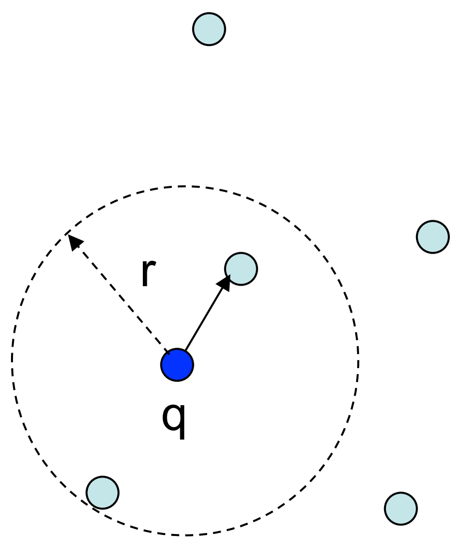
Similarity Search in High Dimensions

Piotr Indyk
MIT

Definitions

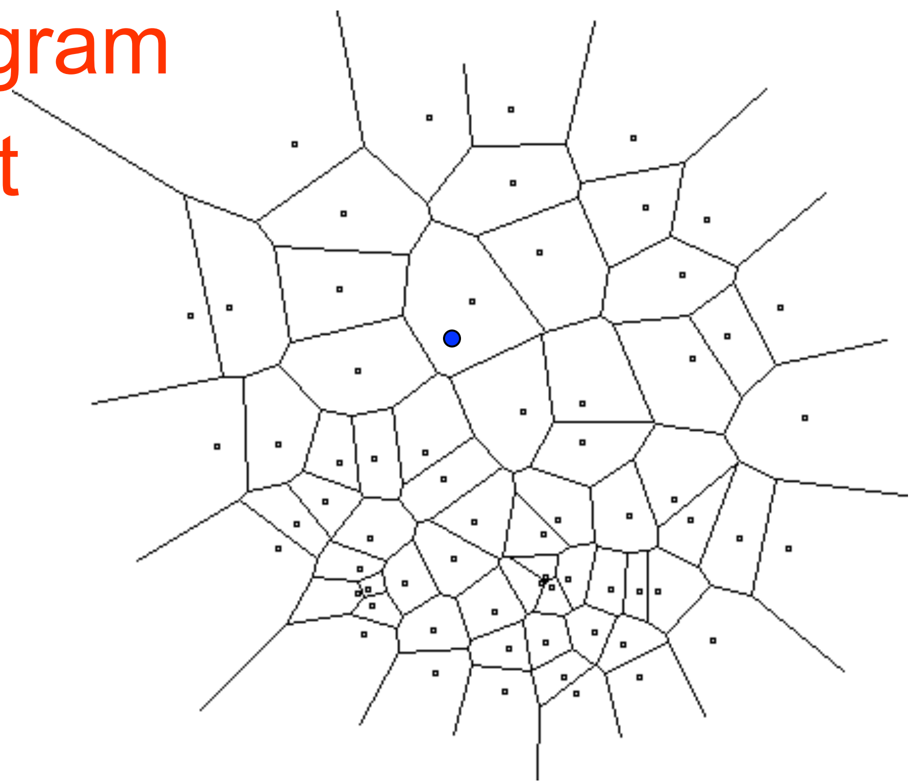
- Given: a set P of n points in R^d
- Nearest Neighbor: for any query q, returns a point $p \in P$ minimizing $||p-q||$
- r-Near Neighbor: for any query q, returns a point $p \in P$ s.t.

  $||p-q|| \leq r$ (if it exists)



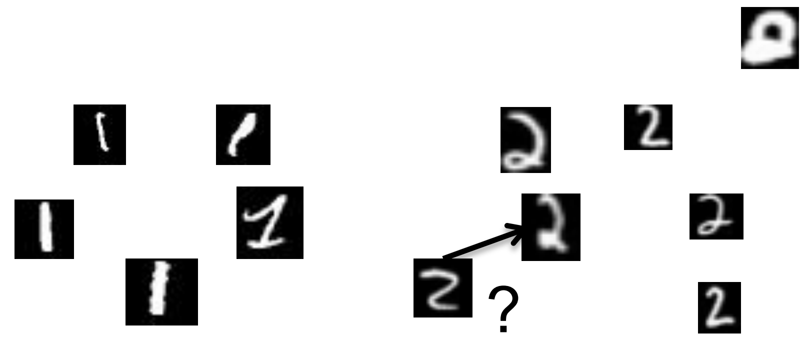
The case of d=2

- Compute Voronoi diagram
- Given q, perform point location
- Performance:
  - Space: O(n)
  - Query time: O(log n)



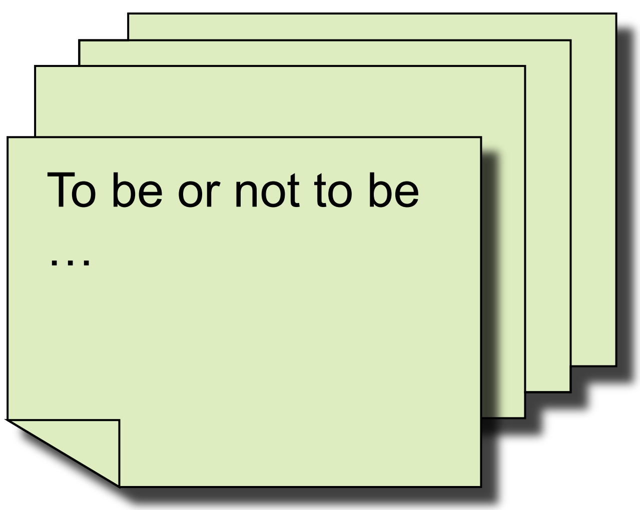
High-dimensional near(est) neighbor: applications

- Machine learning: nearest neighbor rule
  - Find the closest example with known class
  - Copy the class label



Dimension=number of pixels

- Near-duplicate Retrieval



(… , 1, …, 4, … , 2 , …, 2, …)

(… , 6, …, 1, … , 3 , …, 6, …)

(… , 1, …, 3, … , 7 , …, 5, …)

(… , 2, …, 2, … , 1 , …, 1, …)

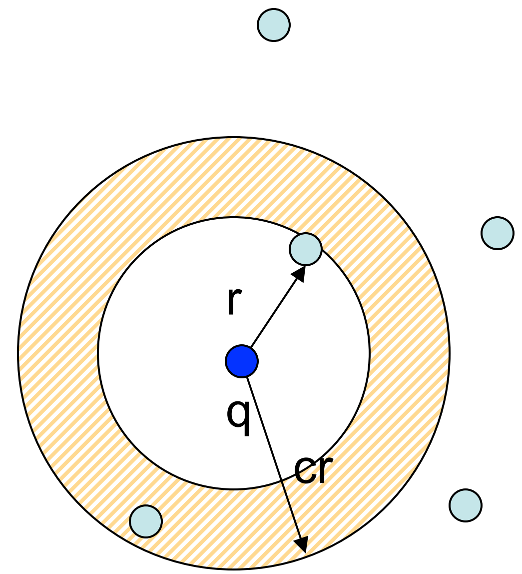
Dimension=number of words

The case of d>2

- Voronoi diagram has size $n^{[d/2]}$
  - [Dobkin-Lipton'78]: $n^{2^{(d+1)}}$ space, $f(d) \log n$
  - [Clarkson'88]: $n^{[d/2](1+\varepsilon)}$ space, $f(d) \log n$ time
  - [Meiser'93]: $n^{O(d)}$ space, $(d+\log n)^{O(1)}$ time
- We can also perform a linear scan: $O(dn)$ time
- Or parametrize by intrinsic dimension
- In practice:
  - kd-trees work "well" in "low-medium" dimensions

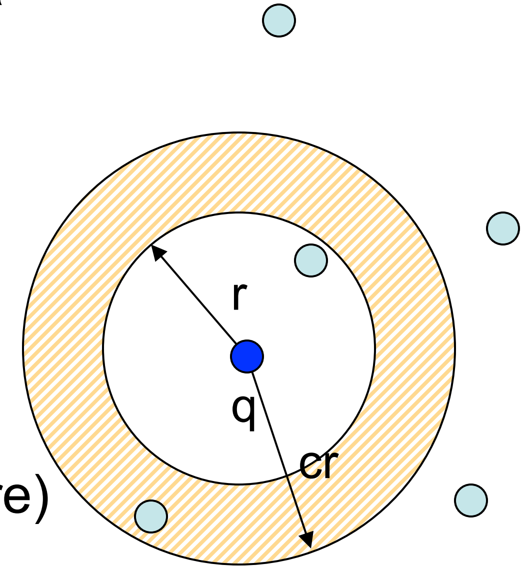
Approximate Nearest Neighbor

- c-Approximate Nearest Neighbor: build data structure which, for any query q
  - returns $p' \in P$, $\|p-q\| \leq cr$,
  - where r is the distance to the nearest neighbor of q



Approximate Near Neighbor

- c-Approximate r-Near Neighbor: build data structure which, for any query q:
  - If there is a point $p \in P$, $||p-q|| \leq r$
  - it returns $p' \in P$, $||p-q|| \leq cr$



- Most algorithms randomized:
  - For each query q, the probability (over the randomness used to construct the data structure) is at least 90%
- Reductions and variants:
  - c-Approx Nearest Neighbor reduces to c-Approx Near Neighbor (Wednesday)
  - One can enumerate all approx near neighbors

    → solving exact near neighbor via filtering
  - Other apps: c-approximate Minimum Spanning Tree, clustering, etc.

Approximate algorithms

- Space/time exponential in d [Arya-Mount'93], [Clarkson'94], [Arya-Mount-Netanyahu-Silverman-Wu'98] [Kleinberg'97], [Har-Peled'02], ….
- Space/time polynomial in d [Indyk-Motwani'98], [Kushilevitz-Ostrovsky-Rabani'98], [Indyk'98], [Gionis-Indyk-Motwani'99], [Charikar'02], [Datar-Immorlica-Indyk-Mirrokni'04], [Chakrabarti-Regev'04], [Panigrahy'06], [Ailon-Chazelle'06]…

Space	Time	Comment	Norm	Ref
$dn+n^{O(1/\varepsilon^2)}$	$d * \log n / \varepsilon^2$ (or 1)	$c=1+\varepsilon$	Hamm, l_2	[KOR'98, IM'98]
$n^{\Omega(1/\varepsilon^2)}$	$O(1)$			[AIP'06]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c$	Hamm, l_2	[IM'98], [GIM'98],[Cha'02]
		$\rho(c)<1/c$	l_2	[DIIM'04]
$dn * \log s$	$dn^{\sigma(c)}$	$\sigma(c)=O(\log c/c)$	Hamm, l_2	[Ind'01]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c^2 + o(1)$	l_2	[AI'06]
		$\sigma(c)=O(1/c)$	l_2	[Pan'06]

$n^{O(1/\varepsilon^2)}$ space, $d * \log n / \varepsilon^2$ query time,
Hamming distance

Hamming distance sketches

[Kushilevitz-Ostrovsky-Rabani'98]

- Let x,y in $\{0,1\}^d$, $r>1$, $\varepsilon>0$, $0<\delta<1$
- Want: $sk: \{0,1\}^d \rightarrow \{0,1\}^t$ such that

  given $sk(x)$, $sk(y)$:
  - If $H(x,y) > (1+\varepsilon)r$, we report YES
  - If $H(x,y) < (1-\varepsilon)r$, we report NO

  with probability $>1-\delta$
- In fact, we test if $H(sk(x),sk(y))>R$ for some R
- How low t can we get ?
- Will see $t=O(\log(1/\delta)/\varepsilon^2)$ suffices

Sketch

- Setup:
  - Choose a random set S of coordinates
    - For each i, we have $Pr[i \in S]=1/r$
  - Choose a random vector u in $\{0,1\}^d$
- Sketch: $Sum_S(x) = \Sigma_{i \in S}\, x_i\, u_i \bmod 2$
- Estimation algorithm:
  - $B = Sum_S(x) + Sum_S(y) \bmod 2$
  - YES, if $B=1$
  - NO, if $B=0$
- Analysis:
  - We have $B=Sum_S(z)$ where $z=x$ XOR y
  - Let $D=||z||_0$
  - $$\begin{aligned} Pr[B=1] &= ½ * Pr[z_S \neq 0] \\ &= ½ * [1-Pr[z_S=0]] \\ &= ½ * [1-(1-1/r)^D\,] \end{aligned}$$
  - For r large enough: $(1-1/r)^D \approx e^{-D/r}$, so
    - If $D> (1+\varepsilon)r$, then $e^{-(1+\varepsilon)} < 1/e - \varepsilon/3$ and $Pr>1/2(1-1/e + \varepsilon/3)$
    - If $D< (1-\varepsilon)r$, then $e^{-(1-\varepsilon)} > 1/e + \varepsilon/3$ and $Pr<1/2(1-1/e - \varepsilon/3)$
  - Using $O(\log(1/\delta)/\varepsilon^2)$ sums does the job (Chernoff bound)

Sketch is good

- Data structure (for P, $r>1$, $\varepsilon>0$)
  - Compute $sk: \{0,1\}^d \to \{0,1\}^t$, $t=O(\log(1/\delta)/\varepsilon^2)$ for $\delta=1/n^{O(1)}$
    - Sketch works (with high probability) for fixed query q and all points p in P
  - Exhaustive storage trick:
    - Compute

      $S=\{u \text{ in } \{0,1\}^t: H(u,p)>R$ for some p in $P\}$
    - Store S (space: $2^t=n^{O(1/\varepsilon^2)}$)
- Query: check whether $sk(q)$ in S

Beyond $\{0,1\}^d$: l_1 norm

- l_1 norm over $\{0\ldots M\}^d$
  - Embed into Hamming space with dimension dM [Linial-London-Rabinovich'94]
    - Compute

      $$\text{Unary}((x_1, \ldots, x_d)) = \text{Unary}(x_1) \ldots \text{Unary}(x_d)$$

    - We have

      $$\|p-q\|_1 = H(\text{ Unary}(p), \text{Unary}(q) \text{ })$$

  - Need to deal with large values of M
- l_1 norm over $[0\ldots s]^d$
  - Round each coordinate to the nearest multiple of $r\, \varepsilon/d$
    - Introduces additive error of $r\, \varepsilon$, or multiplicative $(1+\varepsilon)$ factor
  - Now we have $M=s^*\, d/(r\, \varepsilon)$

Beyond $\{0,1\}^d$: l_1 norm ctd

- l_1 norm over R^d
  - Partition R^d using a randomly shifted grid of side length $s=10r$ [Bern'93]
  - For any two points p and q, the probability that p and q fall into different grid cells is at most

$$|p_1-q_1|/s + |p_2-q_2|/s+..+|p_d-q_d|/s= ||p-q||_1 /s$$

    - If $||p-q||_1 \leq r$, then probability is at most 10%
  - Build a separate data structure for each grid cell
  - To answer a query q, use the data structure for the cell containing q

Beyond $\{0,1\}^d$: l_2 norm

- Embed l_2^d into l_1^t with $t=O(d/\varepsilon^2)$ with distortion $1+\varepsilon$ [Figiel-Lindenstrauss-Milman'76]
  - Use random projections
- Or, use Johnson-Lindenstrauss lemma to reduce the dimension to $t=O(\log n/\varepsilon^2)$ and apply exhaustive storage trick directly in l_2^t [Indyk-Motwani'98]

Next two lectures

- Wednesday: reducing nearest to near neighbor
- Thursday: other algorithms for near neighbor (less space, more query time)
  - Locality Sensitive Hashing