CSE 547
STAT 548

4/25/25

Lecture 8

NN in high dimensions
LSH

Project · published
hw2

Lecture III Finding Nearest Neighbors in High Dimensions

& Big Data

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

CSE 547/STAT 548
Spring 2025

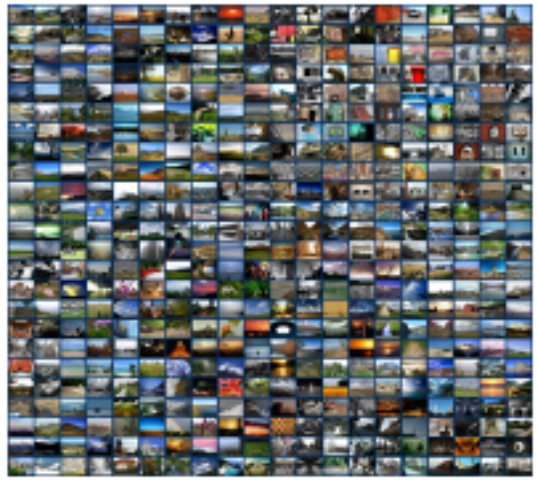
1. Motivation – finding similar items
2. Distance functions
3. Locality Sensitive Hashing
   - Hash functions and hash tables
   - What is Locality Sensitive Hashing
   - LSH functions from random projections
   - Approximate r-neighbor retrival by LSH
4. K-D trees, Ball trees etc.
5. Big data and the curse of dimensionality
6. Finding similar documents
   - Min-Hash

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



D database

Scene Completion Problem

[Hays and Efros, SIGGRAPH 2007]

query (image) = x

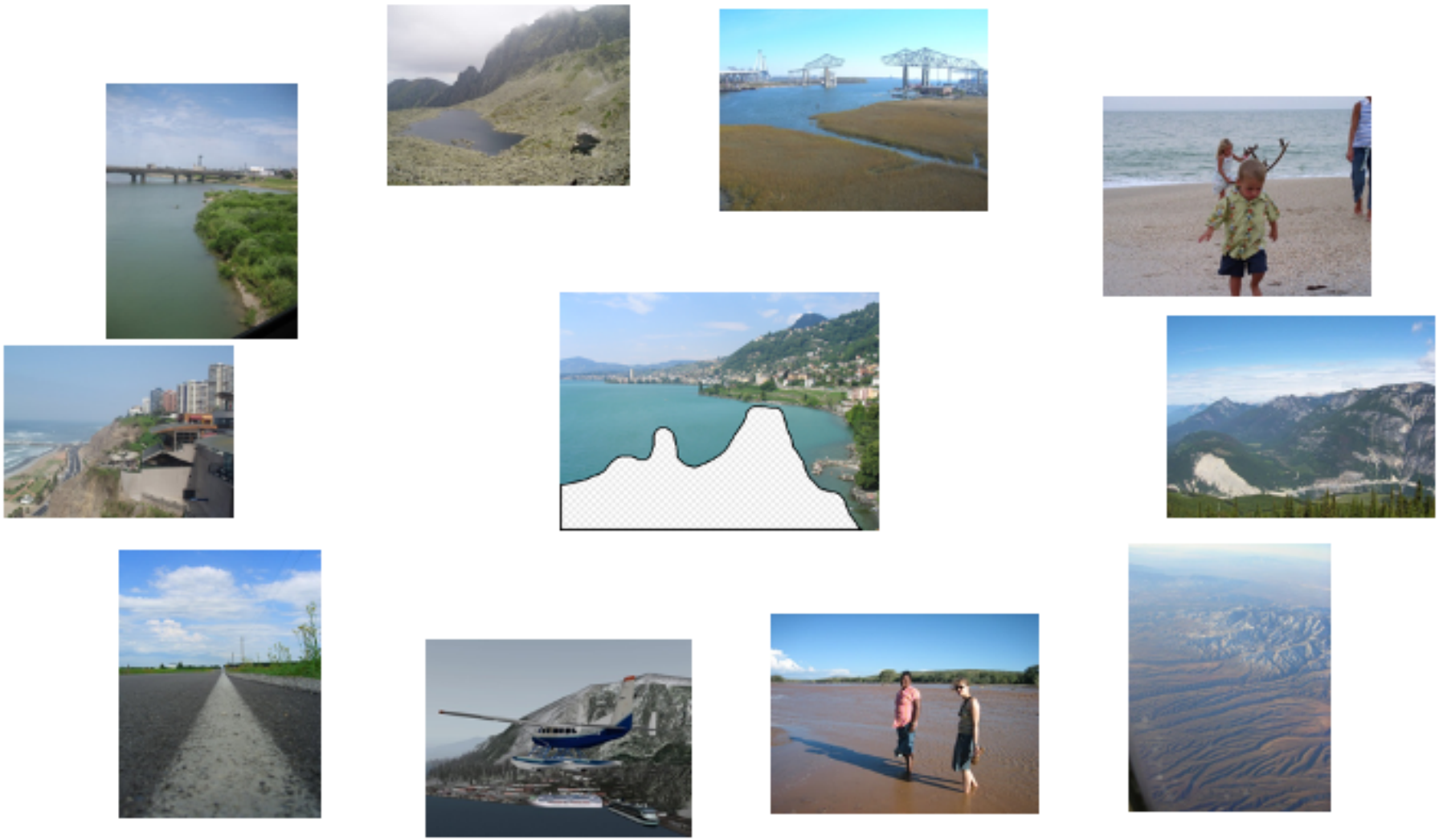
wanted $\{x'\}$ "neighbors" of x

in

$\mathcal{D}$

[Hays and Efros, SIGGRAPH 2007]

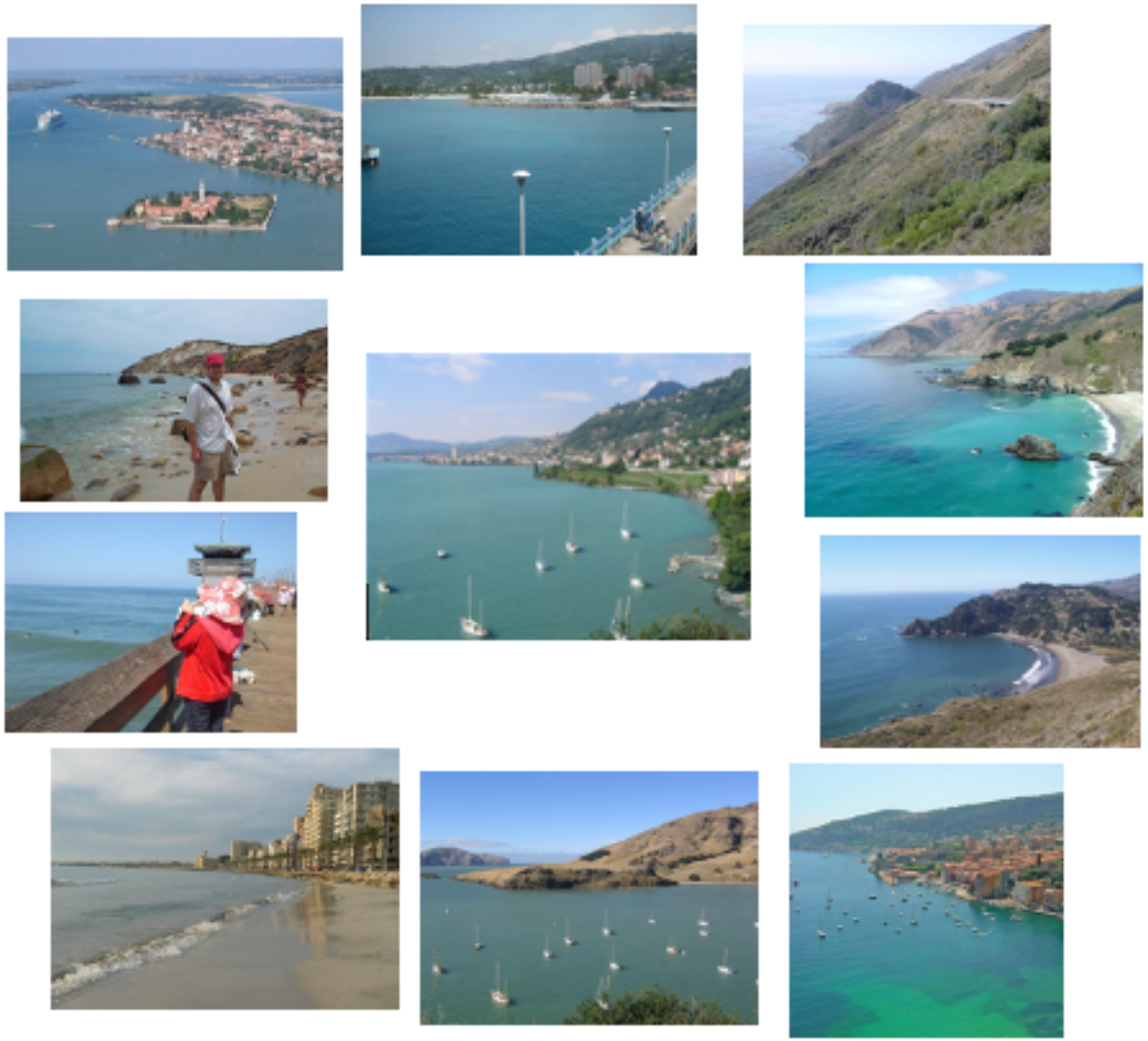
Scene Completion Problem



10 nearest neighbors from a collection of 20,000 images

[Hays and Efros, SIGGRAPH 2007]

Scene Completion Problem



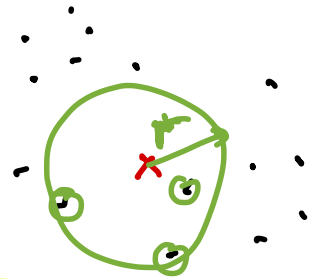
10 nearest neighbors from a collection of a million images

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)



Approx NN algs → index $\mathcal{D}$ ← distance

r usually fixed

$x \in \mathcal{D} \Rightarrow \text{neigh}(x) \ni x$

[1] with high probability

Distance and similarity functions

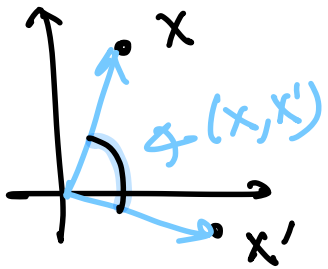
Distances

- **Euclidean** $x, x' \in \mathbb{R}^d$, $d_{Euclid}(x, x') = \|x - x'\| = \sqrt{x^T x + (x')^T x' - 2x^T x'}$
- **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

Locality Sensitive Hashing

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

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

  the **hash code** of x.

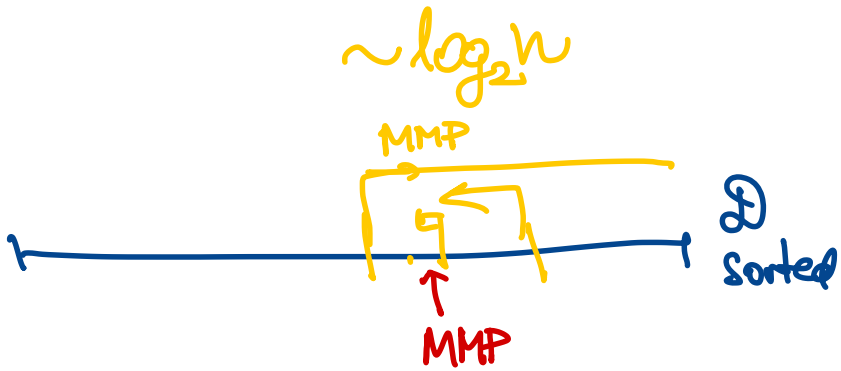
$h_j(x) \in \{0,1\}$

h_j = the j-th hash function

- 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.

Store large $\mathcal{D}$ $|\mathcal{D}| = n$ with constant access time

indep of n!



Hash Table

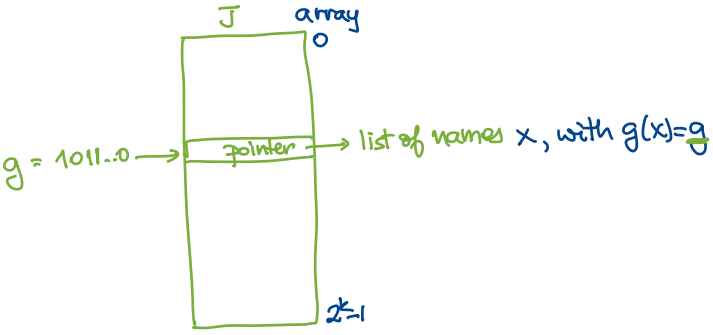
name x $\xrightarrow{\text{hashing}}$ $g(x)$

$g(x) \approx$ uniform $\{0,1\}^k$

choose k

$2^k \ll$ # all possible strings

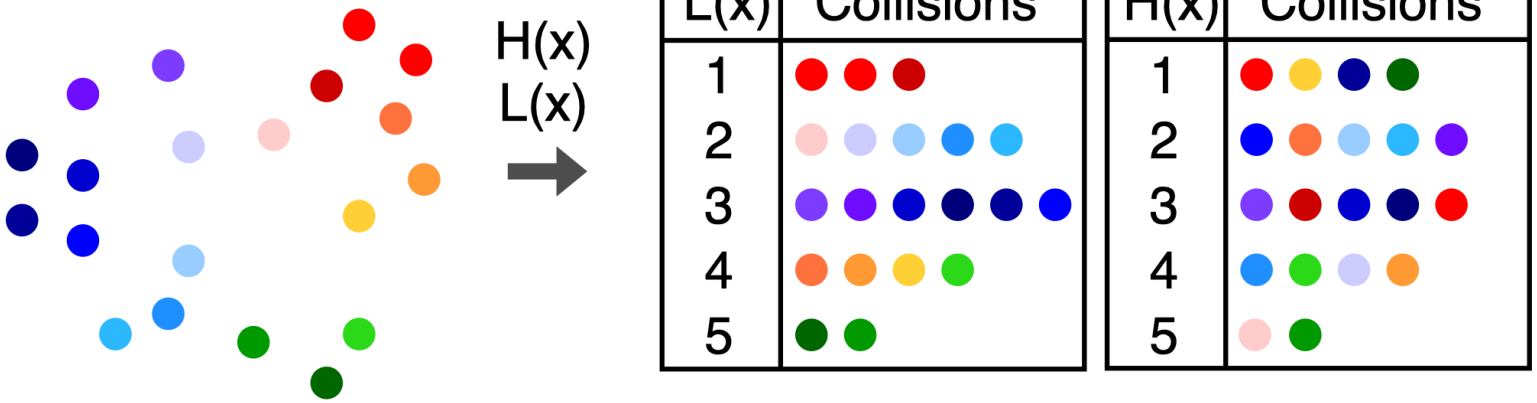
$>$ # all possible names



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

$\mathcal{H} = \{h, h, \ldots\}$

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

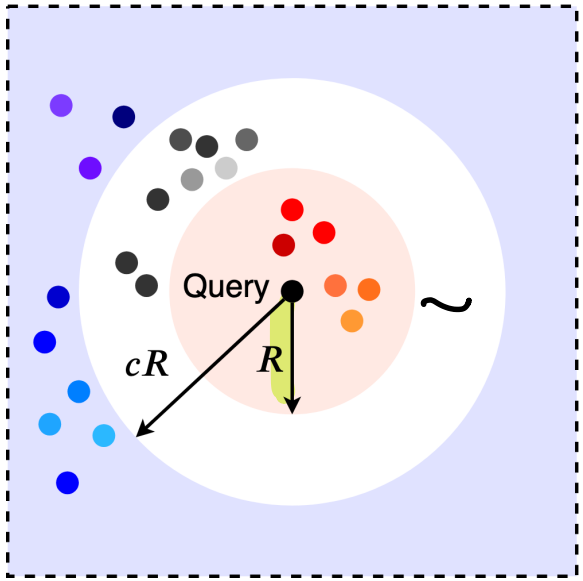
$$Pr[h(x) = h(x')] \geq p_1 \quad \text{when } ||x - x'|| \leq r \quad \text{(2)}$$

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

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 $\rho < 1$.

correct retrieval



by Ben Coleman randorithms.com

$$\frac{\text{Vol } B_R(\text{query})}{\text{Total Vol}(\mathcal{D})}$$

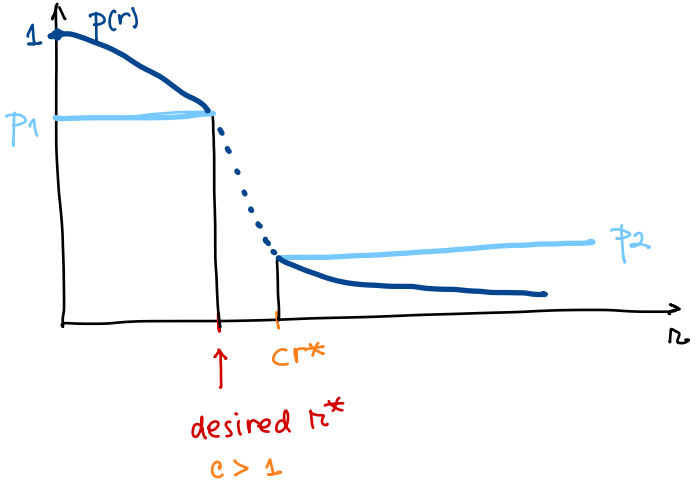
$p_1 = \frac{1}{2} = \frac{1}{8^{1/3}}$

$p_2 = \frac{1}{8}$

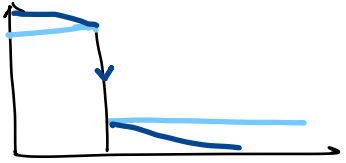
$\rho = \frac{1}{3}$

Intuition about (2), (3)

$$\Pr\left[h(x) = h(x'),\ \|x - x'\| = r \right] = p(r)$$



if $c = 1 \Rightarrow p(r)$ discontinuous !



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

$p_1 = Pr[h(x) = h(x')]$ $\quad \|x - x'\| < r$ good

$p_2 = Pr[\ \text{---}\ h\ \text{---}\ ]$ $\quad \|x - x'\| \geq cr$ $\quad c > 1$ bad

Note: $k \sim \log n \Rightarrow$ access time depends on n

$g \in \{0,1\}^k$ $\quad Pr[g(x) = g(x') \mid \|x - x'\| > cr] = p_2^k = p_{bad}$

Want $p_{bad} = \frac{1}{n} \rightarrow \frac{1}{n} = p_2^k \Rightarrow \boxed{k = \frac{\log n}{\log 1/p_2}}$

$p_{good} = Pr[g(x) = g(x') \mid \|x - x'\| < r] = p_1^k = p_2^{\rho k} = \left(\frac{1}{n}\right)^{\rho} = \boxed{\frac{1}{n^{\rho}}}$

How to find **good** hash functions?

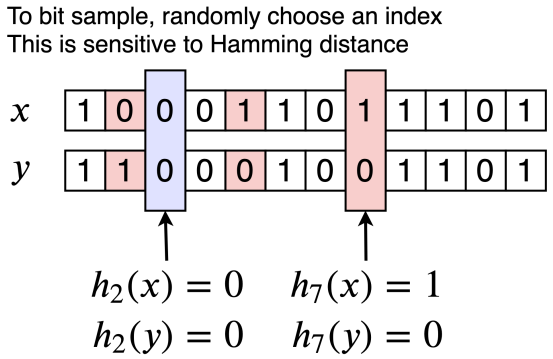
- We need large families of h functions
- that are easy to generate randomly
- and fast to compute for a given x

- 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

$$p_1(x, x') = 1 - \underbrace{\frac{d_H(x, x')}{d}}_{\Pr[h(x) \neq h(x')]} \quad (7)$$



by Ben Coleman

randorithms.com

$x \in \{0,1\}^d$

LSH function for Euclidean and L1 distance

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

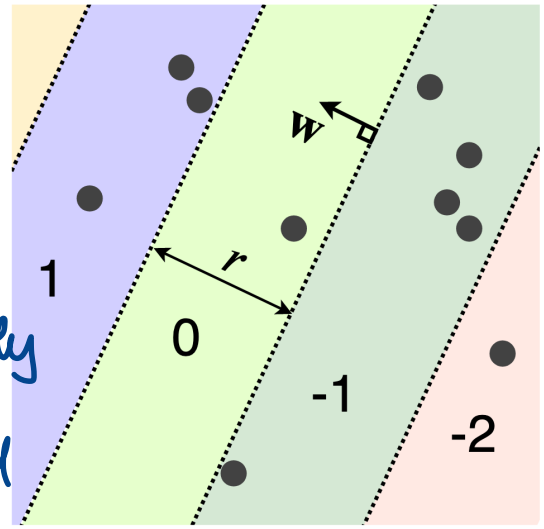
$$h_{w,b}(x) = \lfloor \frac{w^T x + b}{r} \rfloor \qquad (8)$$

b ~ unif [0, r)

- 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)

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

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



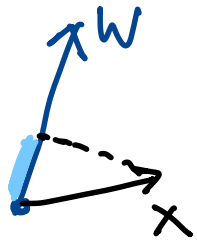
randorithms.com

by Ben Coleman

direction uniformly

distributed

simplify
r=1
||w||=1

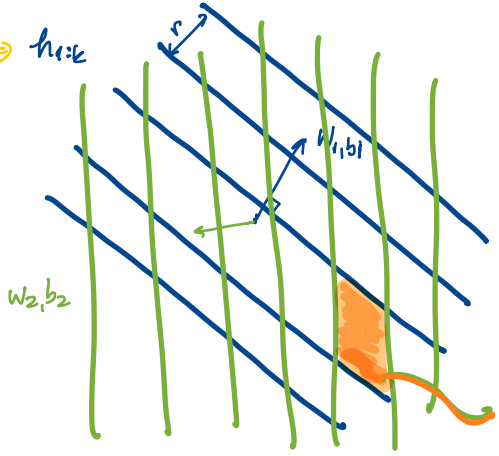


$w^T x$ if $\|w\|=1$

$h : \mathbb{R}^d \to \mathbb{Z}$

Sample $w_{1:k} \sim N(0, I_d)$
$b_{1:k} \sim \text{unif}[0, r)$ $\Rightarrow h_{1:k}$

$g(x) = [\, h_1(x) \quad \dots \quad h_k(x) \,]$



$g(x) = g(x')$

$g(x) = [\, h^{(x)}_{w_1,b_1}, h^{(x)}_{w_2,b_2} \,]$