

Hashing, sketching, and other approximate algorithms for high-dimensional data

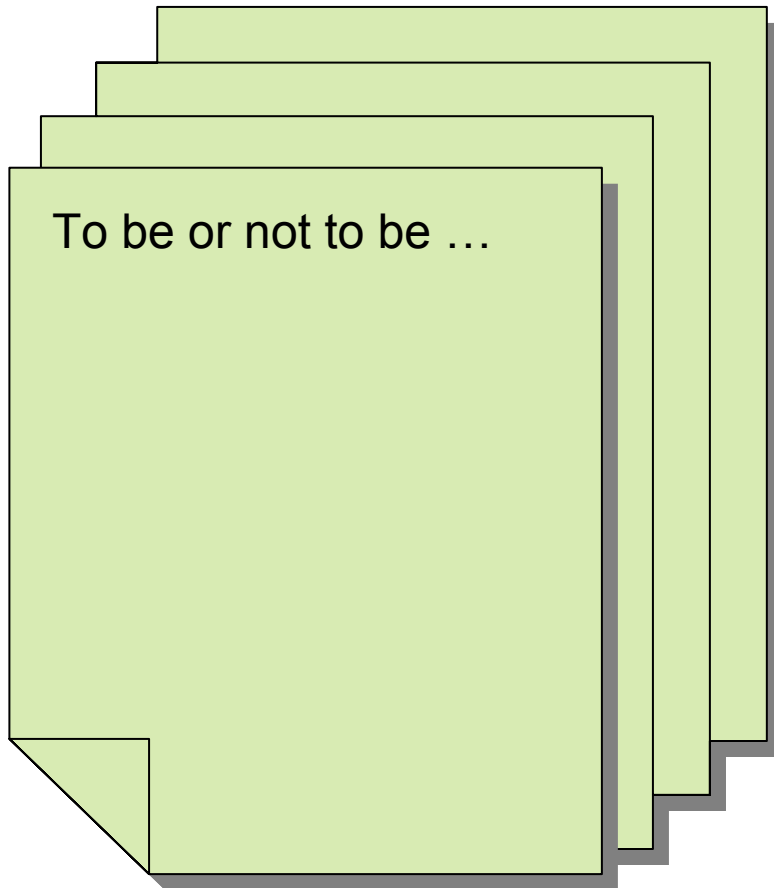
Piotr Indyk

MIT

Plan

- Intro
 - High dimensionality
 - Problems
- Technique: randomized projection
 - Intuition
 - Proofoid
- Applications:
 - Sketching/streaming
 - Nearest Neighbor Search
- Conclusions
- Refs

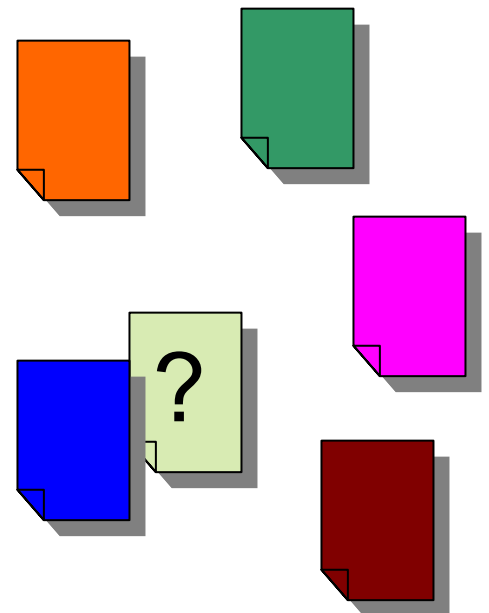
High-Dimensional Data



(... , 1, ..., 4, ... , 2 , ..., 2, ...)
(... , 6, ..., 1, ... , 3 , ..., 6, ...)
(... , 1, ..., 3, ... , 7 , ..., 5, ...)
(... , 2, ..., 2, ... , 1 , ..., 1, ...)
to be or not

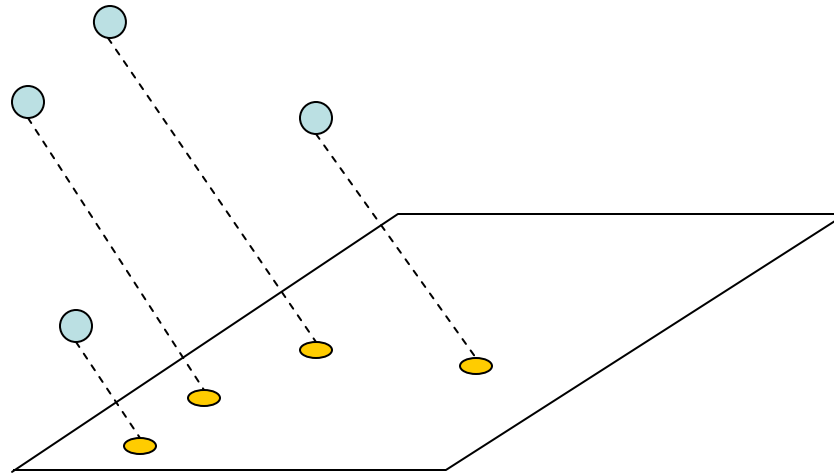
Problems

- Storage
 - How to represent the data “accurately” using “small” space
- Search
 - How to find “similar” documents
- Learning, etc....



Randomized Dimensionality Reduction

Randomized Dimensionality Reduction (a.k.a. “Flattening Lemma”)



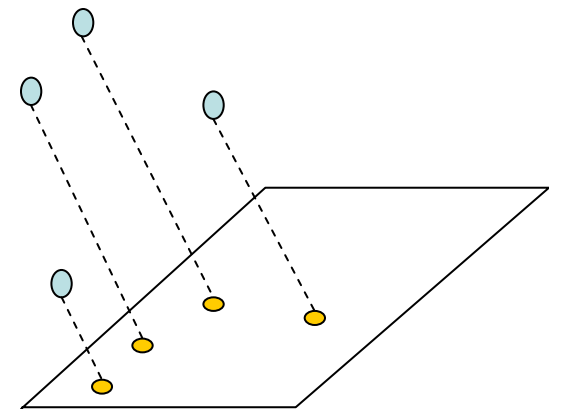
- Johnson-Lindenstrauss lemma (1984)
 - Choose the projection plane “at random”
 - The distances are “approximately” preserved with “high” probability

Dimensionality Reduction, Formally

- **JL:** For any set of n points X in \mathbb{R}^d under Euclidean norm, there is a $(1+\varepsilon)$ -distortion embedding of X into $\mathbb{R}^{d'}$, for $d' = O(\log n / \varepsilon^2)$
- **JL':** There is a distribution over random linear mappings $A: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, such that for any vector x we have $\|Ax\| = (1 \pm \varepsilon) \|x\|$ with probability

$$1 - e^{-Cd'\varepsilon^2}$$

- Questions:
 - What is the distribution ?
 - Why does it work ?



Normal Distribution

- Normal distribution:
 - Range: $(-\infty, \infty)$
 - Density: $f(x) = e^{-x^2/2} / (2\pi)^{1/2}$
 - Mean=0, Variance=1
- Basic facts:
 - If X and Y independent r.v. with normal distribution, then $X+Y$ has normal distribution
 - $\text{Var}(cX) = c^2 \text{Var}(X)$
 - If X, Y independent, then $\text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y)$

Back to the Embedding

- We use mapping Ax where each entry of A has normal distribution
- Let a^1, \dots, a^d be the rows of A
- Consider $Z = a^i * x = a * x = \sum_i a_i x_i$
- Each term $a_i x_i$
 - Has normal distribution
 - With variance x_i^2
- Thus, Z has normal distribution with variance $\sum_i x_i^2 = \|x\|^2$
- This holds for each a^j

What is $\|Ax\|_2$

- $\|Ax\|^2 = (a^1 * x)^2 + \dots + (a^{d'} * x)^2 = Z_1^2 + \dots + Z_{d'}^2$
where:
 - All Z_i 's are independent
 - Each has normal distribution with variance $\|x\|^2$
- Therefore, $E[\|Ax\|^2] = d' * E[Z_1^2] = d' \|x\|^2$
- By “law of large numbers” (quantitative):
$$\Pr[|\|Ax\|^2 - d' \|x\|^2| > \varepsilon d'] < e^{-C d' \varepsilon^2}$$
for some constant C

Streaming/sketching implications

- Can replace d -dimensional vectors by d' -dimensional ones
 - Cost: $O(dd')$ per vector
 - Faster method known [Ailon-Chazelle'06]
- Can avoid storing the original d -dimensional vectors in the first place

(thanks to linearity of the mapping A)

- Suppose:
 - x is the histogram of a document
 - We are receiving a stream of document words w_1, w_2, w_3, \dots
- For each word w , we want to update Ax to Ax' where $x'_w = x_w + 1$ (and the rest of x stays the same)
- Can be done via $Ax' = A(x + e_w) = Ax + Ae_w$
- Streaming algorithms [Alon-Matias-Szegedy'96]

(... , 2, ..., 2, ... , 1 , ..., 1, ...)

to be or not

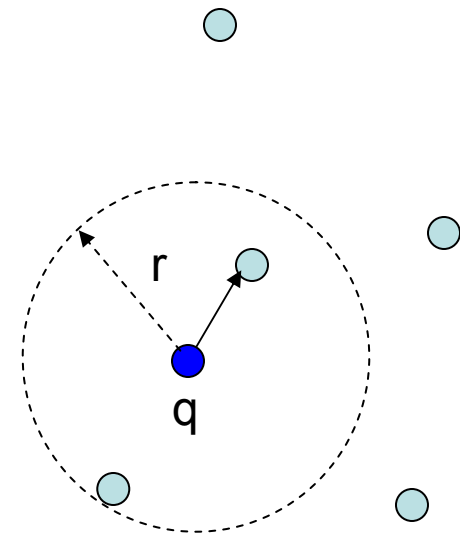
More Streaming/Sketching

- Generalizes to L_p norms, $p \in [0, 2]$
 - Generate matrix A from p -stable distribution
 - E.g., for $p=1$ we have Cauchy distribution
 - Estimate $\|x\|_p$ using
 - median($|a^1 x|, \dots, |a^d x|$) [Indyk'00]
 - geometric mean, harmonic mean [Church-Hastie-Li'05..07]
- Can handle “Jaccard coefficient” [Broder'97]
 - For two sets A, B , define $J(A, B) = |A \cap B| / |A \cup B|$
 - “Min-wise hashes”: functions h such that
$$\Pr[h(A) = h(B)] = J(A, B)$$
 - Can sketch set A into $\langle h_1(A), \dots, h_k(A) \rangle$
- Can reconstruct approximation of x from Ax

Nearest neighbors

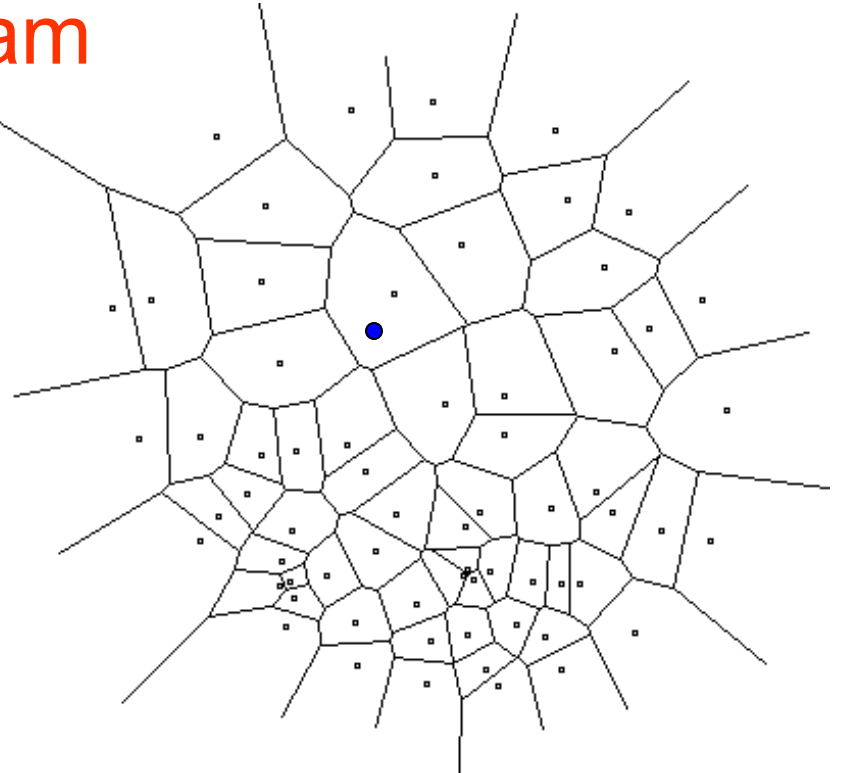
Near(est) neighbor

- Given: a set P of points in \mathbb{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: $O(\log n)$

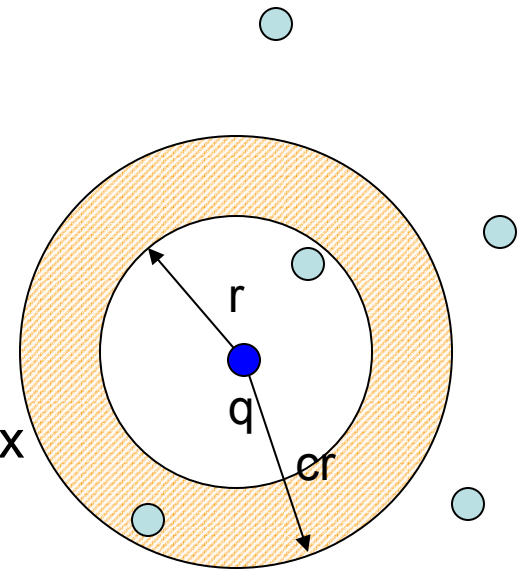


The case of $d > 2$

- Voronoi diagram has size $n^{O(d)}$
- We can also perform a linear scan: $O(dn)$ time
- That is pretty much all what known for exact algorithms with theoretical guarantees
- In practice:
 - kd-trees work “well” in “low-medium” dimensions
 - Near-linear query time for high dimensions

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 $p' \in P$, $\|p' - q\| \leq cr$
- Reductions:
 - **c**-Approx Nearest Neighbor reduces to **c**-Approx Near Neighbor (log overhead)
 - One can enumerate **all** approx near neighbors
 - can solve **exact** near neighbor problem
 - Other apps: **c**-approximate Minimum Spanning Tree, clustering, etc.



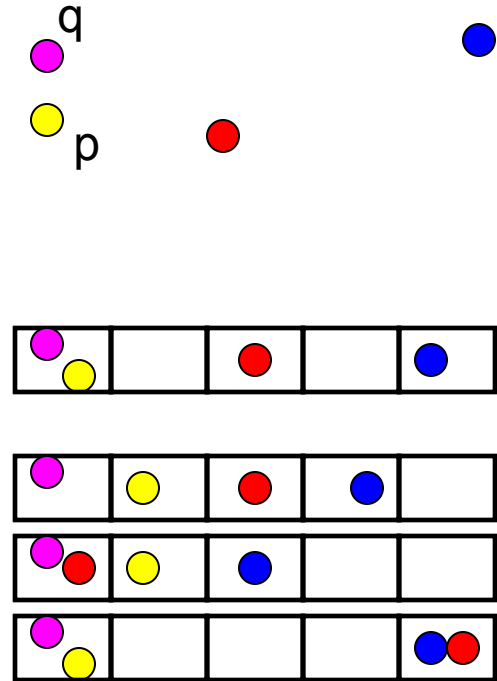
Approximate algorithms

- Space/time exponential in d [Arya-Mount-et al], [Kleinberg'97], [Har-Peled'02], [Arya-Mount-...]
- Space/time polynomial in d [Kushilevitz-Ostrovsky-Rabani'98], [Indyk-Motwani'98], [Indyk'98], [Gionis-Indyk-Motwani'99], [Charikar'02], [Datar-Immorlica-Indyk-Mirroknj'04], [Chakrabarti-Regev'04], [Panigrahy'06], [Ailon-Chazelle'06]...

Space	Time	Comment	Norm	Ref
$dn+n^{4/\varepsilon^2}$	$d * \log n / \varepsilon^2$ 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], [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]
		$\sigma(c)=O(1/c)$	l_2	[Pan'06]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c^2 + o(1)$	l_2	[AI'06]
$dn * \log s$	$dn^{\sigma(c)}$	$\sigma(c)=O(1/c^2)$	l_2	[AI'06]

Locality-Sensitive Hashing

- Idea: construct hash functions $g: \mathbb{R}^d \rightarrow \mathbb{U}$ such that for any points p, q :
 - If $\|p-q\| \leq r$, then $\Pr[g(p)=g(q)]$ is ~~“high”~~ “not-so-small”
 - If $\|p-q\| > cr$, then $\Pr[g(p)=g(q)]$ is “small”
- Then we can solve the problem by hashing



LSH [Indyk-Motwani'98]

- A family H of functions $h: \mathbb{R}^d \rightarrow U$ is called (P_1, P_2, r, cr) -sensitive, if for any p, q :
 - if $\|p-q\| < r$ then $\Pr[h(p)=h(q)] > P_1$
 - if $\|p-q\| > cr$ then $\Pr[h(p)=h(q)] < P_2$
- Examples:
 - Hamming distance
 - LSH functions: $h(p)=p_i$, i.e., the i -th bit of p
 - Probabilities: $\Pr[h(p)=h(q)] = 1-D(p,q)/d$
 - Jaccard coefficient
 - Min-wise hashing (slide 12)

$p=10010010$
 $q=11010110$

LSH Algorithm

- We use functions of the form

$$g(p) = \langle h_1(p), h_2(p), \dots, h_k(p) \rangle$$

- Preprocessing:

- Select $g_1 \dots g_L$
- For all $p \in P$, hash p to buckets $g_1(p) \dots g_L(p)$

- Query:

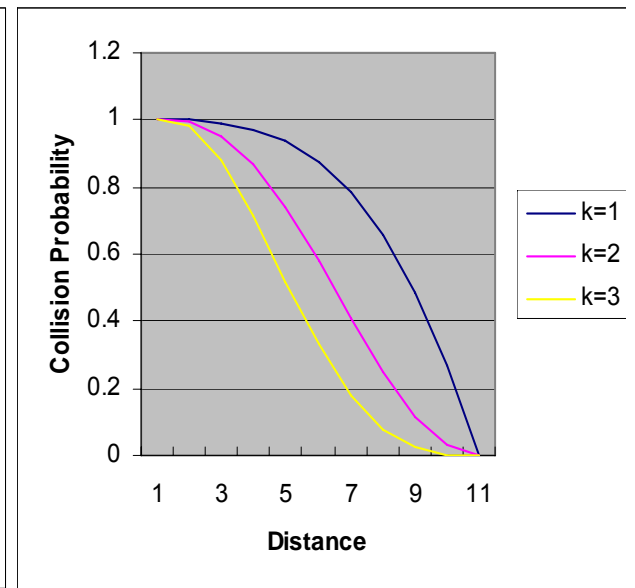
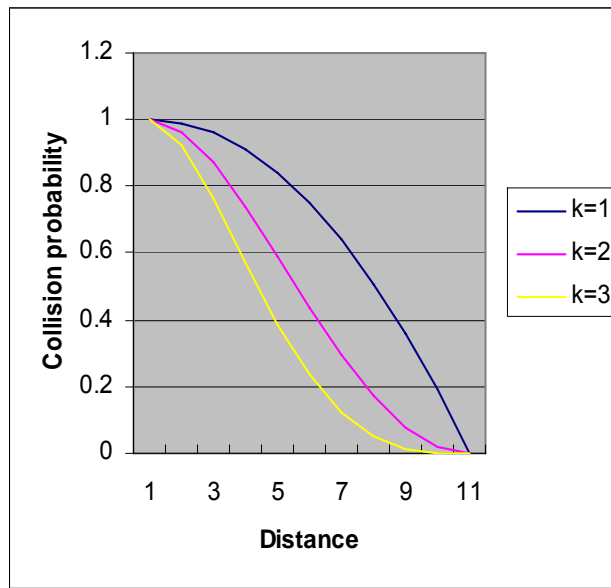
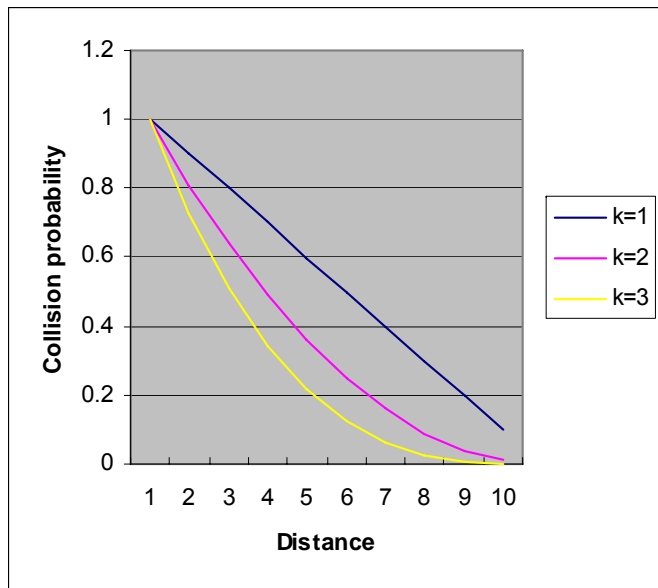
- Retrieve the points from buckets $g_1(q), g_2(q), \dots$, until
 - Either the points from all L buckets have been retrieved, or
 - Total number of points retrieved exceeds $2L$
- Answer the query based on the retrieved points
- Total time: $O(dL)$

Analysis

- LSH solves c -approximate NN with:
 - Number of hash fun: $L=n^\rho$,
 $\rho=\log(1/P1)/\log(1/P2)$
 - E.g., for the Hamming distance we have
 $\rho=1/c$
 - Constant success probability per query q

Proof by picture

- Hamming distance
- Collision prob. for $k=1..3$, $L=1..3$ (recall: $L=\#\text{indices}$, $k=\#\text{h's}$)
- Distance ranges from 0 to 10 (max)



- The argument can be massaged to show that

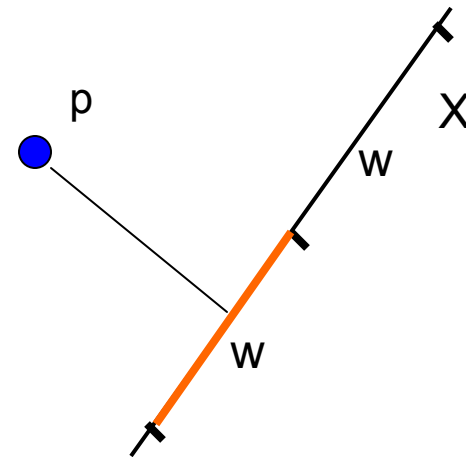
$$L = n^{\rho}, \quad \rho = \log_{1/P_2}(1/P_1)$$

works with constant probability.

Projection-based LSH

[Datar-Immorlica-Indyk-Mirroknii'04]

- Define $h_{X,b}(p) = \lfloor (p \cdot X + b) / w \rfloor$:
 - $w \approx r$
 - $X = (X_1 \dots X_d)$, where X_i is chosen from:
 - Gaussian distribution (for l_2 norm)
 - “s-stable” distribution* (for l_s norm)
 - b is a scalar
- Simple enough
- Code available [Andoni-Indyk'05]

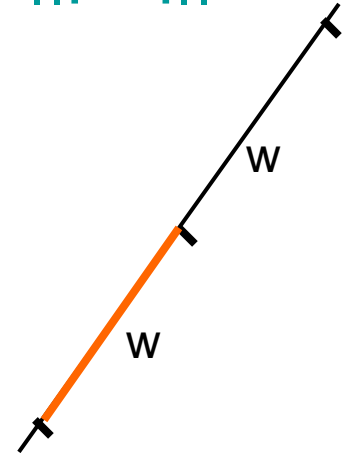


Analysis

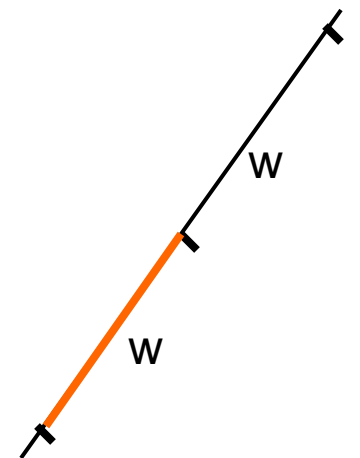
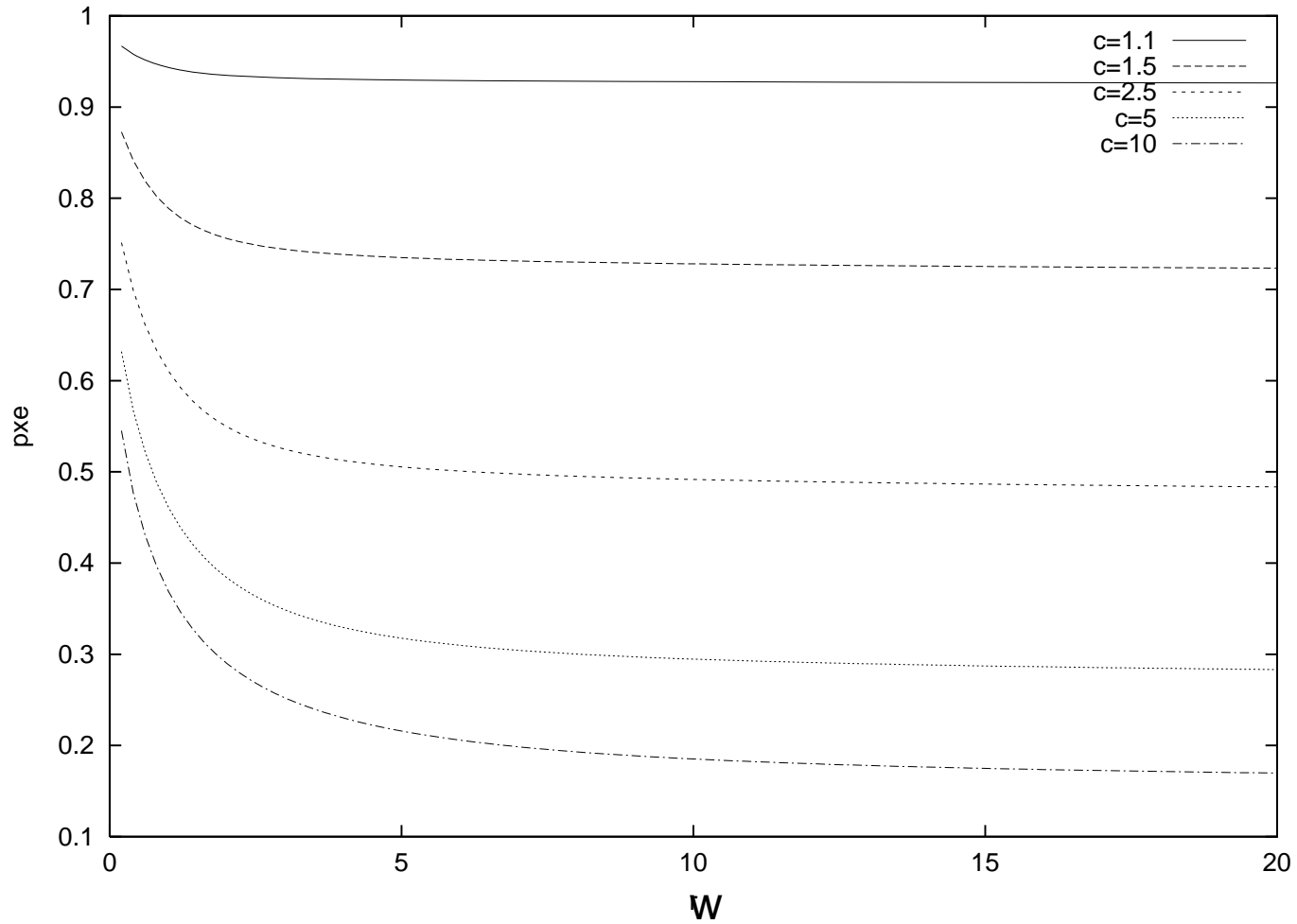
- Need to:
 - Compute $\Pr[h(p)=h(q)]$ as a function of $\|p-q\|$ and w ; this defines P_1 and P_2
 - For each c choose w that minimizes

$$\rho = \log_{1/P_2}(1/P_1)$$

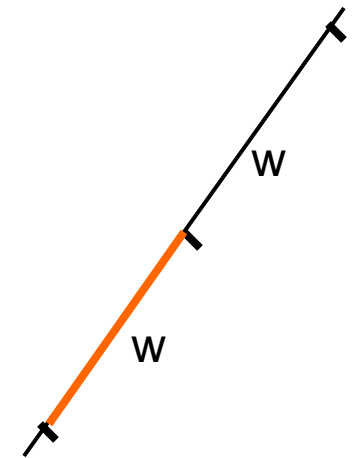
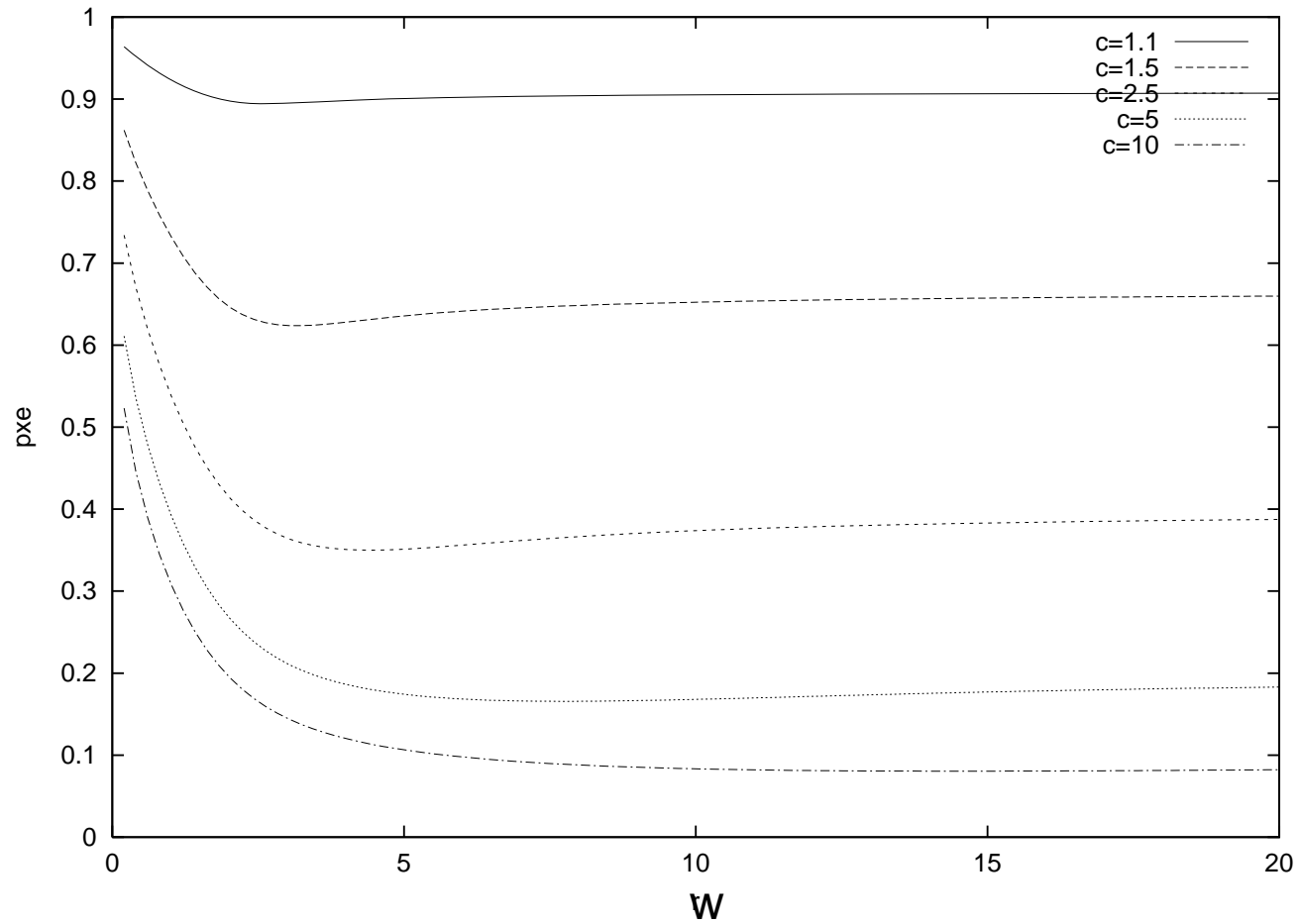
- Method:
 - For l_2 : computational
 - For general l_s : analytic



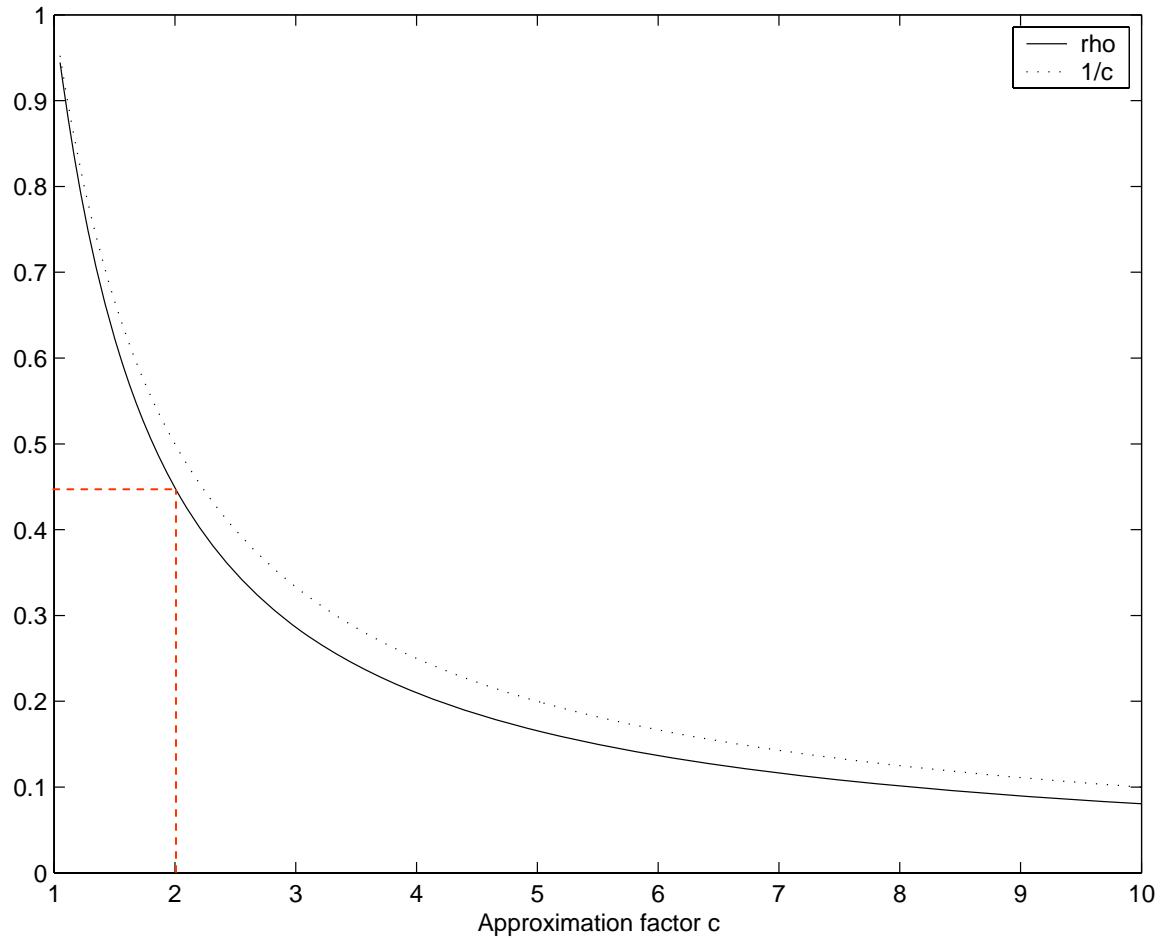
$\rho(w)$ for various c 's: I_1



$\rho(w)$ for various c 's: I_2



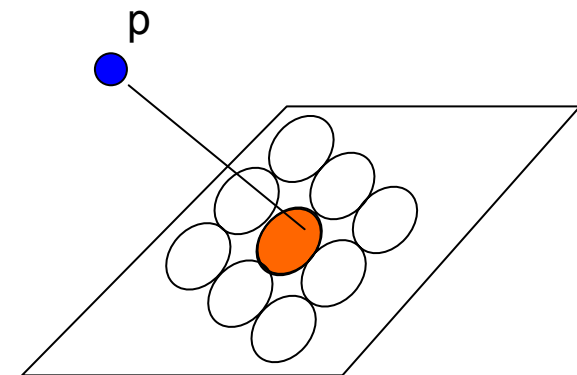
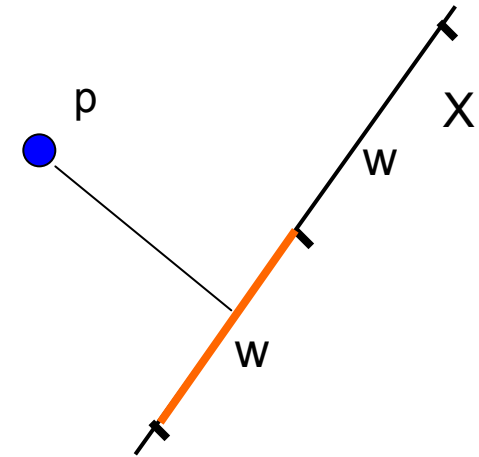
$\rho(c)$ for l_2



New LSH scheme

[Andoni-Indyk'06]

- Instead of projecting onto \mathbb{R}^1 , project onto \mathbb{R}^t , for constant t
- Intervals \rightarrow lattice of balls
 - Can hit empty space, so hash until a ball is hit
- Analysis:
 - $\rho = 1/c^2 + O(\log t / t^{1/2})$
 - Time to hash is $t^{O(t)}$
 - Total query time: $dn^{1/c^2 + o(1)}$
- [Motwani-Naor-Panigrahy'06]: LSH in l_2 must have $\rho \geq 0.45/c^2$



Conclusions

- Overview of randomized approximate algorithms for high-dimensional data
 - Reduce space
 - Reduce time
- Randomized dimensionality reduction plays important role
 - Source of randomization and approximation

If you would like to RTFM

- Random projections: monograph by S. Vempala
- Nearest neighbor in high dimensions:
 - CRC Handbook'03 (my web page)
 - CACM Survey (draft, on request)
- Streaming:
 - Survey: S. Muthu Muthukrishnan (see his web page)
 - Summer school +materials: Google “Madalgo”
- Streaming for CL: [Church-Hastie-Li, ACL'05]
- LSH for CL: [Ravichandran-Pantel-Hovy, ACL'05]
(use related algorithm by [Charikar'02])
- LSH for web clustering: [Broder et al, WWW'97], [Gionis et al, WebDB'00, WWW'02]
- Code available (see my web page)

Thanks!

- To the organizers
- To Mike and Regina
- To you 😊

PCA vs JL

- Technical differences: average square error (PCA) vs maximum error (JL)
- PCA advantage:
 - Data dependent
 - Can adjust to distribution
- PCA disadvantage:
 - Data dependent
 - Requires linear storage, and linear update time if data set changes

Experiments

LSH Experiments (with '04 version)

- **E²LSH**: Exact Euclidean LSH (with Alex Andoni)
 - Near Neighbor
 - User sets r and P = probability of NOT reporting a point within distance r (=10%)
 - Program finds parameters k, L, w so that:
 - Probability of failure is at most P
 - Expected query time is minimized
- **Nearest neighbor**: set radius (radiae) to accommodate 90% queries (results for 98% are similar)
 - 1 radius: 90%
 - 2 radiae: 40%, 90%
 - 3 radiae: 40%, 65%, 90%
 - 4 radiae: 25%, 50%, 75%, 90%

Data sets

- MNIST OCR data, normalized (LeCun)
 - $d=784$
 - $n=60,000$
- Corel_hist
 - $d=64$
 - $n=20,000$
- Corel_uci
 - $d=64$
 - $n=68,040$
- Aerial data (Manjunath)
 - $d=60$
 - $n=275,476$

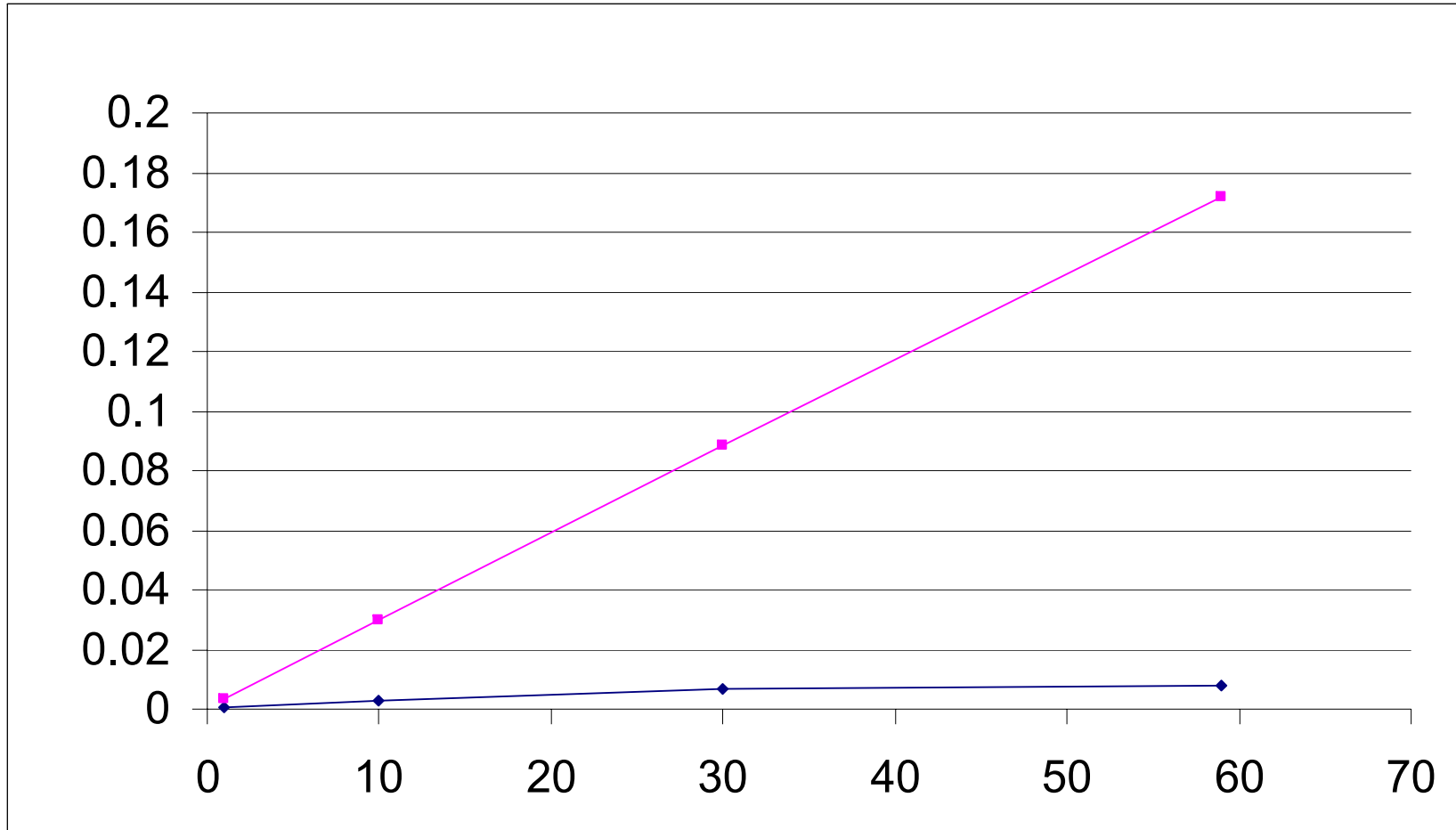
Other NN packages

- ANN (by Arya & Mount):
 - Based on kd-tree
 - Supports exact and approximate NN
- Metric trees (by Moore et al):
 - Splits along arbitrary directions (not just x,y,..)
 - Further optimizations

Running times

	MNIST	Speedup	Corel_hist	Speedup	Corel_uci	Speedup	Aerial	Speedup
E2LSH-1	0.00960							
E2LSH-2	0.00851		0.00024		0.00070		0.07400	
E2LSH-3			0.00018		0.00055		0.00833	
E2LSH-4							0.00668	
ANN	0.25300	29.72274	0.00018	1.011236	0.00274	4.954792	0.00741	1.109281
MT	0.20900	24.55357	0.00130	7.303371	0.00650	11.75407	0.01700	2.54491

LSH vs kd-tree (MNIST)



Caveats

- For ANN (MNIST), setting $\epsilon=1000\%$ results in:
 - Query time comparable to LSH
 - Correct NN in about 65% cases, small error otherwise
- However, no guarantees
- LSH eats much more space (for optimal performance):
 - LSH: 1.2 GB
 - Kd-tree: 360 MB