Recap: Finding similar documents

- **Goal:** Given a large number (N in the millions or billions) of text documents, find pairs that are “near duplicates”

- **Application:**
  - Detect mirror and approximate mirror sites/pages:
    - Don’t want to show both in a web search

- **Problems:**
  - Many small pieces of one doc can appear out of order in another
  - Too many docs to compare all pairs
  - Docs are so large or so many that they cannot fit in main memory
Recap: 3 Essential Steps

1. **Shingling**: Convert documents to large sets of items

2. **Minhashing**: Convert large sets into short signatures, while preserving similarity

3. **Locality-sensitive hashing**: Focus on pairs of signatures likely to be from similar documents
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.
Recap: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the document

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

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

- A natural similarity measure is the Jaccard similarity:
  \[
  \text{Sim}(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}
  \]
Recap: Min-hashing

- Prob. $h_{\pi}(C_1) = h_{\pi}(C_2)$ is the same as $Sim(C_1, C_2)$:
  \[ \Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = Sim(C_1, C_2) \]
Recap: LSH

- Hash cols of signature matrix $M$. Similar columns likely hash to the same bucket.
  - Cols. $x$ and $y$ are a candidate pair if $M(i, x) = M(i, y)$ for at least frac. $s$ values of $i$.
  - Divide matrix $M$ into $b$ bands of $r$ rows.

- $\text{Sim}(C_1, C_2) = s$
- Prob. that at least 1 band identical $= 1 - (1 - s^r)^b$
- Given $s$, tune $r$ and $b$ to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.

$b=20, r=5$

<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>

1/19/2011
S-curves as a func. of $b$ and $r$

![Graphs showing probability of sharing a bucket as a function of similarity for different values of $r$ and $b$.]
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.

**Theory of LSH**
Theory of LSH

- We have used LSH to find similar documents
  - In reality, columns in large sparse matrices with high Jaccard similarity
  - e.g., customer/item purchase histories

- Can we use LSH for other distance measures?
  - e.g., Euclidean distances, Cosine distance
  - Let’s generalize what we’ve learned!
For min-hash signatures, we got a min-hash function for each permutation of rows.

An example of a family of hash functions:

- A “hash function” is any function that takes two elements and says whether or not they are “equal”
  - **Shorthand**: \( h(x) = h(y) \) means “\( h \) says \( x \) and \( y \) are equal.”
- A **family** of hash functions is any set of hash functions
  - A 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

- Suppose we have 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 $S$:
  1. If $d(x, y) \leq d_1$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at least $p_1$
  2. If $d(x, y) \geq d_2$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at most $p_2$
A $(d_1, d_2, p_1, p_2)$-sensitive function

High probability; at least $p_1$

Low probability; at most $p_2$
Let:

- $S$ = sets,
- $d$ = Jaccard distance,
- $H$ is family of minhash functions for all permutations of rows

Then 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.
Claim: $H$ is a $(\frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3})$-sensitive family for $S$ and $d$.

If distance $\leq \frac{1}{3}$ (so similarity $\geq \frac{2}{3}$) then probability that min-hash values agree is $\geq \frac{2}{3}$

For Jaccard similarity, minhashing gives us 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
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.

Two constructions:
- **AND** construction like “rows in a band”
- **OR** construction like “many bands”
AND 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 $i$

- 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

- Proof: Use the fact that $h_i$'s are independent
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 $1$ $i$.

**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.

**Proof:** Use the fact that $h_i$’s are independent.
**Effect of AND and OR Constructions**

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

$$y = 1 - (1 - x^r)^b$$
Composing Constructions

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

- **Example**: Take \( H \) and construct \( H' \) by the AND construction with \( r = 4 \). Then, from \( H' \), construct \( H'' \) by the OR construction with \( b = 4 \)
Table for Function $1-(1-p^4)^4$

<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>

**Example:** Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.8785,.0064)-sensitive family.
Apply a b-way OR construction followed by an r-way AND construction

Transforms probability $p$ into $(1-(1-p)^b)^r$.

- The same S-curve, mirrored horizontally and vertically

**Example**: Take $H$ and construct $H'$ by the OR construction with $b = 4$. Then, from $H'$, construct $H''$ by the AND construction with $r = 4$. 
Table for Function $(1-(1-p)^4)^4$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$(1-(1-p)^4)^4$</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>

**Example:** Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.9936,.1215)-sensitive family.
Example: Apply the (4,4) OR-AND construction followed by the (4,4) AND-OR construction

Transforms a (0.2,0.8,0.8,.02)-sensitive family into a (0.2,0.8,0.9999996,0.0008715)-sensitive family

Note this family uses 256 (=4*4*4*4) of the original hash functions
Pick any two distances \( x < y \)

Start with a \((x, y, (1-x), (1-y))\)-sensitive family

Apply constructions to produce \((x, y, p, q)\)-sensitive family, where \(p\) is almost 1 and \(q\) is almost 0

The closer to 0 and 1 we get, the more hash functions must be used
For cosine distance,
\[ d(A, B) = \theta = \arccos\left(\frac{A \cdot B}{\|A\|\|B\|}\right) \]
there is a technique called
**Random Hyperplanes**
- Technique similar to minhashing

**A \(d_1,d_2,(1-d_1/180),(1-d_2/180)\)-sensitive family for any \(d_1\) and \(d_2\).**

**Reminder:** \((d_1,d_2,p_1,p_2)\)-sensitive

1. If \(d(x,y) \leq d_1\), then prob. that \(h(x) = h(y)\) is at least \(p_1\)
2. If \(d(x,y) \geq d_2\), then prob. that \(h(x) = h(y)\) is at most \(p_2\)
Random Hyperplanes

- Pick a random vector \( v \), which determines a hash function \( h_v \) with two buckets

- \( h_v(x) = +1 \) if \( v \cdot x > 0 \); \( = -1 \) if \( v \cdot x < 0 \)

- LS-family \( \mathbf{H} \) = set of all functions derived from any vector

- **Claim:** For points \( x \) and \( y \),
  \[
  \Pr[h(x) = h(y)] = 1 - \frac{d(x,y)}{180}
  \]
Proof of Claim

Look in the plane of $x$ and $y$.

Hyperplane normal to $\nu$

$h(x) = h(y)$

$\text{Prob[Red case]} = \frac{\theta}{180}$

Hyperplane normal to $\nu$

$h(x) \neq h(y)$
Signatures for Cosine Distance

- Pick some number of random vectors, and hash your data for each vector
- The result is a signature (sketch) of +1’s and –1’s for each data point
- Can be used for LSH like the minhash signatures for Jaccard distance
- Amplified using AND and OR constructions
How to pick random vectors?

- Expensive to pick a random vector in $M$ dimensions for large $M$
  - $M$ random numbers

- A more efficient approach
  - It suffices to consider only vectors $\nu$ consisting of $+1$ and $-1$ components
  - Why is this more efficient?
**LSH for Euclidean Distance**

- **Simple idea**: Hash functions correspond to lines
- Partition the line into buckets of size $a$
- Hash each point to the bucket containing its projection onto the line
- Nearby points are always close; distant points are rarely in the same bucket
Points at distance \( d \)

If \( d \ll a \), then the chance the points are in the same bucket is at least \( 1 - \frac{d}{a} \).
If $d >> a$, $\theta$ must be close to $90^\circ$ for there to be any chance points go to the same bucket.
If points are distance $d \leq a/2$, prob. they are in same bucket $\geq 1 - d/a = \frac{1}{2}$

If points are distance $> 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$

- $\cos \theta \leq \frac{1}{2}$
- $60 \leq \theta \leq 90$
- I.e., at most $1/3$ probability.

Yields a $(a/2, 2a, 1/2, 1/3)$-sensitive family of hash functions for any $a$

Amplify using AND-OR cascades
For previous distance measures, we could start with an \((x, y, p, q)\)-sensitive family for any \(x < y\), and drive \(p\) and \(q\) to 1 and 0 by AND/OR constructions.

Here, we seem to need \(y \geq 4x\).
But as long as \( x < y \), the probability of points at distance \( x \) falling in the same bucket is greater than the probability of points at distance \( y \) doing so.

Thus, the hash family formed by projecting onto lines is an \((x, y, p, q)\)-sensitive family for some \( p > q \).

Then, amplify by AND/OR constructions.