Lecture III Finding Nearest Neighbors in High Dimensions

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

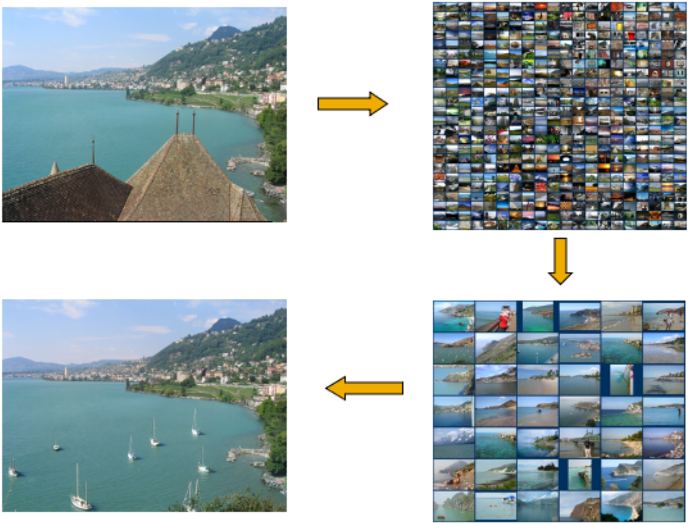
CSE 547/STAT 548
Spring 2025

Reading MMDS Ch.: 3. Finding similar items HTF Ch.:, Murphy Ch.: **Reading:** Lecture 16 notes by Moses Charikar, section 3.2; optionally Cormen, Leiserson, Rivest, Stein "Introduction to Algorithms", chapter on hashing.

Thanks to `mmds.com` (Leskovec, Rajaraman and Ullman) and `randorithms.com` (Ben Coleman)

[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem

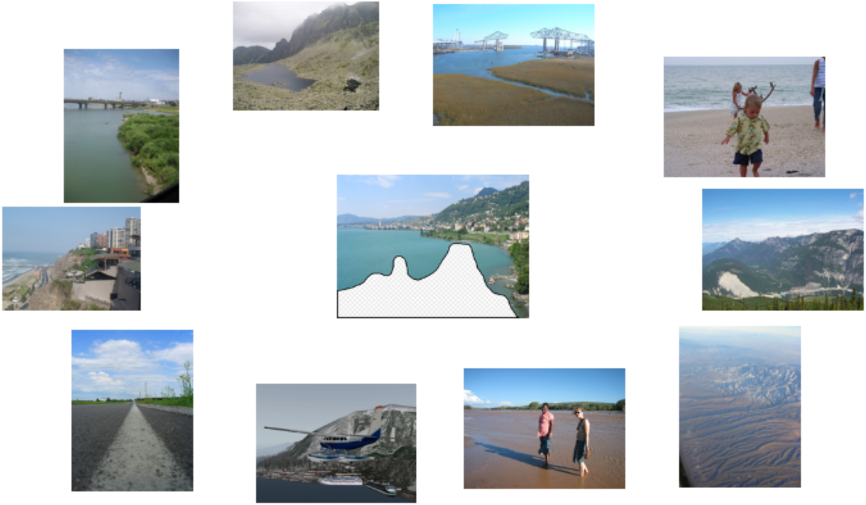


[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem

[Hays and Efros, SIGGRAPH 2007]

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

A Common Metaphor

- **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

The problem: finding neighbors in high dimensions

- Given $\mathcal{D}$ of size n in $\mathbb{R}^d$, and given a **query point** x find the neighbors of x in $\mathcal{D}$
  - here: all neighbors in radius r
  - sometimes the k nearest-neighbors
  - sometimes just 1 neighbor
- query point can be in $\mathcal{D}$, e.g. in clustering, dimension reduction, or not (e.g. retrieval, image completion)
- $n \ll 10^6$ and $d > 10^2$
- Brute force (suppose we need neighbors of all $x_i \in \mathcal{D}$)
  - compute time $\mathcal{O}(n^2 d)$ – Too large!
- Can we do it exactly in subquadratic time? Probably NO
  - [if the SETH (Strong Exponential Time Conjecture) holds]
- Rephrased problem: find approximate nearest neighbors
  - e.g. if x has neighbor $x' \in \mathcal{D}$ at distance r, return an $x'' \in \mathcal{D}$ at distance $\leq cr$
  - with $c > 1$ some constant, and w.h.p.[1], usually measured by a confidence δ
  - we measure performance of algorithm as function of (c, r, δ)

[1] with high probability

Distance and similarity functions

Distances

- **Euclidean** $x, x' \in \mathbb{R}^d$, $\quad d_{Euclid}(x, x') = \|x - x'\| = \sqrt{x^T x + (x')^T x' - 2x^T x'}$
- **L1 (Manhattan)** $x, x' \in \mathbb{R}^d$ $d_{L1}(x, x') = \|x - x'\|_1$
- **Hamming** $x, x' \in \{0,1\}^d$ $d_H(x, x') = x^T x + (x')^T x' - 2x^T x' = \#x + \#x' - 2\#(x \cap x')$

Similarities

- **cosine** $x, x' \in \mathbb{R}^d$ or $\{0,1\}^d$ $\cos(x, x') = \dfrac{x^T x'}{\sqrt{(x^T x)((x')^T x')}}$
- **Jaccard** $x, x' \in \{0,1\}^d$ $J(x, x') = \dfrac{\#(x \cap x')}{\#(x \cup x')} = \dfrac{x^T x'}{x^T x + (x')^T x' - x^T x'}$
- Note that if $x, x' \in \{0,1\}^d$ they can be seen as indicator functions for subsets of $1 : n$.
- Hence $x^T x' = \#(x \cap x')$ represents the cardinality of the intersection of sets given by x, x'
- All distances above are metrics.

Hash functions and hash codes

Let the data space be $\mathbb{R}^d$, and assume some fixed probability measure on this space.

- A **family of hash functions** is a set $\mathcal{H} = \{h : \mathbb{R}^d \to \{0, 1\}\}$ with the following properties
  1. For each h, $Pr[h(x) = 1] \approx \frac{1}{2}$
  2. The binary random variables defined by the functions in $\mathcal{H}$ are mutually independent. (Or, if $\mathcal{H}$ is not finite, a "not too large" random sample of such random variables is mutually independent.)
- Let $h_{1:k}$ be a mutually independent subset of $\mathcal{H}$. We call

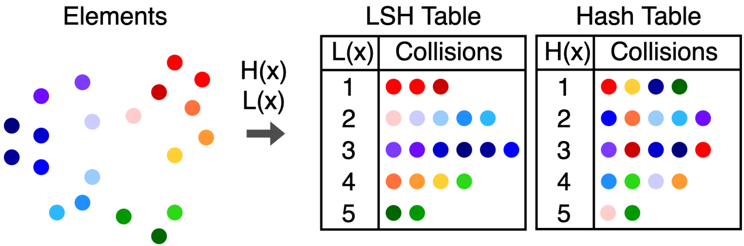
$$g(x) \;=\; [h_1(x)\, h_2(x)\, \ldots\, h_k(x)] \in \{0, 1\}^k \tag{1}$$

  the **hash code** of x.
- Note that the codes $g(x)$ are (approximately) uniformly distributed; the probability of any $g \in \{0, 1\}^k$ is about $\frac{1}{2^k}$.
- Useful hash functions must be fast to compute.

Hash tables

- A **hash table** $\mathcal{T}$ is a data structure in which points in $\mathbb{R}^d$ can be stored in such a way that
  1. All points with the same code g are in the same **bin** denoted by $\mathcal{T}_g$. The table need not use space for empty bins.
  2. Given any value $g \in \{0,1\}^k$, we can obtain a point in $\mathcal{T}_g$ or find if $\mathcal{T}_g = \emptyset$ in constant time (independent of the number of points n stored in $\mathcal{T}$).
     Some versions of hash tables return all points in $\mathcal{T}_g$, e.g., as a list, in constant time.
  3. It is usually assumed that storing a point x with given code $g(x)$ in a hash table is also constant time.
- Hence, using a hash table to store an x or to retrieve something, involves computing k hash functions, then a constant-time access to $\mathcal{T}$.
- When $x' \neq x$ and $g(x') = g(x)$ we call this a **collision**. In some applications (not of interest to us), collisions are to be avoided.

Hashing vs. Locality Sensitive Hashing (LSH)



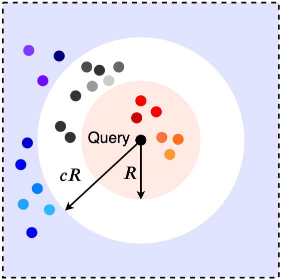
by Ben Coleman `randorithms.com`

Locality Sensitive Hash Functions and Codes

- A hash function h is **locality sensitive** iff for any $x, x' \in \mathbb{R}^d$

$$Pr[h(x) = h(x')] \geq p_1 \quad \text{when } ||x - x'|| \leq r \tag{2}$$
$$Pr[h(x) = h(x')] \leq p_2 \quad \text{when } ||x - x'|| \geq cr \tag{3}$$

  with p_1, p_2, r and $c > 1$ fixed parameters (of the family $\mathcal{H}$) and $p_1 > p_2$.
- W.l.o.g., we set $p_1 = p_2^{\rho}$ for some $\rho < 1$.



by Ben Coleman randorithms.com

LSH functions

- A locality sensitive h makes a weak distinction between points that are close in space vs. points that are far away. A hash code g from locality sensitive hash functions sharpens this distinction, in the sense that the probability of far away points colliding can be made arbitrarily small.

$$p_{bad} \;=\; Pr[g(x) = g(x')\,|\,||x - x'|| > cr] \;\leq\; p_2^k \tag{4}$$

- Assume x is not in $\mathcal{T}$; for any $x' \in \mathcal{D}$ which is far from x,the probability that x' collides with x is $\leq p_{bad}$.
- We construct $\mathcal{T}$ so that $p_{bad} \leq \frac{1}{n}$ for n the sample size. For this we need Exercise (in Homework 1)

$$k \;=\; \frac{\ln n}{-\ln p_2} \quad\Rightarrow\quad p_{bad} \leq \frac{1}{n} \tag{5}$$

- Suppose $x' \in \mathcal{T}$ is "close" to x. What is the probability that $g(x') = g(x)$?

$$p_{good} \;=\; p_1^k \;=\; p_2^{\rho k} \;=\; \frac{1}{n^\rho} \tag{6}$$

  This is the probability that the bin $\mathcal{T}_{g(x)}$ contains x'.

- h depends on the distance d
- h and g sometimes depend on r

How to find **good** hash functions?

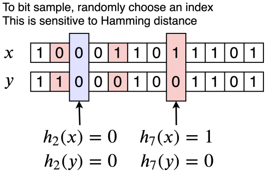
- We need large families of h functions
- that are easy to generate randomly
- and fast to compute for a given x

- Generic method to obtain them: random projections

LSH function for Hamming distance

- $\mathcal{H} = \{h_j = \text{bit}_j(x),\ j = 1 : d\}$
- a random $h \in \mathcal{H}$ samples a random bit of x
- Collision probability

$$p_1(x, x') \;=\; 1 - \frac{d_H(x, x')}{d} \tag{7}$$



by Ben Coleman

randorithms.com

LSH function for Euclidean and L1 distance

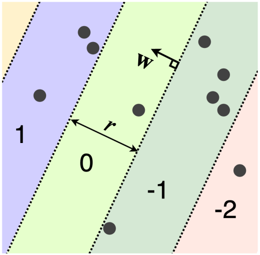
- project x on a random line, round to multiples of r

$$h_{w,b}(x) = \lfloor \frac{w^T x + b}{r} \rfloor \quad (8)$$

- If $w \sim Normal(0, I_d)$, hash function for Euclidean distance
- If $w \sim Cauchy(0,1)^d$, hash function for L1 distance
- Collision probability ($p = 2$ for Normal, $p = 1$ for Cauchy)

$$p_1(x, x') = \text{deterministic function of} \|x - x'\|_p \quad (9)$$

- Hash function space $\mathcal{H}_r$ is infinite, and depends on r



randorithms.com

by Ben Coleman

Analysis of projection on a random vector

- Data are $x \in \mathbb{R}^d$ as usual.
- Define $h_{w,b} : \mathbb{R}^d \to \mathbb{Z}$ by

$$h_{w,b}(x) = \lfloor \frac{w^T x + b}{r} \rfloor \tag{10}$$

  with $r > 0$ a width parameter, $w \in \mathbb{R}^d$, $b \in [0, r)$.
- Intuitively, x is "projected" on w[2], then the result is quantized into bins of width r, with a grid origin given by b.

- The family of hash functions is $\mathcal{H}_r = \{h_{w,b},\ w \in \mathbb{R}^d, b \in [0, r)\}$.
- Sampling $\mathcal{H}_r$: $w \sim Normal(0, I_d)$, $b \sim \text{uniform}[0, r)$.
  - Because the Normal distribution is a stable distribution, this ensures that $w^T x$ is distributed as $Normal(0, ||x||^2)$. Exercise Verify this
  - Hence $w^T x - w^T x'$ is distributed as $Normal(0, ||x - x'||^2)$. Exercise Verify this
  - Moreover, if hash functions are sampled independently from $\mathcal{H}_r$,(and nothing is known about x) then $h_{w,b}(x), h_{w',b'}(x)$ are independent random variables. Exercise Prove this

[2] w is not necessarily unit length

LSH function for angles

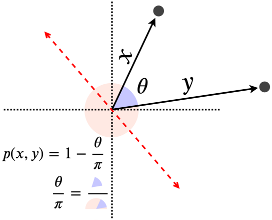
- project x on a random line, take the sign

$$h_{w,b}(x) = \text{sign}(w^T x) \qquad (11)$$

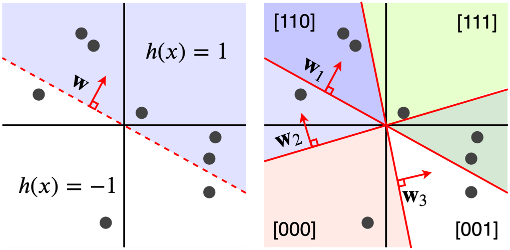
- Collision probability

$$p_1(x, x') \;=\; 1 - \frac{\theta(x, x')}{\pi} \qquad (12)$$

- Hash function space $\mathcal{H}$ is infinite



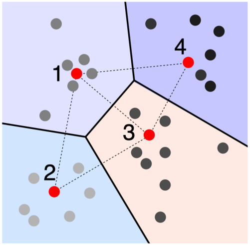
by Ben Coleman
randorithms.com



by Ben Coleman randorithms.com

Clustering LSH

- $\mathcal{H} = \{h = k(x)$, for some clustering of data$\}$
- h takes values in $1 : K$
- This is a data dependent hash function family
- Clustering can be K-means, min-diameter, hieararchical ...
- No theoretical guarantees, but works well in practice



by Ben Coleman

randorithms.com

Approximate r-neighbor retrival by LSH

Input $\mathcal{D}$ set of n points, L mutually independent hash codes $g_{1:L}$ of dimension k.

dexing Construct L hash tables $\mathcal{T}^{1:L}$, each storing $\mathcal{D}$.

trieval Given x

1. compute $g(x)$
2. for $j = 1, 2, \ldots L$
   if the bin $\mathcal{T}^j_{g(x)} \neq \emptyset$
   1. return some (all) x' from it.
   2. stop if a single neighbor is wanted.

Some analysis. We set $L = n^\rho$

- Indexing time $\propto kn^{\rho+1}$
- Retrieval time $\propto kn^\rho$
- Space used $\propto kn^{\rho+1}$

- For each $x' \in \mathcal{D}$ close to x, the probability that x' is NOT returned for any $j \in 1 : L$ is

$$(1 - \frac{1}{n^\rho})^{n^\rho} \approx \frac{1}{e} \tag{13}$$

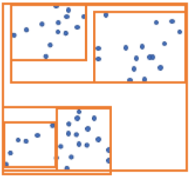
  This can be made arbitrarily small by multiplying L with a constant.
- For each $x' \in \mathcal{D}$ far from x, the probability that x' is NOT returned for any $j \in 1 : L$ is

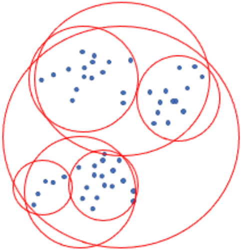
$$(1 - \frac{1}{N})^{n^\rho} \approx \left(\frac{1}{e}\right)^{1/n^{1-\rho}} \approx \frac{1}{e^0} = 1 \tag{14}$$

- Hence, we are almost sure not to return a far point, and have a significant probability to return a close point when one exists, if no points neither far nor close are in the data. This is

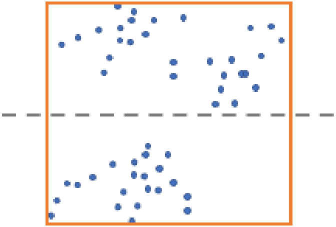
Heuristics for neighbors in high-dimensions

- typically a form of hierarchical clustering
- **K-D tree** for low dimensions (but observed to work well in high dimensions too)
- **Ball tree** for high dimensions

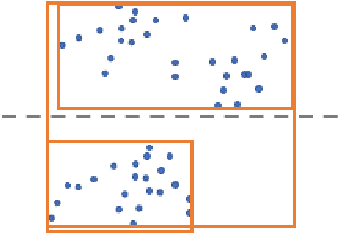




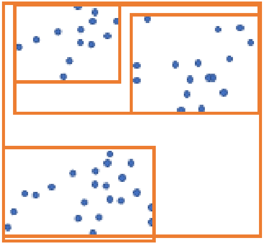
K-D Tree



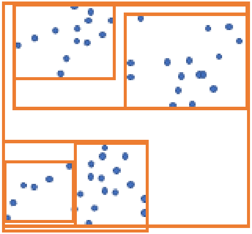
K-D Tree



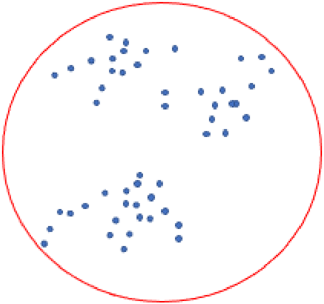
K-D Tree



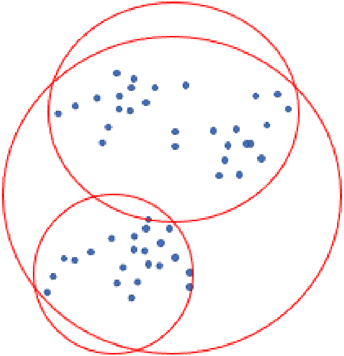
K-D Tree



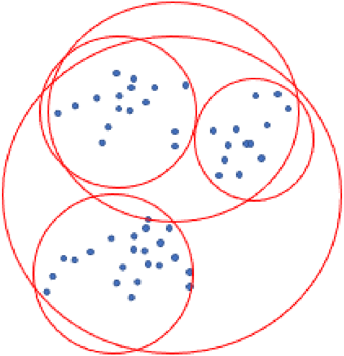
Ball Tree



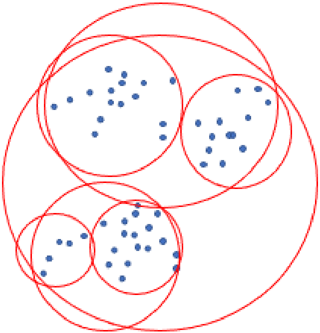
Ball Tree



Ball Tree



Ball Tree



K-D Tree construction

node k:

- $b^{\min}_{1:d}$, $b^{\max}_{1:d}$ min, max of box in each dimension
- $j_{\max}, \Delta_{\max} = \text{argmax}, \max_j \{b^{\max}_j - b^{\min}_j, j = 1 : d\}$ the largest dimension of the box
- $n_k, \bar{x}_k, \ldots$ number of points in node, mean, other statistics
- if k is **leaf** then $\mathcal{D}_k$ an array of the data under this node
- pointers p_k, l_k, r_k to parent and children nodes

Algorithm SPLIT-NODE(k)

It is assumed that k is **leaf**, hence $l_k, r_k =$**null**

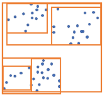
1. Create new leaf nodes l_k, r_k children of k and set k as their parent
2. Let $b^* = (b^{\max}_{j_{\max}} + b^{\min}_{j_{\max}})/2$
3. Create empty sets $\mathcal{D}_{l_k}, \mathcal{D}_{r_k}$
4. For $i = 1 : n_k$
    - if $x_{i,j_{\max}} < b^*$ then move x_i from $\mathcal{D}_k$ to $\mathcal{D}_{l_k}$; else move x_i to $\mathcal{D}_{r_k}$
    - update n_{l_k}, n_{r_k} and the other statistics as needed
    - update $b^{\max,\min}_{l_k, r_k}$
5. Update $\Delta_{l_k,\max}, j_{l_k,\max}$ and $\Delta_{r_k,\max}, j_{r_k,\max}$

Searching for r-neighbors with K-D Tree

- Denote by Node_k the d-dimensional box $[b_1^{\min}, b_1^{\max}] \times \ldots [b_d^{\min}, b_d^{\max}]$
- When is $B_r(x) \cap \text{Node}_k \neq \emptyset$?
  - x close to a corner: closest corner is $c = [\min\{|b_j^{\min} - x_j|, |b_j^{\max} - x_j|\}]_{j=1:d}$
  - x is interior or close to a face: $x_j \in [b_j^{\min}, b_j^{\max}]$ if $j \neq j_0$, and $x_j \in [b_j^{\min} - r, b_j^{\max} + r]$ for $j = j_0$
- When is $\text{Node}_k \subset B_r(x)$?
  - furthest corner is $c' = [\max\{|b_j^{\min} - x_j|, |b_j^{\max} - x_j|\}]_{j=1:d}$
  - if $\|x - c'\| \leq r$ then all $\text{Node}_k \subset B_r(x)$

Retrieving all points in $\mathcal{D} \cap B_r(x)$

- Recursively from the root, examine Node_k
- If $B_r(x) \cap \text{Node}_k = \emptyset$, return with no output
- Else
- If $\text{Node}_k \subset B_r(x)$ output all $\mathcal{D}_k$ and return
- Else examine children of k



Task: Finding Similar Documents

- **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

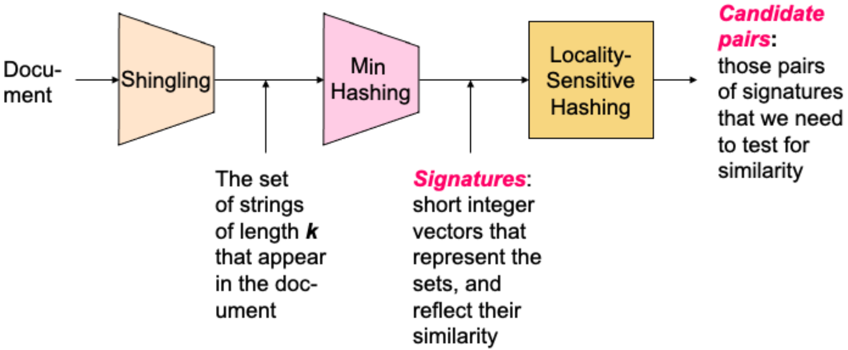
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 8

3 Essential Steps for Similar Docs

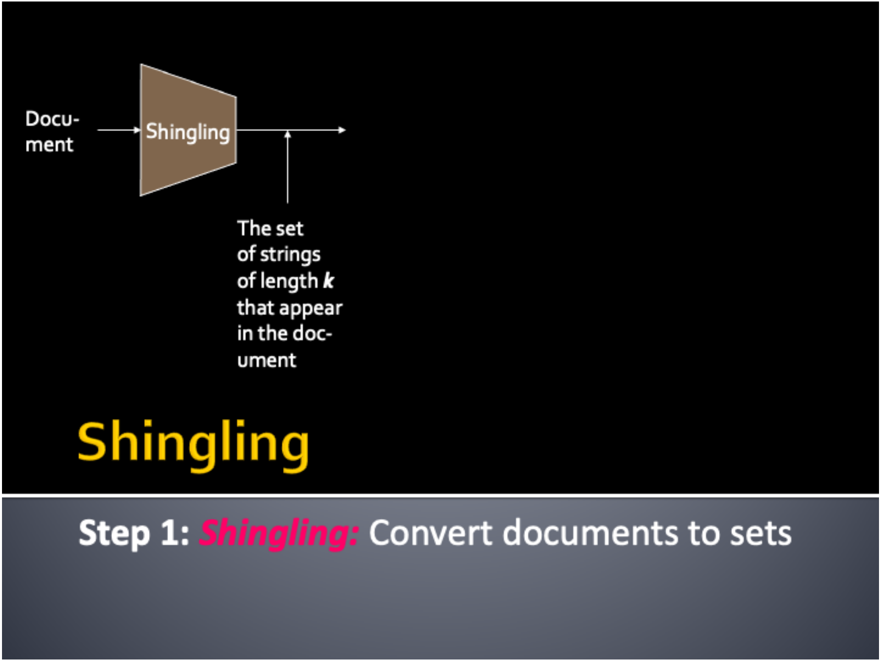
1. ***Shingling:*** Convert documents to sets
2. ***Min-Hashing:*** Convert large sets to short signatures, while preserving similarity
3. ***Locality-Sensitive Hashing:*** Focus on pairs of signatures likely to be from similar documents
   - **Candidate pairs!**

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 9

The Big Picture



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 10


Docu-
ment
Shingling
The set
of strings
of length *k*
that appear
in the doc-
ument
Shingling
Step 1: *Shingling:* Convert documents to sets

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!**

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 12

Define: Shingles

- A *k-shingle* (or *k-gram*) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example: k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ = {ab, bc, ca, ab}

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 13

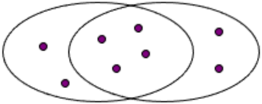
Compressing Shingles

- 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 $\mathbf{D_1}$= abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Similarity Metric for Shingles

- **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:**

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 15

Motivation for Minhash/LSH

- **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**
  - $N(N-1)/2 \approx 5*10^{11}$ 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…

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 17

Motivation for Minhash/LSH

- **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**
  - $N(N-1)/2 \approx 5*10^{11}$ 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…

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 17

Min-Hash – Motivation

- Denote $\mathcal{S} = \{$ space of k-shingles (k-grams) $\}$
- $|\mathcal{S}| = |\text{alphabet}|^k$ HUGE!
- document $\rightarrow c \in \{0,1\}^{|\mathcal{S}|}$ sparse!
- Similarity(document, document') $= J(c, c')$ Jaccard

$$J(c, c') = \frac{\#(c \cap c')}{\#(c \cup c')}$$

- **Wanted** compress $c \rightarrow x$, so that
  - $x \in \mathbb{Z}_+^L$ with $L \ll |\mathcal{S}|$
  - Jaccard is preserved (approximately), i.e.

$$J(c, c') \approx \frac{\#\{x_l = x'_l\}}{L} \quad (15)$$

  (fraction of equal elements in signatures approximates Jaccard)
  - x is called signature of c
- How? **Min-Hash**
- Why not random bit hashing?

Min-Hashing Example

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

1	5	1	5
2	3	1	3
6	4	6	4

2nd element of the permutation is the first to map to a 1

Permutation π / **Input matrix (Shingles x Documents)**

π (1)	π (2)	π (3)				
2	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	2	0	1	0	1
1	6	6	0	1	0	1
5	7	1	1	0	1	0
4	5	5	1	0	1	0

Signature matrix *M*

x_1			
2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 26

The Min-Hash Property

0	0
0	0
1	1
0	0
0	1
1	0

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **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 $y \in C_2$

    One of the two cols had to have 1 at position ***y***
  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 27

Min-Hash high-level summary

- Choose a family of hash functions $\mathcal{H} = \{h_\pi\}$
  - where π are permutatons of $\mathcal{S}$
  - $h_\pi(c) \in \{0, 1, \ldots |\mathcal{S}| - 1\}$
  - $h_\pi(c) =$ number 0's at the beginning of $\pi(c) =$ location of 1st 1 in $\pi(c)$ (zero-indexed)
- so that

$$Pr[h_\pi(c) = h_\pi(c')] \;=\; J(c, c') \quad \text{for all } \pi, c, c'$$ **(Min-Hash Property)**

- Choose L random permutations $\pi_{1:L}$
- Map c vectors to x by

$$x(c) \;=\; [h_{\pi_1}(c),\; h_{\pi_2}(c), \ldots h_{\pi_L}(c)]$$

- Approximate $J(c, c')$ by averaging

$$J(c, c') \;=\; \frac{1}{L} \sum_{l=1}^{L} 1_{[x_l = x'_l]}$$

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org 30

Finding similar documents: Summary

Input	Documents = lists of characters, length large, n large
Shingling	documents $\rightarrow$ binary vectors k-shingle space $\mathcal{S}$ large, c representation high-dimensional
Min-Hash	Binary vector $c \rightarrow$ signature x, $\dim(x) = L \ll \dim(c)$ preserves Jaccard similarity
LSH	on signatures x find neighbors in sub-quadratic time