as a bag (multiset), count \texttt{ab} twice: \texttt{S'(D_1)} = \{ab, bc, ca, ab\}
Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document 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: $h(D_1) = \{1, 5, 7\}$
Document $D_1$ is a set of its $k$-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:

$$ sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} $$

\[ \text{1/13/2014} \]

Jure Leskovec, Stanford C246: Mining Massive Datasets
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 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 $10^6$ 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*

**Document**

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

**Signatures**: short integer vectors that represent the sets, and reflect their 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) = \( 3/4 \)
  - Distance: \( d(C_1,C_2) = 1 - \text{(Jaccard similarity)} = 1/4 \)
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
  - $d(C_1, C_2) = 1 - (\text{Jaccard similarity}) = 3/6$
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
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 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:
  - 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. 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 the Jaccard similarity: It is called **Min-Hashing**
Min-Hashing

- Imagine the rows of the boolean matrix permuted under random permutation \( \pi \)

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

\[
h_\pi(C) = \min_\pi \pi(C)
\]

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

Permutation $\pi$

Input matrix (Shingles x Documents)

Signature matrix $M$

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

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

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

1/13/2014
The Min-Hash 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 doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the $\min$ 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 both are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)$

One of the two cols had to have 1 at position $y$
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 Min-Hash 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>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Signature matrix $M$

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</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>0.67</td>
<td>1.00</td>
</tr>
<tr>
<td>Sig/Sig</td>
<td>0.75</td>
<td>0.75</td>
<td>0.67</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Min-Hash Signatures

- Pick K=100 random permutations of the rows
- Think of \( \text{sig}(C) \) as a column vector
- \( \text{sig}(C)[i] = \) according to the \( i \)-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 \( \sim 100 \) bytes!

- We achieved our goal! We “compressed” long bit vectors into short signatures
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
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 Min-Hash 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 Min-Hash

- 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: 
  
  \[
  M(i, x) = M(i, y) \quad \text{for at least frac.} \ s \ \text{values of} \ i
  \]

  - We expect documents \( x \) and \( y \) to have the same (Jaccard) similarity as their signatures
**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

**Signature matrix $M$**

$b$ bands

$r$ rows per band

One signature

2 1 4 1
1 2 1 2
2 1 2 1
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
Columns 2 and 6 are probably identical (candidate pair).

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.
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
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 $1/3000$th 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 $s$
LSH Involves a Tradeoff

- **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 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 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

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

- Similarity \( t = \text{sim}(C_1, C_2) \) of two sets
- Probability of sharing a bucket
- At least one band identical
- No bands identical
- Some row of a band unequal
- All rows of a band are equal

\[
1 - (1 - t^r)^b
\]
**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 S-curve with probability sharing a bucket on the y-axis and similarity on the x-axis. Blue area: False Negative rate, Green area: False Positive rate.](image-url)
LSH Summary

- 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 $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - 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 $\geq s$