Computational Geometry

3rd Part (c): High-Dimensional Nearest neighbors

Ioannis Emiris

Department of Informatics and Telecommunications
University of Athens

Spring 2015

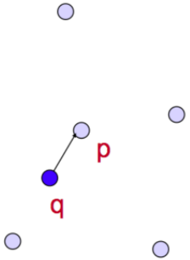
Contents

Outline

Introduction

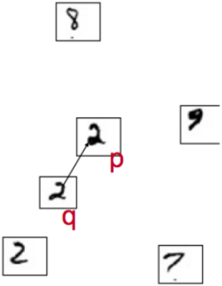
Given a distance function/metric:

- Preprocess: set of points/objects $P = \{p_1, \ldots, p_n\}$ in d dimensions.
- Query: Given a d-dimensional query point/object q, report the closest $p \in P$ to q.

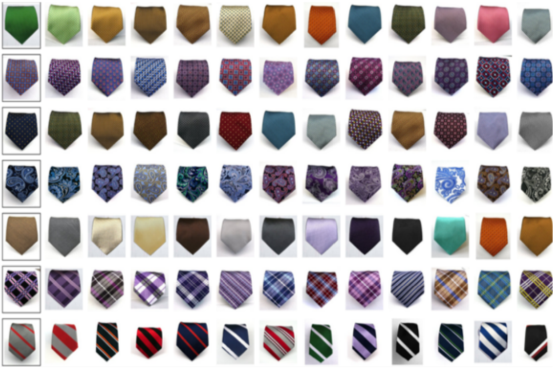


Motivation

- Points model general objects (e.g. handwritten digits)
- Distance between points inverse to similarity measure

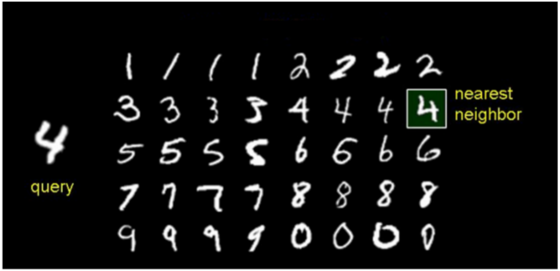


Wide spectrum of applications in many fields of Computer Science. Machine Learning



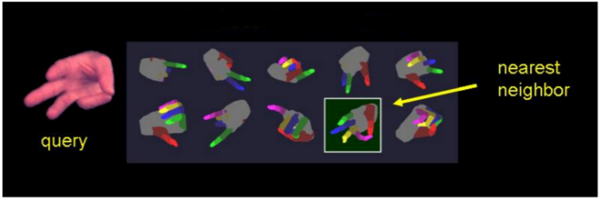
Applications

Wide spectrum of applications in many fields of Computer Science. Pattern Recognition and Classification



Applications

Wide spectrum of applications in many fields of Computer Science. Searching multimedia databases.



Nearest Neighbor

Exact NN

Given set P in d dimensions, and query point q, its NN is point $p_0 \in P$:

$$\text{dist}(p_0, q) \leq \text{dist}(p, q), \quad \forall p \in P.$$

Approximate NN

Given set P in d dimensions, approximation factor $1 > \epsilon > 0$, and query point q, an ϵ-NN, or ANN, is any point $p_0 \in P$:

$$\text{dist}(p_0, q) \leq (1 + \epsilon)\, \text{dist}(p, q), \quad \forall p \in P.$$

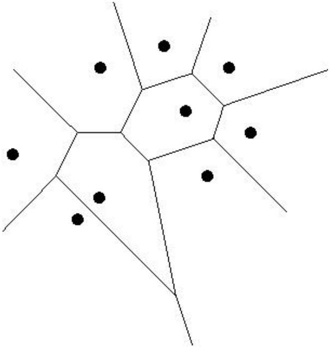
NN in $\mathbb{R}$

Sort/store the n points, use binary search for queries, then:

- Prepreprocessing in $O(n \log n)$ time
- Data structure requiring $O(n)$ space
- Answer the query in $O(\log n)$ time

NN in $\mathbb{R}^2$

- Preprocessing: Voronoi Diagram in $O(n \log n)$.
- Storage $= O(n)$.
- Given query q, find the cell it belongs (point location) in $O(\log n)$.
  NN = site of cell containing q.



Exact NN in $\mathbb{R}^d$

Is it faster than linear-time?

Curse of Dimensionality:

- Complexity of Voronoi diagram grows rapidly $= O(n^{\lceil d/2 \rceil})$.
- Planar point location methods do not extend to higher dimensions.
- The volume of the space increases so fast that data becomes sparse

State of the art:

- kd-trees: Sp $= O(n)$, Query $= O(d \cdot n^{1-1/d})$.
  Most practical for $d \ll \log n$: $O(\log n)$ expected for ``random" points
- Randomized (Clarkson'88): Sp $= O(n^{\lceil d/2 \rceil + \delta})$, Q $\simeq \log n \cdot \exp(d)$.
- n hyperplanes: point location $O(d^5 \log n)$, Sp $= O(n^{d+\delta})$ (Meiser'93)

Approximate NN in $\mathbb{R}^d$

- BBD tree (Arya,Mount et al.94,98) yield optimal query for $d = O(1)$. BBD
  In practice like kd-trees:
  - ANN software (Mount)
  - CGAL offers ``lazy" kd-trees
  - FLANN exploits structure by randomized kd-trees (Lowe-Muja)
- AVD achieve best asymptotic query-space tradeoff wrt n, for $d = O(1)$ (Arya,Mount et al.'09). AVD
  Improvement in non-extreme cases (Arya,Fonseca,Mount'11)
- Locality sensitive hashing (LSH) for ϵ-NN LSH
  Sp $\simeq n^{1/\epsilon^2}$, Q $\simeq d \log n$ (Indyk,Motwani'98)
  Sp $\simeq dn \log n$, Q $\simeq dn^{1/(1+\epsilon)}$ (Panigrahy'06) (Andoni,Indyk'06)

Lower bound on Approximate NN in $\mathbb{R}^d$ (Arya, Mount et al.)

Let $S(n)$, $Q(n)$ denote space and query time. Then, ignoring log factors, the space-time tradeoff is bounded as follows:

$$S(n)Q^2(n) = \Omega\left(\frac{n}{\epsilon^{d-1}}\right).$$

Outline

1. Introduction
   - Structure
2. Trees
   - kd-trees
   - Randomized kd-trees
   - Balanced Box-Decomposition trees
3. Approximate Voronoi Diagrams
   - Quadrees and representatives
   - Well-separated pair decomposition (WSPD)
4. Locality sensitive hashing
   - LSH functions
   - Specific metrics

Grid for Uniform points

- n uniformly distributed points in $[0, 1]^d$
- Cell structure (array) using $c = O(1)$ (Bentley-W-Yao'80):
  -- n/c cells/boxes, each of side $(c/n)^{1/d} < 1$.
  -- Box's volume $= c/n$, expected $\#$points per box $= c$.
  -- Expected query time $= O(1)$, for $d = O(1)$.
  -- But $\#$visited boxes $\leq 3^d - 1$.

Doubling dimension

- Consider general metric spaces, try to capture structure.
- Doubling (Assouad'83) dimension of pointset P is λ if 2^λ balls of radius $r/2$ centered at P are needed to cover any ball of radius r.
  E.g. $\mathbb{R}^d$ has doubling dimension $\Theta(d)$; models growth-limiting property.
- ϵ-NN: Sp $= O(n)$, Q $= O(\log n + 1/\epsilon^\lambda)$ (HarPeled-Mendel'06).
- Random Projection trees (Dasgupta-Freund,STOC) and randomly rotated kd-trees (Vempala) behave well for small doubling dimension

Dimension reduction

- Exact isometry (metric preservation) in $\leq n-1$ dimensions.
- Defn. Near isometry: Bi-Lipschitz embedding $f:(X,d)\rightarrow(Y,e)$ if

$$\exists C>0:\ \ C\cdot d(p,q)\leq e(f(p),f(q))\leq(1+\epsilon)C\cdot d(p,q),\quad\forall p,q\in X.$$

- Thm (Johnson-Lindenstrauss'84). $X\subset\mathbb{R}^d$ then $f:(X,L_2)\rightarrow(\mathbb{R}^k,L_2)$ is bi-Lipschitz for random projection s.t. $k=O(\log|X|/\epsilon^2)$ whp.
- Thm (Magen'02), (M.-Zouzias). $X\subset\mathbb{R}^d$ then $f:(X,L_2)\rightarrow(\mathbb{R}^k,L_2)$ preserves distance of points from t-dim affine hulls within ϵ, for $k=O(t\log|X|/\epsilon^2)$ whp.

ϵ-Nearest-Space

(Magen'02)

Given $X = \cup_{i=1}^{s} S_i$: dim $S_i = t = O(1)$, let $d' = \binom{d}{2} + d + 1$,

$$\xi : \mathbb{R}^d \to \mathbb{R}^{d'} : x \mapsto (1, x_1, \dots, x_d, x_1 x_2, \dots x_{d-1} x_d).$$

Given $x \in \mathbb{R}^d$, nearest S_i iff first $\xi(S_i)$ hit upwards from $\xi(x)$.
Exercise: Prove it! Does $\xi(x)$ also need $x_1^2, \dots, x_d^2$?

Reduction

Upward-hit implemented by (Meiser'93) in $\mathbb{R}^{d^2}$. Embedding using $t \cdot s$ points.
Hence X-query $= O(\log^{11} s/\epsilon^{20})$, space $\simeq s^{\log^2 s/\epsilon^4}$.

Query $\in \mathbb{R}^d$

$k = O(Q \log s/\epsilon^2)$ dimensions work w/prob $1/s^Q$ by extending (Magen,Thm.2).
Exercises: Extend Thm.2. Finish the analysis.

NN-preserving embedding

- Suppose $\lambda =$ doubling dim.
  Then, linear embedding to $\mathbb{R}^k$ is NN-preserving whp, with
$$k = O(\lambda \frac{\log(1/\epsilon)}{\epsilon^2}).$$
- So ϵ-NN: space $= O(n/\epsilon^k)$, query $= O(k \log(n/\epsilon))$.
- $X = \cup_{i=1}^{s} S_i : f(S_i)$ affine sets, then Nearest-Object-preserving f yields $k = O(\lambda \log s \cdot \log(1/\epsilon)/\epsilon^2)$ whp.

(Indyk-Naor'07)

Outline

Outline

kd-trees

- Different strategies to pick splitting coordinate.
- Leaves contain bucket of ≥ 1 points.

Complexity:

- Sp $= O(d \cdot n)$.
- construction of balanced tree: $O(d \cdot n \log n)$ by sorting per dimension, $O(n \log n)$ by linear-time median computation.
- insert/delete into balanced kd-tree $= O(\log n)$.
- NN $= O(d \cdot n^{1-1/d})$ at worst, but $O(\log n)$ expected, if $d = O(1)$, for several distributions (Bentley et al'77,Bentley'90).
  Topdown: $\log n$ to bucket, expected $O(1)$ neighbors, recurse to root

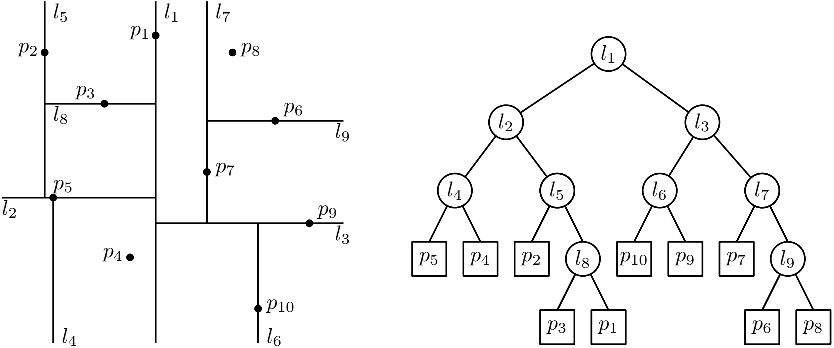
Topdown NN

Procedure NN($node$), given query q

if *node* is bucket (leaf) **then**
  Search all points in *node*, update current best
else {internal node}
  if cut-coor$(q) \leq$ node's cut-value **then**
    NN(left-child)
    if cut-coor$(q) +$ current best distance $>$ node's cut-value **then**
      NN(right-child)
    end if
  else {cut-coor$(q) >$ node's cut-value}
    NN(right-child)
    if cut-coor$(q) -$ current best distance $\leq$ node's cut-value **then**
      NN(left-child)
    end if
  end if(left/right)
end if(node)

Overall algorithm: NN(root).

Running example



Exercises

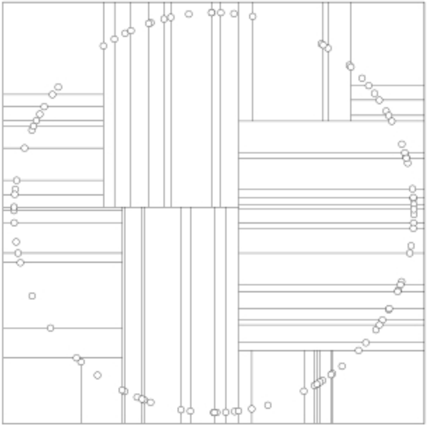
run algo

1. Run NN for a query point of your choice in the previous example dataset.
2. Find a query point in the previous example dataset for which the current best point is updated a max number of times: how large can this be?

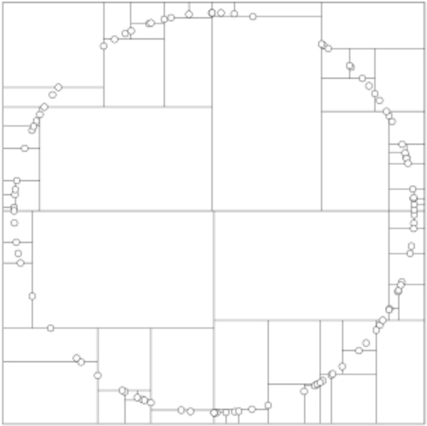
worst case

Find 7 points in $\mathbb{R}^2$ and a query that needs to check all nodes in the kd-tree for deterministic NN.

Splitting at max spread



median of set



closest to box centre

Extensions

kNN

- Store k current best points.
- Current ball encloses k current best points.
- Eliminate sibling if none of its points closer than any of k current best points.

Outline

Construction and query

Construct:

- Create r kd-trees s.t. searches are largely independent.
- Find $O(1)$ coord's maximizing variance: Pick random splitting coordinate. Hence small number of coordinates used for splitting.
- Principal Component Analysis finds moment axes: rotate to align them with the coordinte axes.

Search:

- Upper bound on total $\#$nodes to be searched.
- Single Priority queue stores candidates across r trees.
- Result similar to searching after projection to lower-dim space (Silpa-Anan,Hartley:CVPR08)
- Overall r independent projections to lower dimension so that NN among kNN, for small k, with high probability.

FLANN: Fast Library for ANN

(Lowe:IJCV04), software (Lowe,Muja)

Given the data: Automatic choice of algorithm and automatic configuration.

Algorithm Choices include:

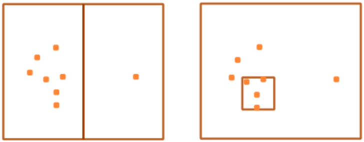
- Randomized kd-trees,
- Hierarchical k-means trees.

k-means unsupervised learning

- Given k clusters, each with centroid,
- Classify data points to nearest centroid,
- Calculate each cluster's barycenter: k new centroids
- Repeat until convergence

Outline

Box: set theoretic difference of two boxes, one enclosed in the other.



"Empirical runtimes for most distributions show little/no practical advantage over kd-trees" (Arya, Mount, Netanyahu et al.'94,98).

- Construct $= O(n \log n)$, $n = \#$points in the tree.
- Space $= O(dn)$.
- k ϵNNs in time $O(d(d/\epsilon)^d + k) \log n)$.
- Dynamic: point insertion/deletion $= O(\log n)$.

▸ overview

Outline

Outline

Approximate Voronoi Diagram (AVD)

Motivation

Reduce space for d-dimensional Voronoi diagram of N points from $\Theta(N^{\lceil d/2 \rceil})$ to almost linear.

Idea

Voronoi diagram only implicitly represented in AVD. Boundaries of Voronoi regions not explicitly stored.

Description

Partition underlying space using block decomposition (e.g. quadtree), and associate cell b to point $p \in S$, s.t. p is ϵ-NN for all $q \in b$.

Complexity

AVD (Har-Peled'01)

- construction time and storage in $O(\frac{N}{\epsilon^d} \log N \log \frac{N}{\epsilon}) \simeq O(\frac{N}{\epsilon^d} \log^2 N)$,
- ϵ-NN query in $O(\log \frac{N}{\epsilon}) \simeq O(\log N)$.

Tradeoffs (Arya, Mount et al.:J.ACM'09)

Take parameter $2 \leq \gamma \leq \frac{1}{\epsilon}$:

$$\text{Space} = O(N\gamma^{d-1} \log \frac{1}{\epsilon}) \text{ lies between } O(N \log \frac{1}{\epsilon}) \text{ and } O(\frac{N}{\epsilon^d} \log \frac{1}{\epsilon}).$$

$$\text{Query} = O(\log(N\gamma) + \frac{1}{(\epsilon\gamma)^{\frac{d-1}{2}}}) \text{ is between } O(\log N + \frac{1}{\epsilon^{\frac{d-1}{2}}}) \text{ and } O(\log \frac{N}{\epsilon}).$$

overview

Decomposition

- (PR) Quadtree recursively decomposes space into congruent blocks (subboxes inside boxes), until each block $= \emptyset$ or contains 1 point.
- Cell b represented by site $r_b \in S$, s.t. $r_b = \epsilon$-NN for every point in b.
- If b represented by > 1 sites, e.g. r_c also, then r_c is also ϵ-NN $\forall p \in b$
- Site r_b can be associated with different blocks in quadtree.
- Avoid multiple associations. Decomposition blocks are maximal: reduces likelihood of multiple associations, does not guarantee it.

(t, ϵ)-AVD

- Bucket capacity $t \geq 1$.
- Allow up to t elements $r_{ib} \in S$ be associated with each block b, where each point in b has one of the r_{ib} as ϵ-NN.
- If decomposition rule based on regular decomposition, then result analogous to bucket variant of (PR) Quadtree.
- Decomposition halts when number of different ϵ-NNs of points in b is $\leq t$.

Problem: Constructive definition of (t, ϵ)-AVD is circular: assumes we know NN.

Outline

Definitions

The WSPD is 1st phase in constructing AVD: poly(n), but costly in practice.

Definition

Subsets X_i, Y_i are well-separated if contained in min enclosing spheres S_x, S_y:

$$\text{distance of centers } \geq a \cdot \max\{\text{radius}(S_x), \text{radius}(S_y)\},$$

where a is the separation factor.

Definition

A WSPD of S is set $\{(X_1, Y_1), \ldots, (X_m, Y_m)\}$ of pairs of subsets $\subset S$ s.t.:

1. X_i and Y_i are well-separated with separation factor a, $i = 1, \ldots, m$.
2. For points $x \neq y \in S$, $\exists$ unique (X_i, Y_i):
   either $x \in X_i, y \in Y_i$, or $x \in Y_i, y \in X_i$.

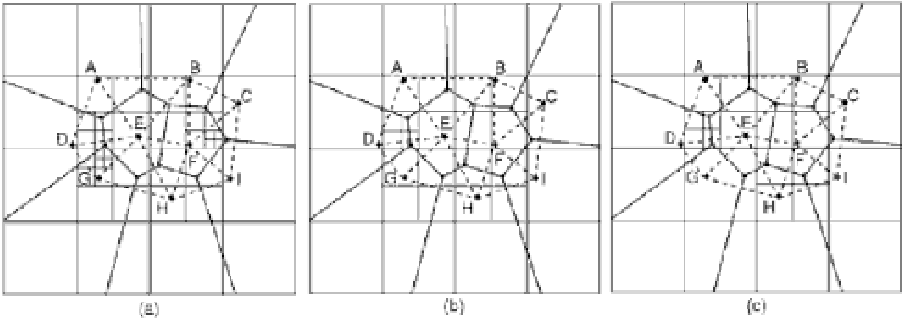
AVD from WSPD

- Construct quadtree for (t, ϵ)-AVD by digitizing bisector of well separated subsets $X, Y \subset S$.
- Apply process only to $O(n)$ well-separated pairs: #cells produced depends on ϵ and distance between pairs, not on n.
- Produce set of cells c_j: Each c_j associated with p_1 or p_2: associated point is an ϵ-NN from $\{p_1, p_2\}$.
- Consider NN search as tournament. Every pair p_1, p_2 competes to see who is closer to given point.
- Algorithm eliminates $n - 1$ points, remaining winning point is NN.
- By overlaying digitizations, all query points of cell share a common ϵ-NN.

Finalize AVD

- Overlay and merge digitizations in single quadtree decomposition.
- Store it as balanced box-decomposition (BBD) tree.
- Select representative for each cell by running algorithm for NN.
- Construction time $\simeq$ (# cells) $\times$ (NN query).

Example: 2d



Block decompositions in decreasing refinement, induced by (a)PM_1, (b)PM_2, (c)PM_3 quadtrees for AVD of A, B, C, D, E, F, G, H, I;
Voronoi diagram shown with broken lines.

Outline

LSH idea

LSH Family

We call a family H of hashing functions (r, c, P_1, P_2)-sensitive, $P_1 > P_2$ if, for any points $p \neq q \in \mathbb{R}^d$ and any randomly selected function $h \in_R H$:

- if $\|p - q\| \leq r$, then $\text{prob}_H[h(q) = h(p)] \geq P_1$,
- if $\|p - q\| \geq c$, then $\text{prob}_H[h(q) = h(p)] \leq P_2$.

LSH uses hashing functions (amplified) of the form

$$g(p) = (h_1(p), h_2(p), \ldots, h_k(p))$$

where the h_i are chosen at random from H (Indyk,Motwani'98)

Construction and search

Preprocess

- Select $L = n^{\rho}$ hashing functions $g_1, \ldots, g_L$.
- Construct L hashtables and hash all points to all tables.

Query

- Retrieve points from buckets $g_1(q), g_2(q), \ldots$ until:
  Either points from all L buckets are retrieved, or
  Total number of points retrieved is $> 2L$.
- Answer query based on retrieved points.

Outline

Projection based (Indyk et al.'04)

- Pick regular grid: Shift and rotate randomly.
- Hash function is $h : p \mapsto$ closest grid point.
- Gives $\rho = \log_{1/P_2} \frac{1}{P_1} \simeq 1/c$.

Near optimal (Andoni-Indyk'06)

- Project onto $\mathbb{R}^t$, constant t.
- Grid of balls: p can hit empty space: hash till ball is hit.
- Gives $\rho = 1/c^2 + O(\log t/\sqrt{t})$,
- Space $O^*(n^{1/(1+\epsilon)^2})$, query $O^*(n^{1+1/(1+\epsilon)^2})$

overview

Hash-table

LSH creates hash-table using (amplified) hash functions by concatenation:

$$g(p) = [h_1(p), h_2(p), \cdots, h_k(p)], \quad h_i \in_R H.$$

If the range of g is too large; to avoid empty buckets, we may combine the $h_i(p)$ into a new integer $\phi(p) < g(p)$; see, e.g., Euclidean space 1-dim hashing

Construction

Preprocess

- Having defined H and hash-function g:
- Select L hashing functions $g_1, \ldots, g_L$.
- Initialize L (sparse) hashtables, hash all points to all tables using g (or ϕ).

Large $k \Rightarrow$ larger gap between P_1, P_2. Small $P_1 \Rightarrow$ larer L so as to find neighbors. A practical choice is $L = 5$ (or 6).

Range (r, c)-Neighbor search

Input: r, c, query q i from 1 to L each item p in bucket $g_i(q)$ $\mathrm{d}(q, p) < cr$ **output** p

Decision problem: "**return** p" instead of "**output** p".

At end "**return** FAIL"; may also FAIL if many examined points.

NN search

Approximate NN

Input: query q Let $b \leftarrow$Null; $d_b \leftarrow \infty$ i from 1 to L each item p in bucket $g_i(q)$ large number of retrieved items (e.g. $> 3L$) b $\mathrm{d}(q, p) < d_b$ $b \leftarrow p$; $d_b \leftarrow \mathrm{d}(q, p)$ b

Theoretical bounds for $c(1+\epsilon)$-NN by reduction to $((1+\epsilon)^i, c)$-Neighbor decision problems, $i = 1, 2, \ldots, \log_{1+\epsilon} d$.

Known LSH-able metrics

- Hamming distance,
- L_2: Euclidean distance,
- L_1: Manhattan distance,
- L_k distance for any $k \in [0, 2)$,
- L_2 distance on a sphere,
- Cosine similarity,
- Jaccard coefficient.

$$\text{Recall} \quad \mathrm{dist}_{l_k}(x, y) = \sqrt[k]{\sum_{i=1}^{d} (x_i - y_i)^k}.$$

(Andoni-Indyk:J.ACM'08)

Outline

Hamming distance

Given strings x, y of length d, their Hamming distance $\mathrm{d}_H(x, y)$ is the number of positions at which x and y differ.

Example

Let $x = 10010$ and $y = 10100$. Then, $\mathrm{d}_H(x, y) = 2$.

Definition of hash functions

Recall idea. Given $x = (x_1, \ldots, x_d) \in \{0, 1\}^d$:

$$H = \{h_i(x) = x_i \ : \ i = 1, \ldots, d\}.$$

Obviously, $|H| = d$.
Pick uniformly at random $h \in_R H$: Then $\text{prob}[h(x) \neq h(y)] = \mathrm{d}_H(x, y)/d$,

$$\text{prob}[h(x) = h(y)] = 1 - \mathrm{d}_H(x, y)/d.$$

The family H is $(r_1, r_2, 1 - r_1/d, 1 - r_2/d)$-sensitive, for $r_1 < r_2$.

However probabilities $1 - r_1/d$, $1 - r_2/d$ can be close to each other.

Amplification

Given parameter k, define new family G by concatenation:

$$G = \{g : \{0,1\}^d \to \{0,1\}^k \mid g(x) = [h^1(x), \cdots, h^k(x)]\},$$

where $h^i \in_R H$.

-- We must have $L < |G| = d^k$, so as to pick L different g's.
-- The range of each g is $[0, 2^k)$, so $k < \lg n$.
-- May further use $\phi(\cdot)$ to avoid empty buckets; cf. Euclidean space 1-dim hashing.

Build Hash-tables

Build

Pick uniformly at random L functions $g_1, \ldots, g_L \in_R G$ (assuming $L < d^k$) i from 1 to L Initialize (one-dim) hash-table T_i, of size 2^k: for each $p \in P$, store p in bucket $g_i(p)$.

Complexity

Time to build: $O(Lnk)$ H-function calls.
Space: L hashtables and n pointers to strings per table $= O(Ln)$ pointers.
Also store n strings $= O(dn)$ bits.
(r, c)-Neighbors: Query $= O(L(k + d))$, assuming $O(1)$ strings per bucket.

Euclidean Space

$$\text{Recall:} \quad \text{dist}_{l_2}(x, y)^2 = \sum_{i=1}^{d} (x_i - y_i)^2.$$

Let point $p \in \mathbb{R}^d$, and d-vector $v \sim \mathcal{N}(0, 1)^d$ have coordinates identically independently distributed (i.i.d.) by the standard normal. Set $w \in \mathbb{N}^*$, pick t uniformly $\in_R [0, w)$. Define:

$$h(p) = \lfloor \frac{p \cdot v + t}{w} \rfloor \in \mathbb{Z};$$

essentially: project p on the line of v, shift by t, partition into cells of length w. The optimal value for w depends on P and q. In general, $w = 4$ is good. Also $k = 4$ (or 5), and L is 5 (or 6).

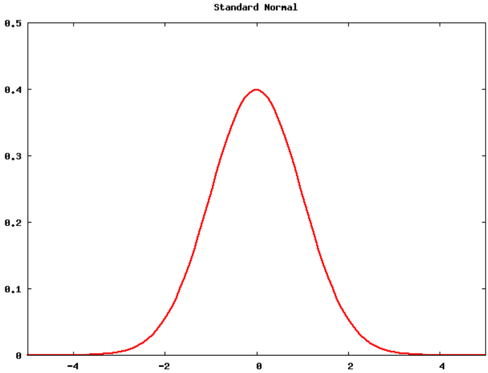
Normal distribution

Vector $v \sim \mathcal{N}(0,1)^d$ has coordinates distributed according to the standard normal (Gaussian) distribution:

$$v_i \sim \mathcal{N}(0,1), \quad i = 1, 2, \ldots, d:$$

with mean $\mu = 0$, variance $\sigma^2 = 1$ (σ is the standard deviation).

The bell curve:



Normal from Uniform

Given uniform U generator (Wikipedia):

- Marsaglia: Use independent uniform $U, V \in_R (-1, 1)$, $S = U^2 + V^2$. If $S \geq 1$ then start over, otherwise

$$X = U\sqrt{\frac{-2\ln S}{S}}, \qquad Y = V\sqrt{\frac{-2\ln S}{S}}$$

  are independent and standard normally distributed.

Hash-table

We may build a *k-dimensional* hash-table with indexing function:

$$g(p) = [h_1(p), h_2(p), \ldots, h_k(p)].$$

Many buckets shall be empty. Hence build 1-dim hash-table with classic index:

1-dimensional hash-function

$$\phi(p) = (r_1 h_1(p) + r_2 h_2(p) + \cdots + r_k h_k(p) \bmod M) \bmod \text{TableSize},$$

where `int` $r_i \in_R \mathbb{Z}$, prime $M = 2^{32} - 5$ if $h_i(p)$ are `int`, TableSize$= n/2$ (or n).

Recall $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$.

Hashing trick

Remember object IDs so as not to search entire bucket.

Object ID

$$\mathrm{ID}(p) = r_1 h_1(p) + r_2 h_2(p) + \cdots + r_k h_k(p) \bmod M$$

is locality sensitive: depends on w-length cells on the v-lines.
Then indexing hash-function is $\phi(p) = \mathrm{ID}(p) \bmod \mathrm{TableSize}$.

Store ID along with pointer to object.
Search follows pointers only for p: $\mathrm{ID}(p) = \mathrm{ID}(q)$.
Can have smaller TableSize$= n/8$ or $n/16$ (heuristic choice).

Hash-function

$$\text{Recall} \quad \text{dist}_{l_1}(x, y) = \sum_{i=1}^{d} |x_i - y_i|.$$

Consider $\mathbb{R}^d$, r is the radius of the range search.
Pick reals: $w \gg r$, uniformly distributed $s_i \in_R [0, w)$, $i = 0, 1, \ldots, d-1$.
Construct d-dimensional hashtable, corresponding to grid shifted by the s_i's, where every cell is a bucket; the cell size is determined by w.

Locality sensitive function

$$\text{Let } a_i = \lfloor \frac{x_i - s_i}{w} \rfloor \in \mathbb{Z} \quad i = 0, 1, \ldots, d-1, \text{ then:}$$

$h(x) = a_{d-1} + m \cdot a_{d-2} + \cdots + m^{d-1} \cdot a_0$, $m > \max_i a_i$.
By concatenation, hash-function

$$g(x) = [h_1(x), h_2(x), \cdots, h_k(x)].$$

LSH for Cosine distance / similarity

Consider $\mathbb{R}^d$, equipped with cosine similarity of two vectors:

$$\cos(x, y) = (x \cdot y)/(\|x\| \cdot \|y\|),$$

which expresses the angle between vectors x, y.

For comparing documents or, generally, long vectors based on direction, not length.

Shall be approximated by random projections (next slide).

Random projection

Let $r_i \sim \mathcal{N}(0,1)^d$. Define $h_i(x) = \begin{cases} 1, & \text{if } r_i \cdot x \geq 0 \\ 0, & \text{if } r_i \cdot x < 0 \end{cases}$.

Then $F = \{h_i(x) \mid \text{ for every } r_i \sim \mathcal{N}(0,1)^d\}$ is a locality sensitive family.

Intuition: Each r_i is normal to a hyperplane. If two vectors lie on the same side of many random hyperplanes, then very likely they are similar (Andoni-Indyk'08).

Lemma

Two vectors match with probability proportional to their cosine.

(Amplification) Given parameter k, define new family $G(F)$ by concatenation:

$$G(F) = \{g : \mathbb{R}^d \to \{0,1\}^k \mid g(x) = [h_1(x), h_2(x), \cdots, h_k(x)]\}.$$