Locality-Sensitive Hashing: Theory and Applications

Ludwig Schmidt

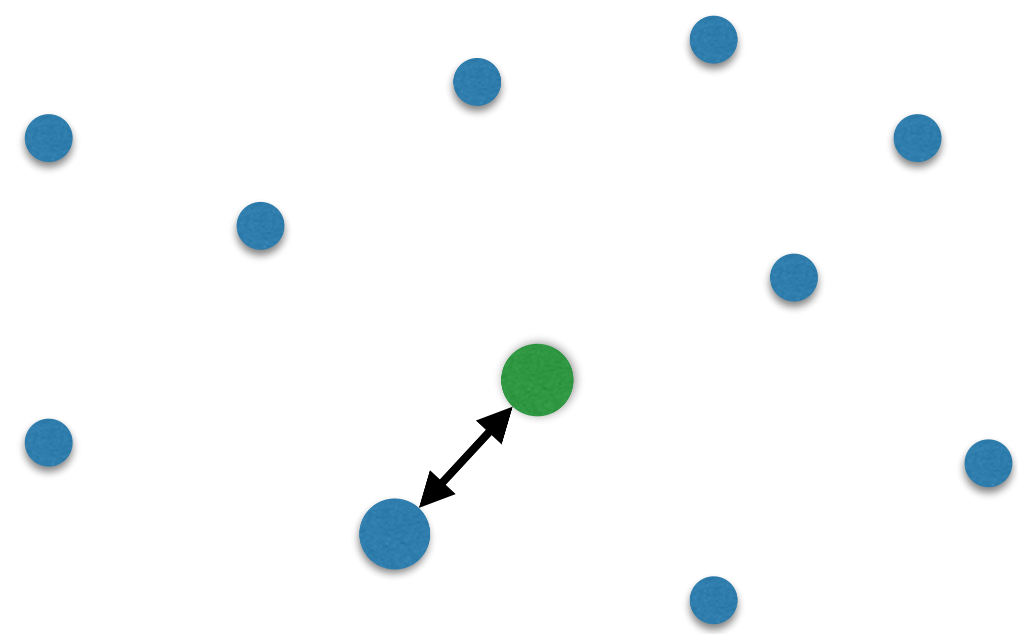
MIT → Google Brain → UC Berkeley

Based on joint works with Alex Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Kunal Talwar.

Nearest Neighbor Search

1. Build a data structure for a given **set of points** in R^d.

2. (Repeated) For a given **query**, find the **closest** point in the dataset with "good" probability.



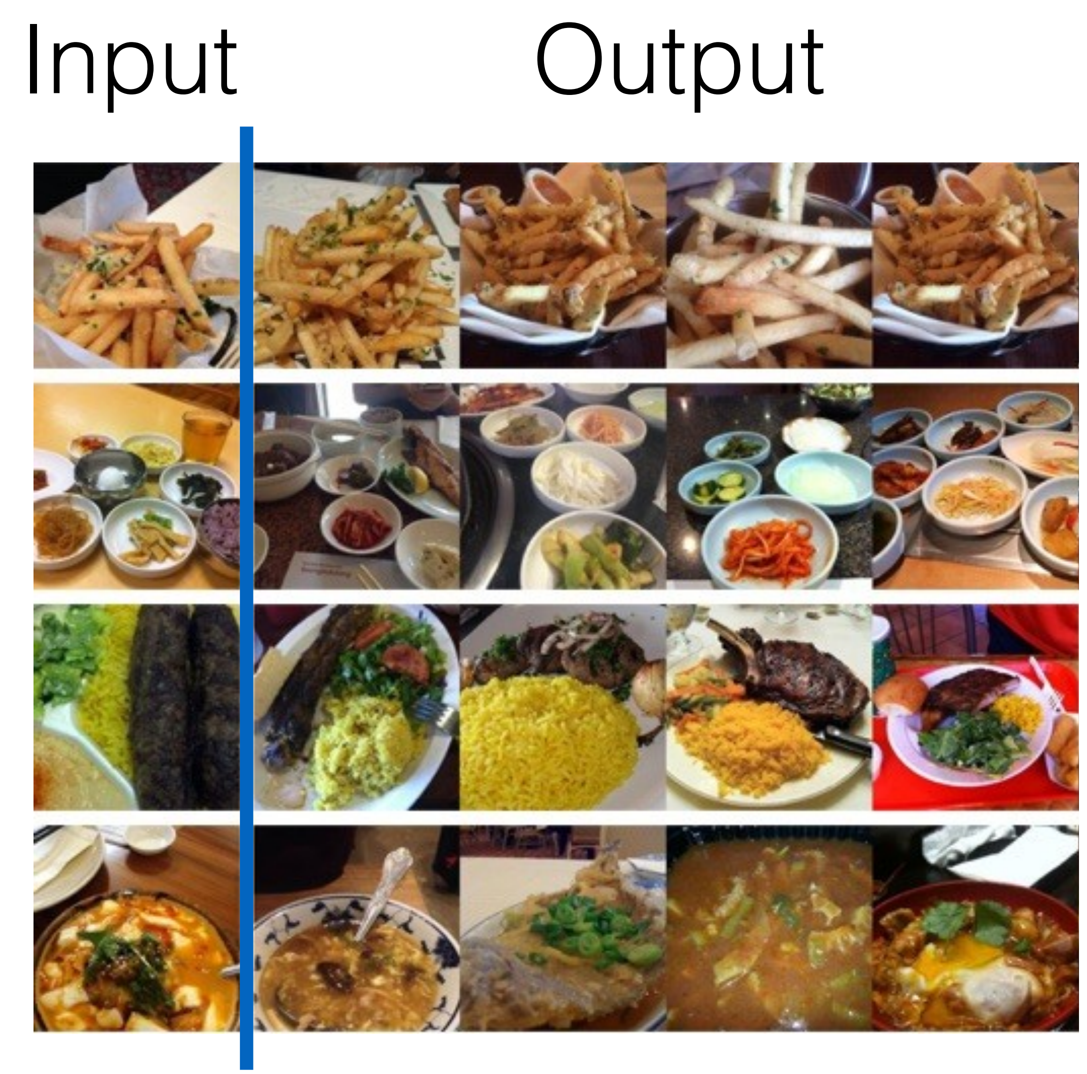
Main goal: **fast** queries with **high accuracy.**

Motivation

Large number of applications

Data retrieval

- Images (SIFT, …)
- Text (tf-idf, …)
- Audio (i-vectors, …)



Motivation

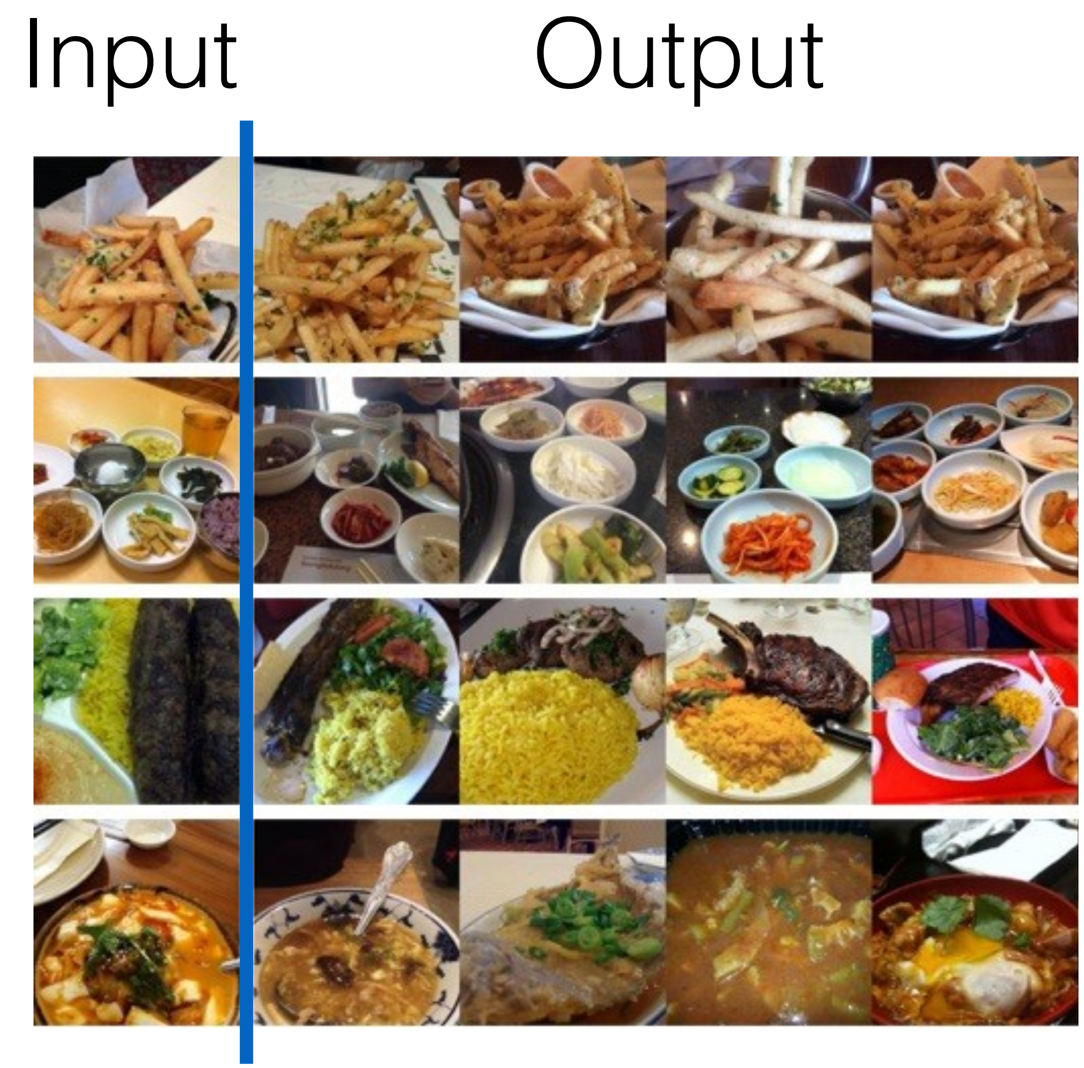
Large number of applications

Data retrieval

- Images (SIFT, …)
- Text (tf-idf, …)
- Audio (i-vectors, …)

Sub-routine in other algorithms

- Optimization
- Cryptanalysis
- Classification

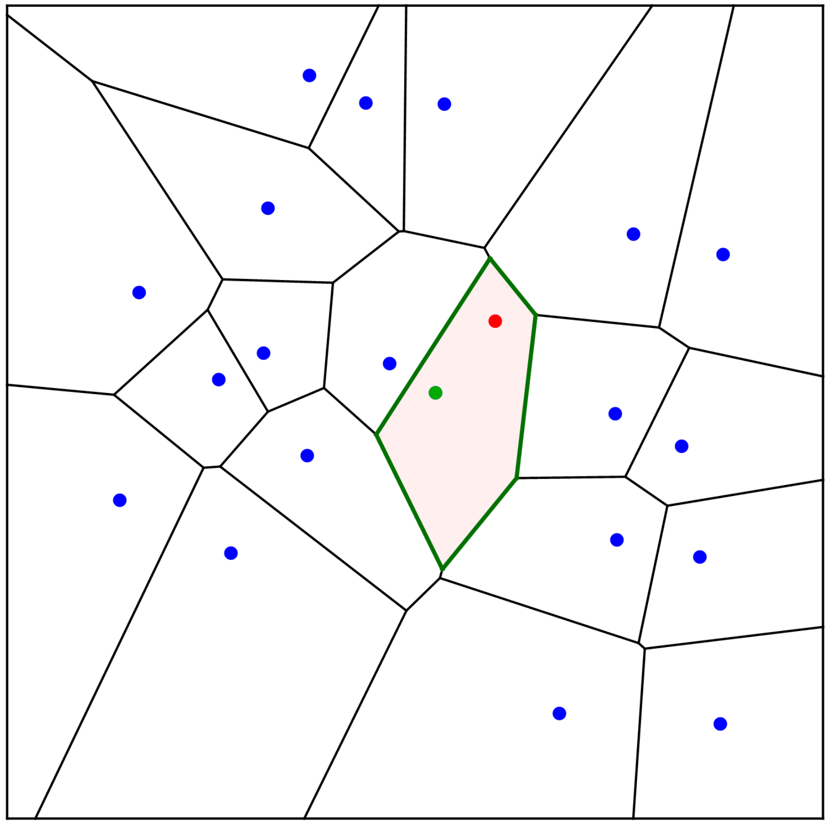


A Simple Problem?

In 2D: **Voronoi diagram**

- O(n log n) setup time
- O(log n) query time

Almost ideal data structure!

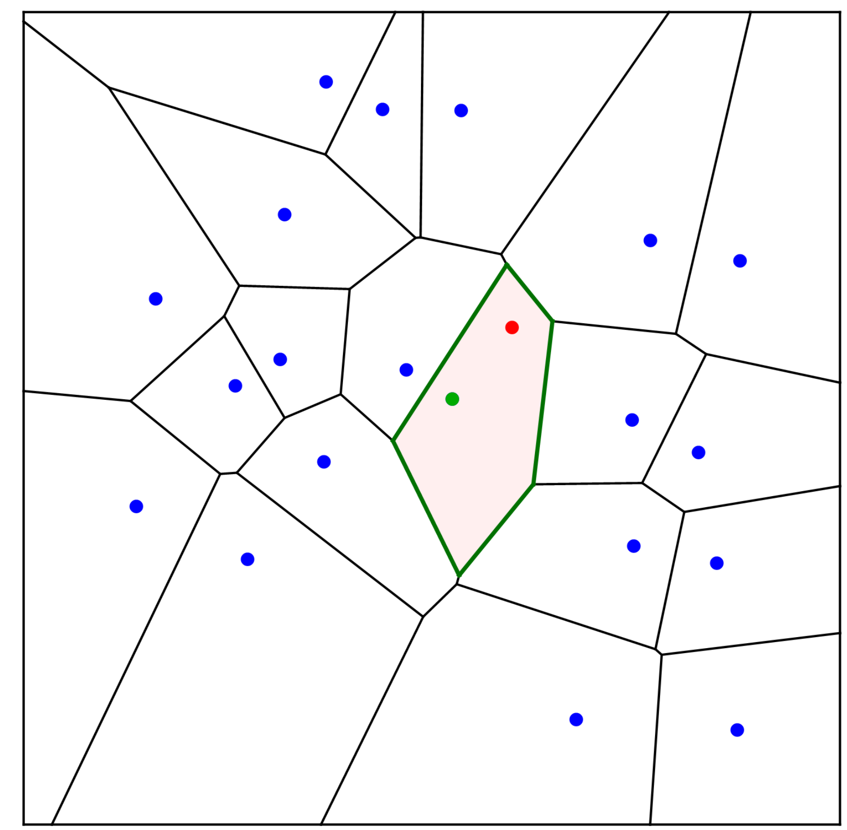


A Simple Problem?

In 2D: **Voronoi diagram**

- O(n log n) setup time
- O(log n) query time

Almost ideal data structure!



Problem?

Many applications require high-dimensional spaces.

→ **"Curse of dimensionality"**

$\mathbb{R}^d$

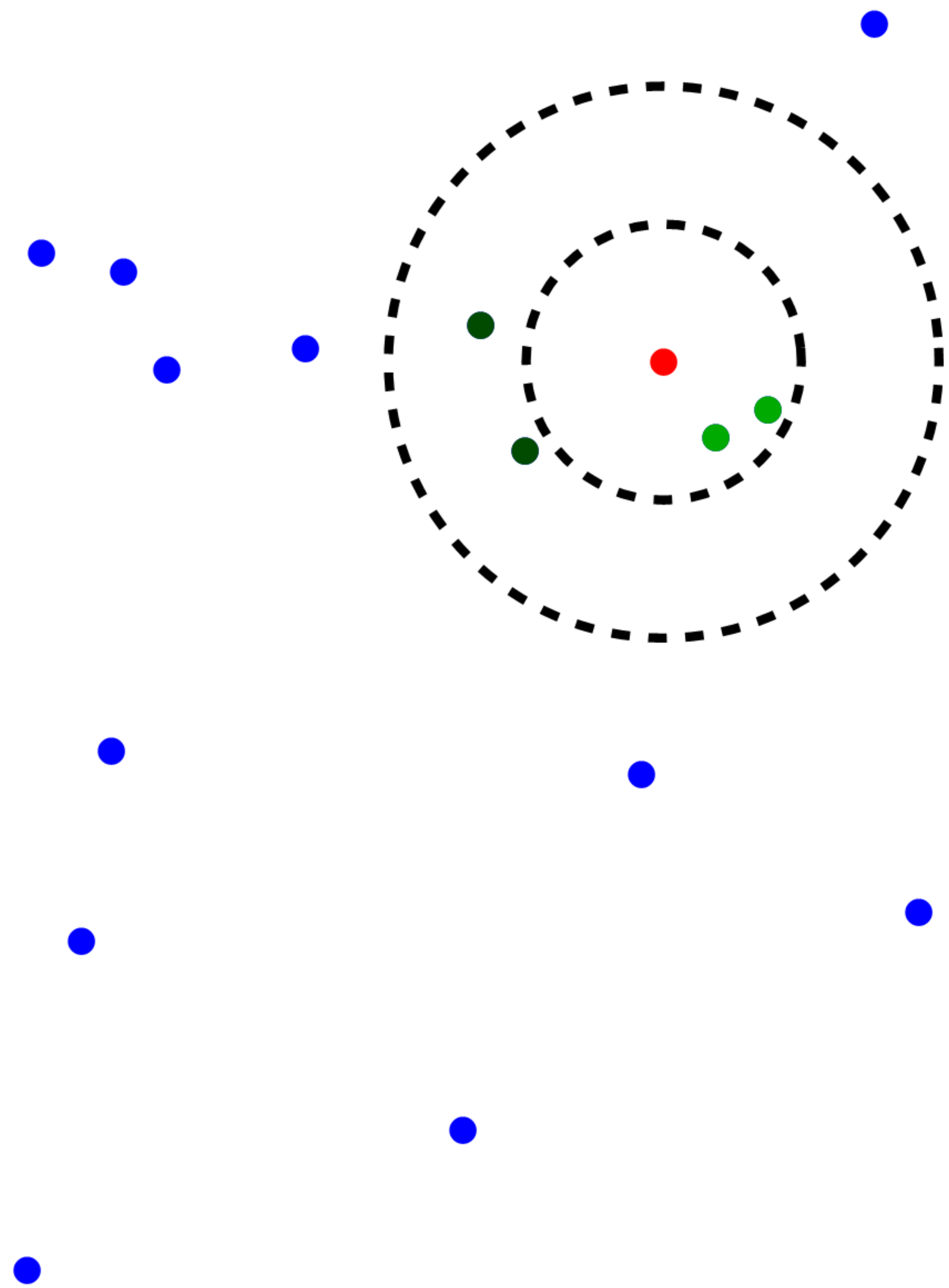
Rescue from high Dimensionality

Real data often has **structure.**

Example: significant gap

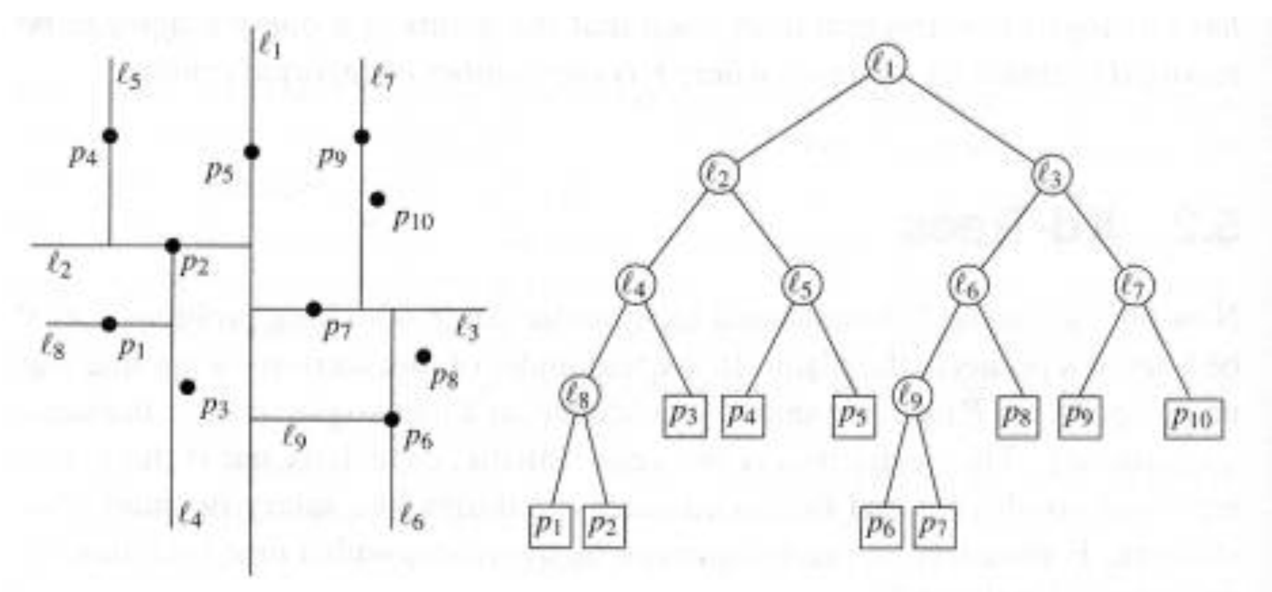
between distance from query to

- nearest neighbor
- average point



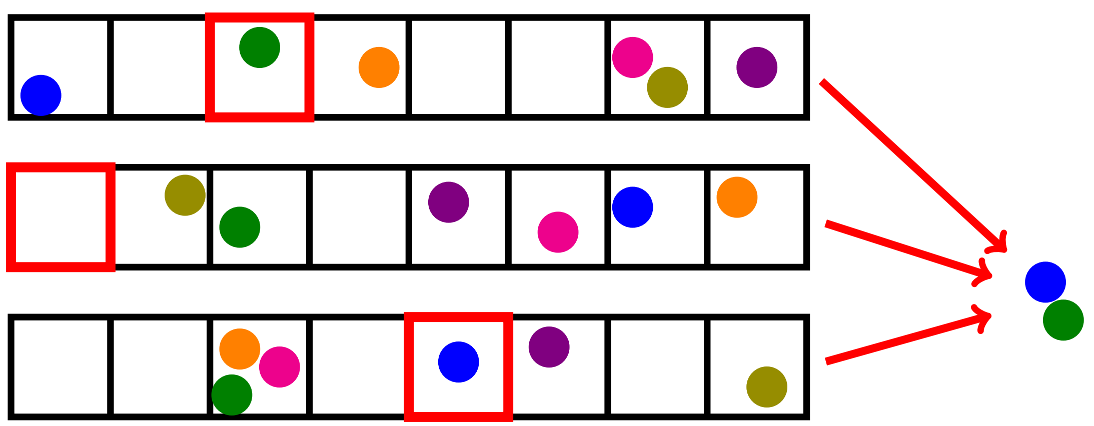
Common Methods

Tree data structures



Locality-sensitive hashing

Vector quantization



Nearest-neighbor graphs

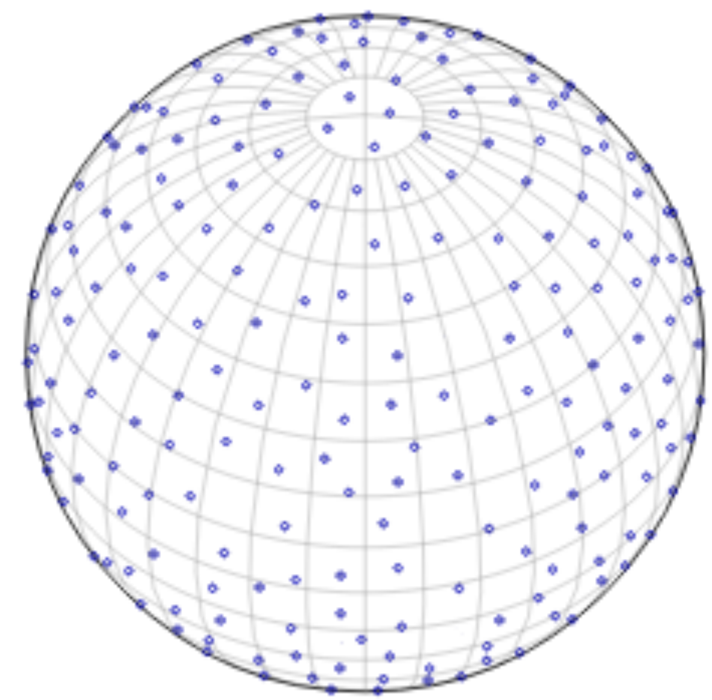
1. Locality-Sensitive Hashing
2. Cross-Polytope Hash
3. LSH in Neural Networks

1. Locality-Sensitive Hashing
2. Cross-Polytope Hash
3. LSH in Neural Networks

Formal Problem

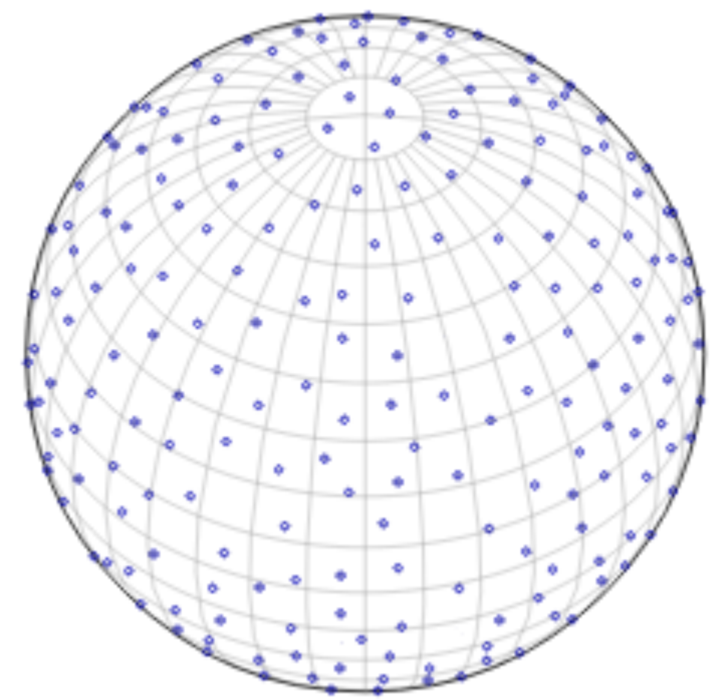
Spherical case

- Points are on the unit sphere.
- Angles between most points around 90°.



Formal Problem

Spherical case



- Points are on the unit sphere.
- Angles between most points around 90°.

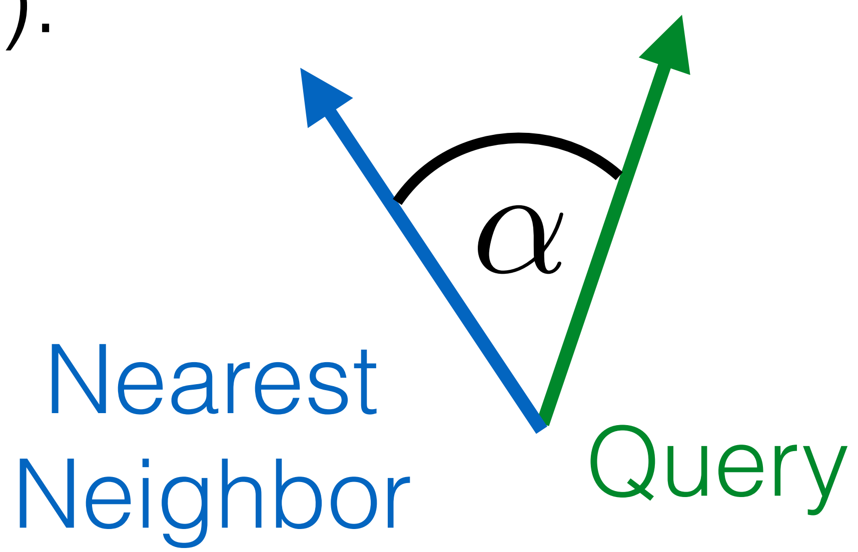
Why?

- **Theory**: general case reduces to this case.
- **Practice**: good model for many (pre-processed) instances.

Formal Problem

Similarity Measures (all equivalent here):

- Cosine Similarity
- Angular Distance
- Euclidean Distance



Why? Often (approximately) the goal in practice.

Locality-Sensitive Hashing

Introduced in [Indyk, Motwani, 1998].

Main idea: partition R^d randomly such that nearby points are more likely to appear in the same cell of the partition.

Locality-Sensitive Hashing

Introduced in [Indyk, Motwani, 1998].

Main idea: partition R^d randomly such that nearby points are more likely to appear in the same cell of the partition.

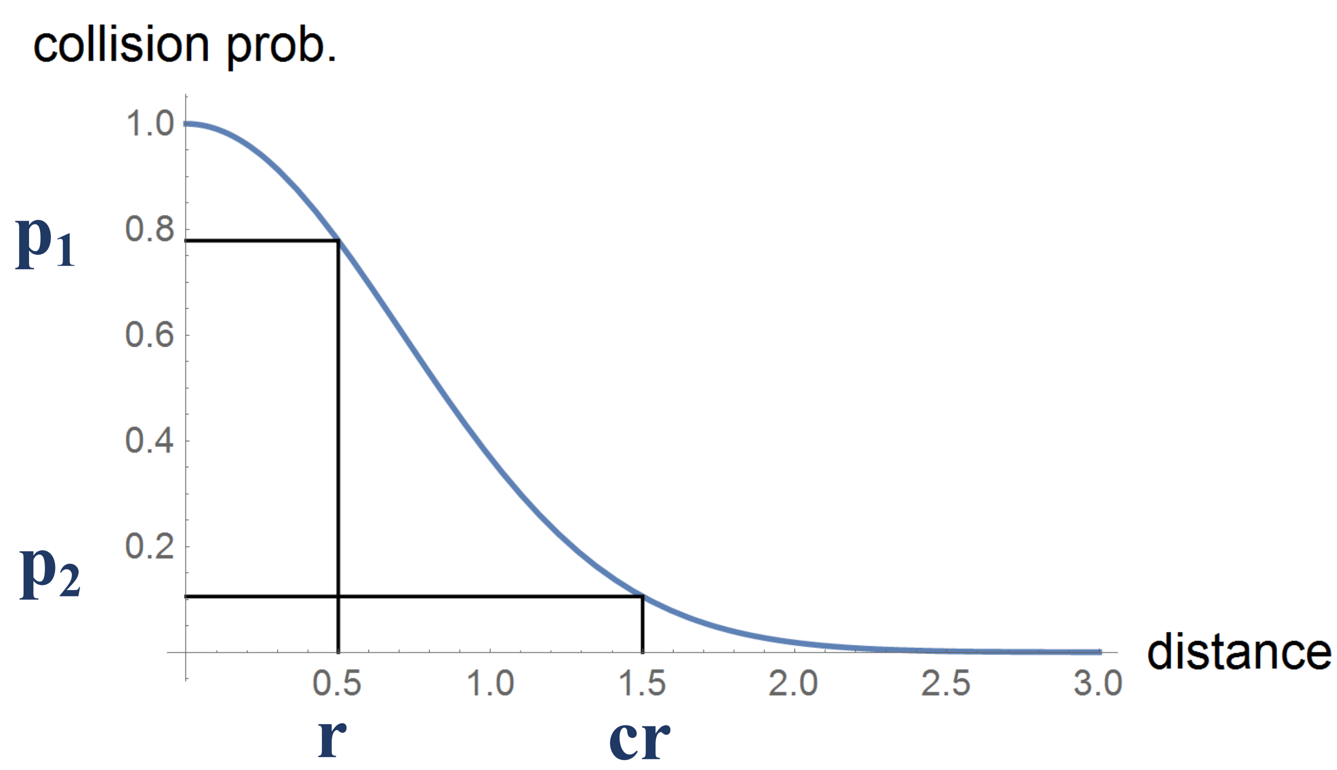
What about **hash functions**?

Random hash function
  = random space partitioning

LSH: Formal Definition

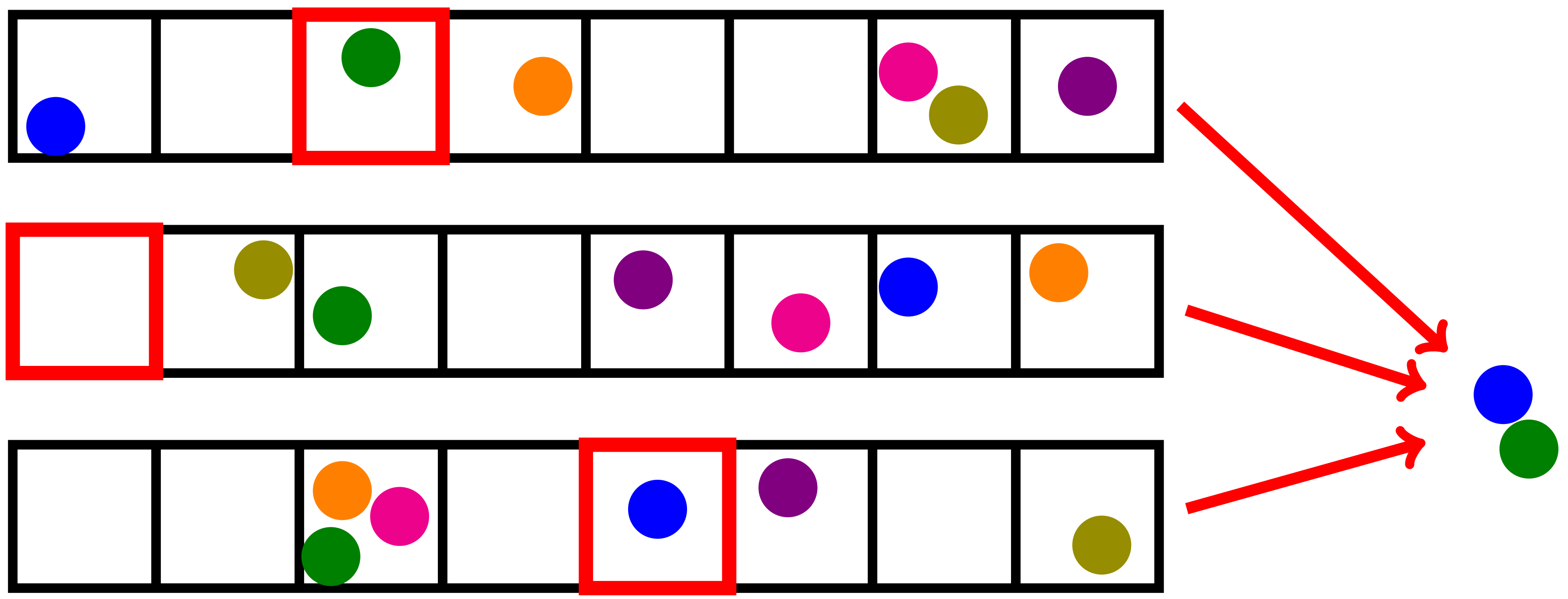
A family of hash functions is (r, $c\,r$, p_1, p_2)-locality sensitive if for every p, q in R^d, the following holds:

- if $\| p - q \| < r$, then $P[h(p) = h(q)] > p_1$
- if $\| p - q \| > c\,r$, then $P[h(p) = h(q)] < p_2$



LSH: Data Structure

Multiple hash tables with independent hash functions.



Query

1. Find candidates in hash buckets h(q).
2. Compute exact distances for candidates.

LSH: Theory

Query time quantified as a function of **sensitivity** ρ.

Intuitively: gap between "nearby" collision probability and "far away" collision probability.

Formally: $$\rho = \frac{\log 1/p_1}{\log 1/p_2}$$

LSH: Theory

Query time quantified as a function of **sensitivity** ρ.

Intuitively: gap between "nearby" collision probability
and "far away" collision probability.

Formally: $$\rho = \frac{\log 1/p_1}{\log 1/p_2}$$

For example: often $\rho \approx 1/c$ where "nearby" is distance r
and "far away" is distance c r.

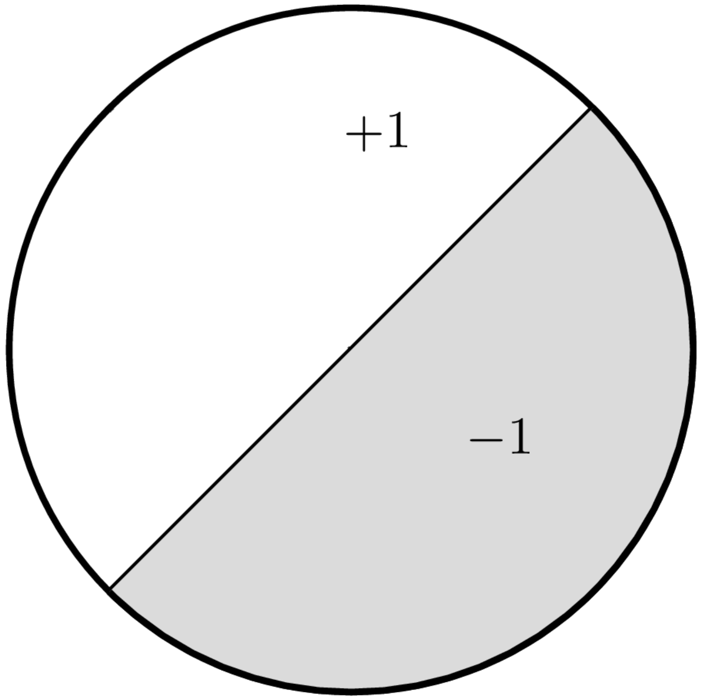
Query time is $O(n^{\rho})$, data structure size $O(n^{1+\rho})$.

Most Common LSH Family

Hyperplane LSH

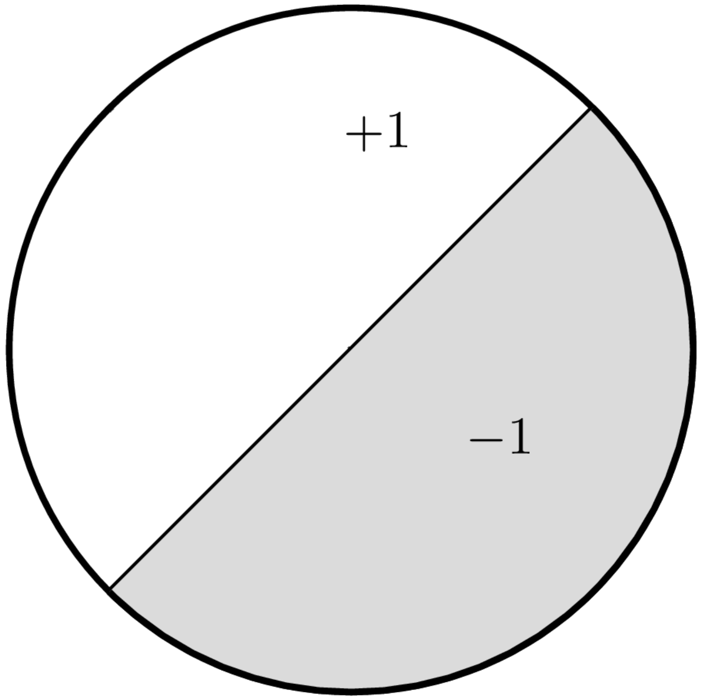
Introduced in [Charikar 2002], inspired by [Goemans, Williamson 1995].

Hash function: partition the sphere with a random hyperplane.



Most Common LSH Family

Hyperplane LSH



Introduced in [Charikar 2002], inspired by [Goemans, Williamson 1995].

Hash function: partition the sphere with a random hyperplane.

Locality-sensitive: let α be the angle between points p and q:

$$P[h(p) = h(q)] \;=\; 1 - \frac{\alpha}{\pi}$$

Empirical State of the Art, 2015

GloVe dataset
(word embeddings)

100-dim
1.2 mio vectors

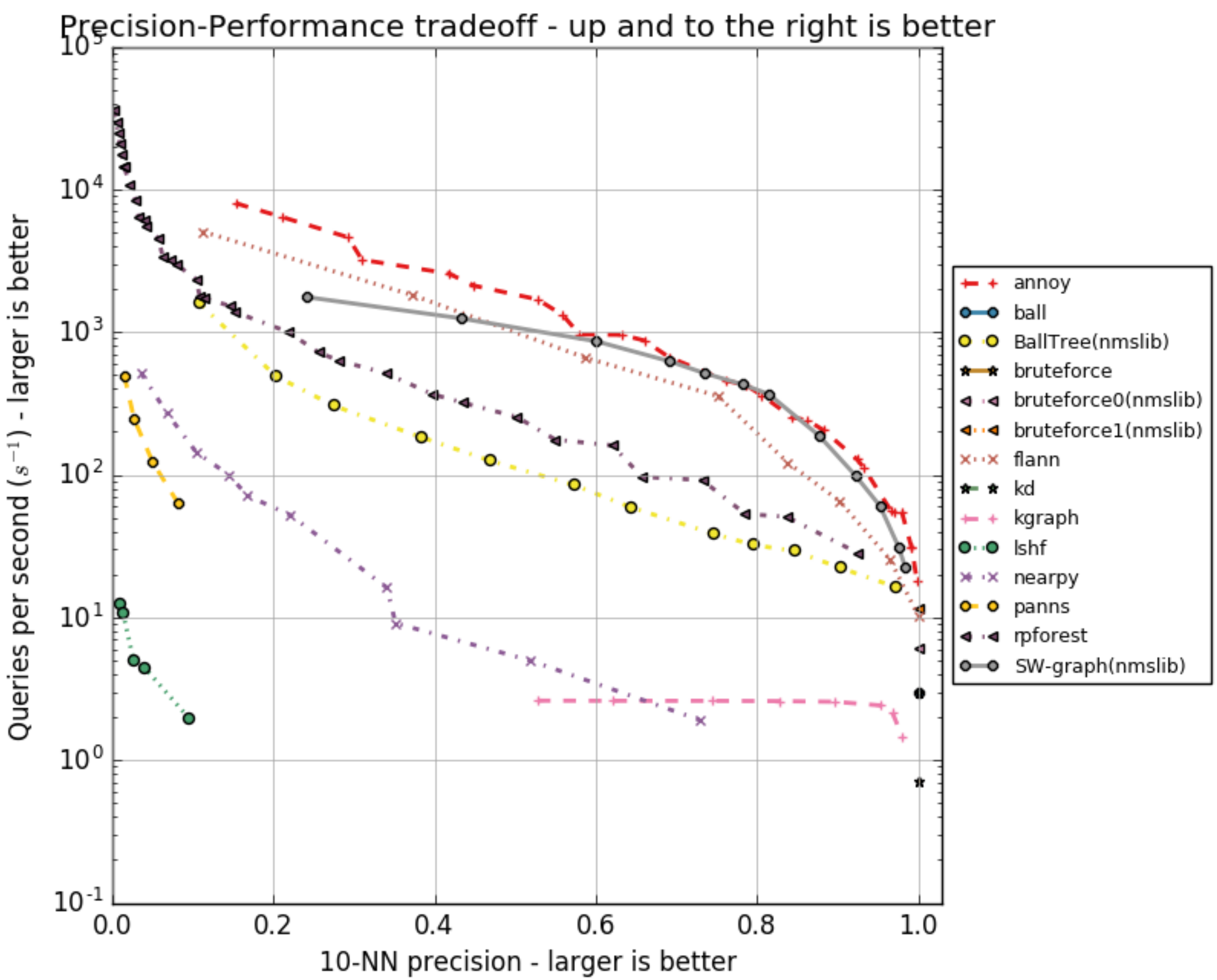
Source:
ann-benchmarks
Erik Bernhardsson

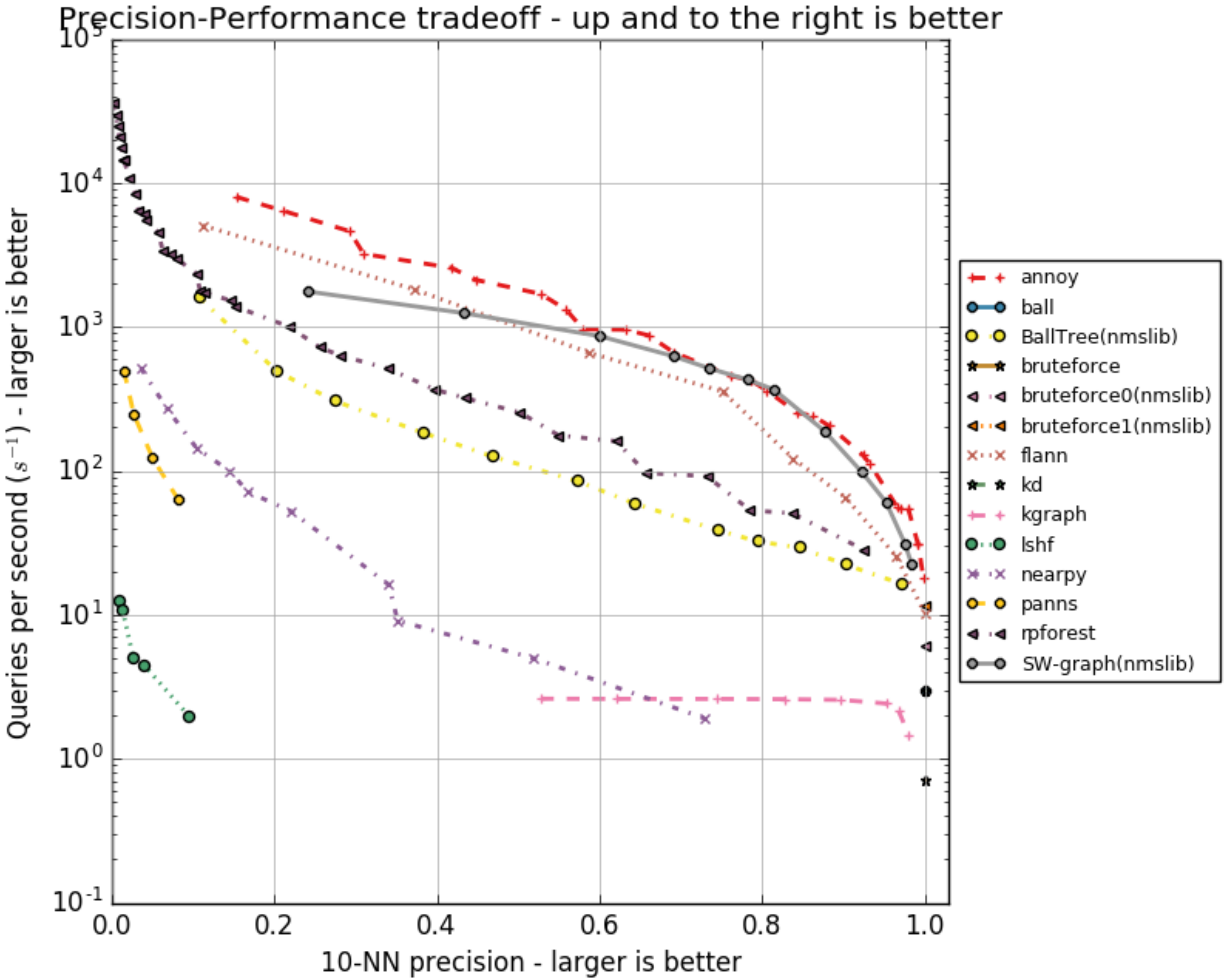
Empirical State of the Art, 2015

GloVe dataset
(word embeddings)

100-dim
1.2 mio vectors

Source:
ann-benchmarks
Erik Bernhardsson




Precision-Performance tradeoff - up and to the right is better
Queries per second (s^{-1}) - larger is better
10-NN precision - larger is better
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
0.0
0.2
0.4
0.6
0.8
1.0
annoy
ball
BallTree(nmslib)
bruteforce
bruteforce0(nmslib)
bruteforce1(nmslib)
flann
kd
kgraph
lshf
nearpy
panns
rpforest
SW-graph(nmslib)

Empirical State of the Art, 2015

SIFT dataset
(word embeddings)

128-dim
1 mio vectors

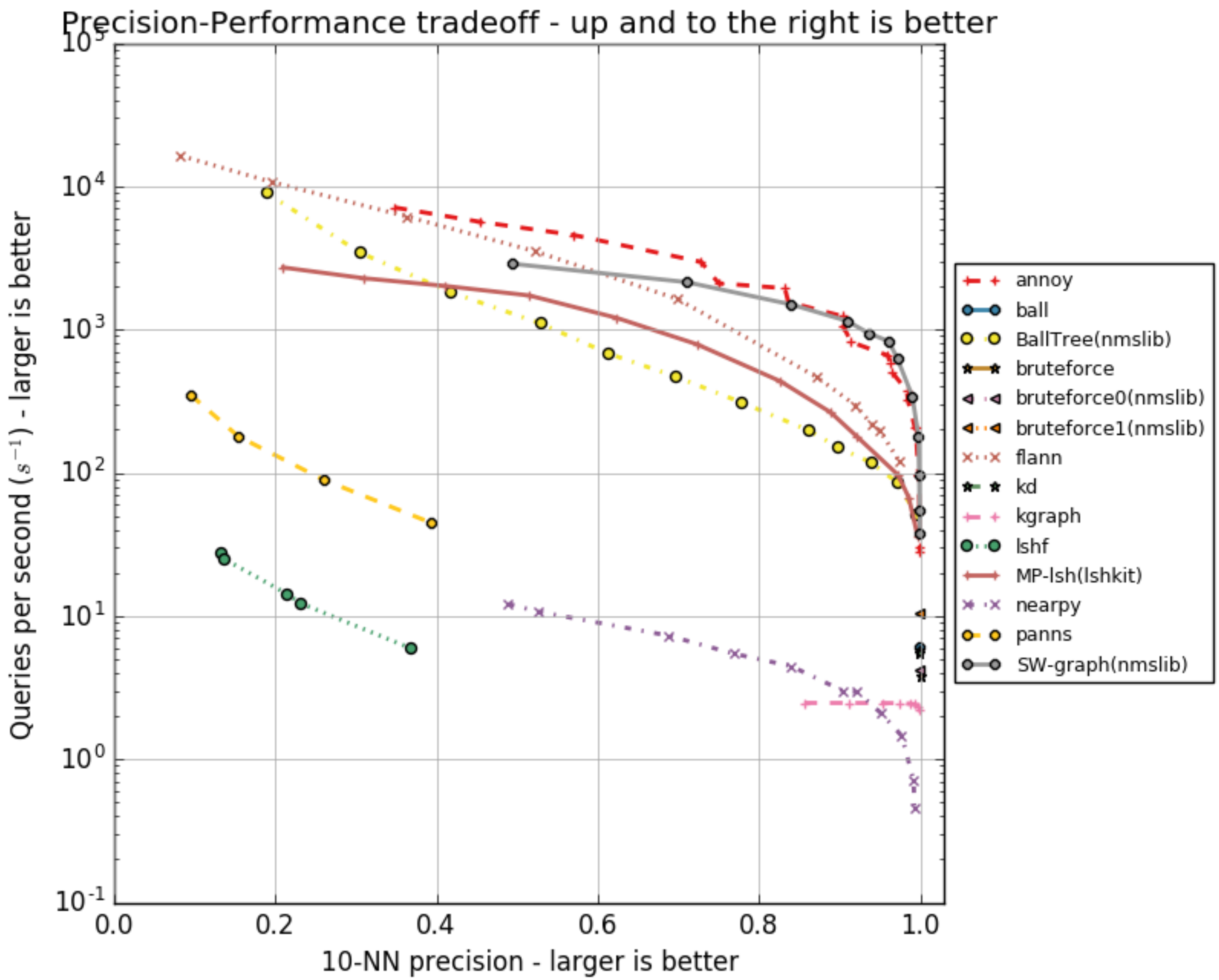
Source:
ann-benchmarks
Erik Bernhardsson

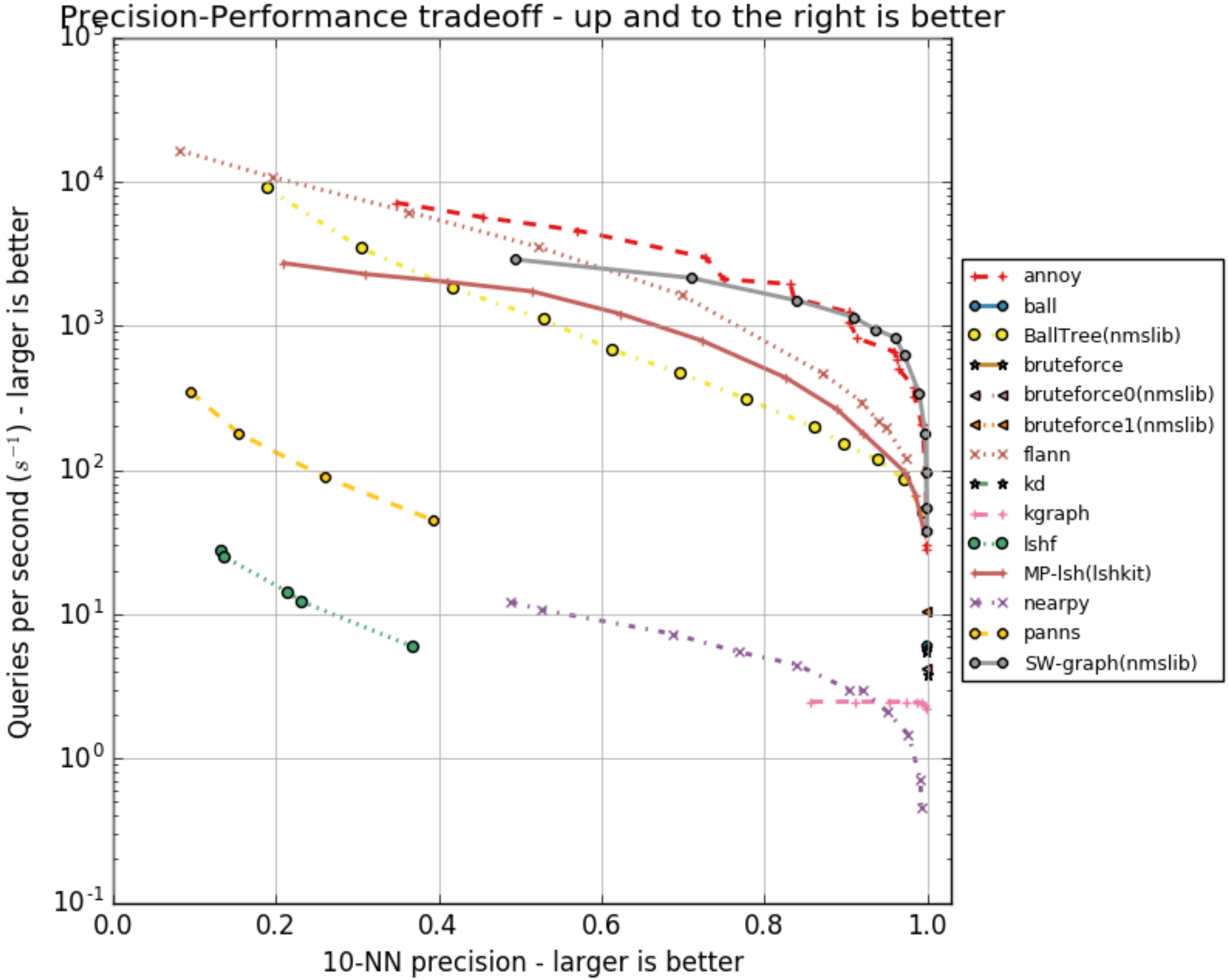
Empirical State of the Art, 2015

SIFT dataset
(word embeddings)

128-dim
1 mio vectors

Source:
ann-benchmarks
Erik Bernhardsson




Precision-Performance tradeoff - up and to the right is better
Queries per second (s^{-1}) - larger is better
10-NN precision - larger is better
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
0.0
0.2
0.4
0.6
0.8
1.0
annoy
ball
BallTree(nmslib)
bruteforce
bruteforce0(nmslib)
bruteforce1(nmslib)
flann
kd
kgraph
lshf
MP-lsh(lshkit)
nearpy
panns
SW-graph(nmslib)

Empirical State of the Art, 2015

Annoy (Approximate Nearest Neighbors Oh Yeah)

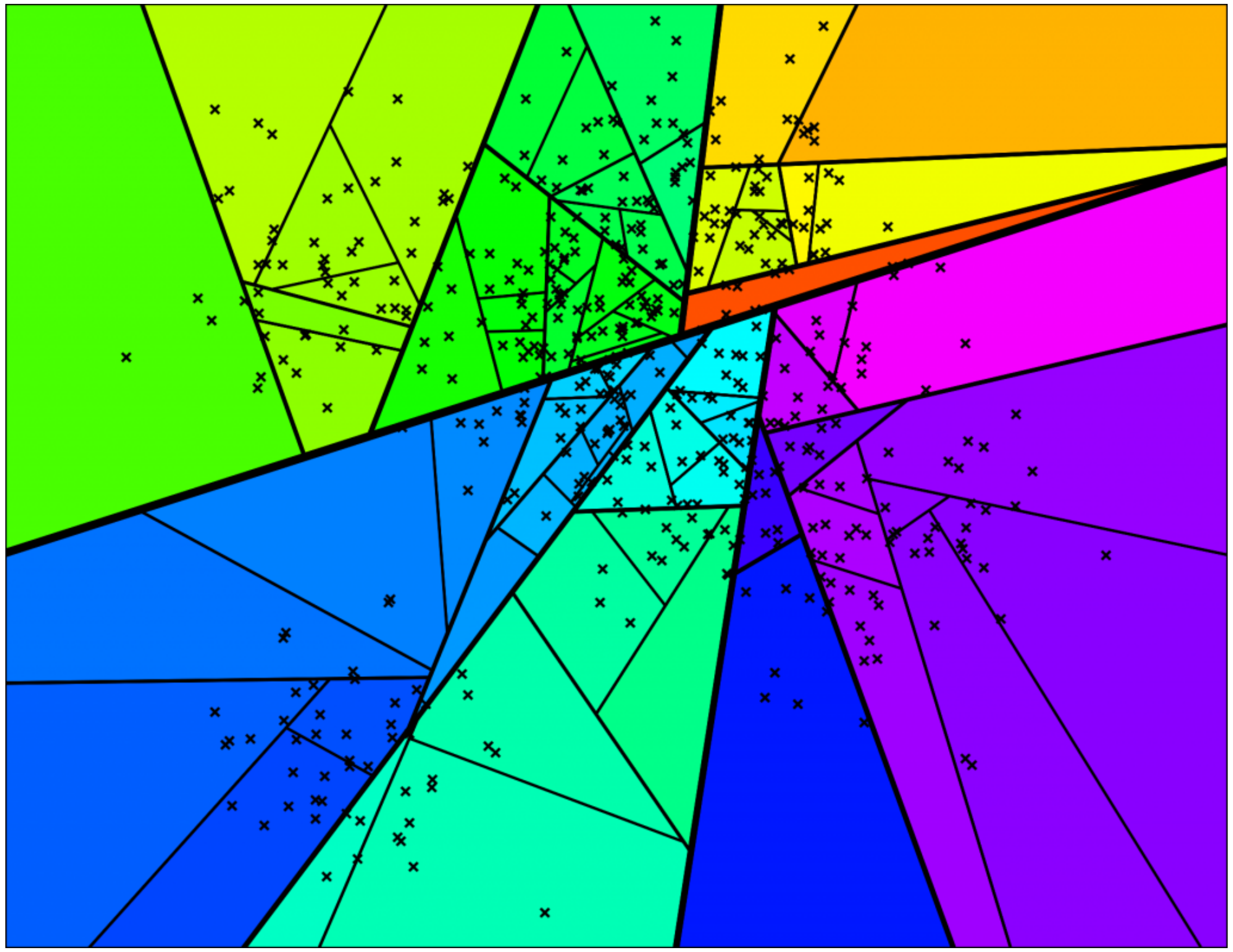
“Annoy was built by Erik Bernhardsson in a couple of afternoons during Hack Week.” (at Spotify in 2013)

Empirical State of the Art, 2015

Annoy (Approximate Nearest Neighbors Oh Yeah)

"Annoy was built by Erik Bernhardsson in a couple of afternoons during Hack Week." (at Spotify in 2013)

Algorithm: Hybrid between hyperplane hash and kd-tree.



Progress Since Hyperplane

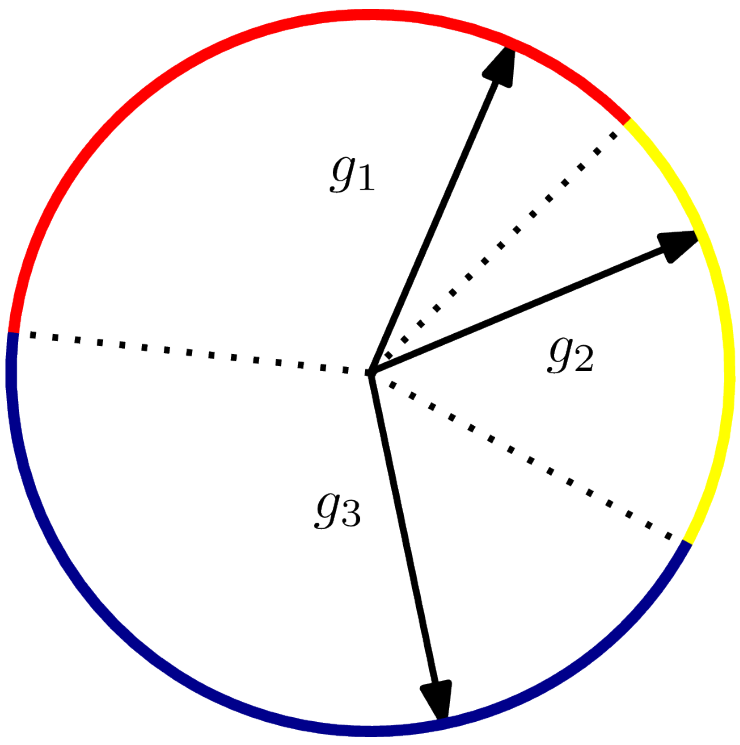
Query time is $O(n^\rho)$, c is the gap between “near” and “far”.

Algorithm	ρ
Hyperplane hash [Charikar 2002]	$1 / c$
Andoni, Indyk 2006	$1 / c^2$
Andoni, Indyk, Nguyen, Razenshteyn 2014	$7 / 8c^2$
Voronoi hash [Andoni, Razenshteyn 2015]	$1 / 2c^2$

For near = 45°: exponent 0.42 (hyperplane) vs 0.18 [AR’15].

Voronoi Hash

For each hash function, sample T random unit vectors $g_1, g_2, \ldots g_T$.



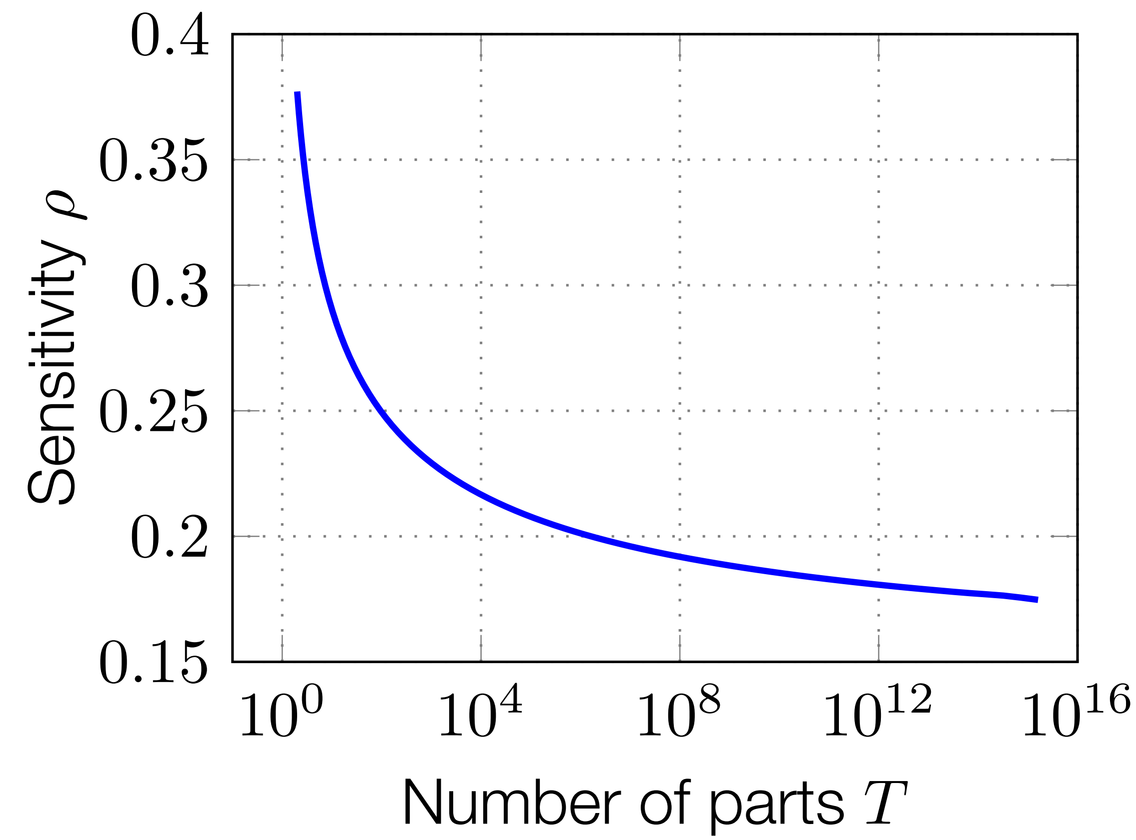
Hash function

To hash a given point p, find the closest g_i.

Cost of Hash Computation

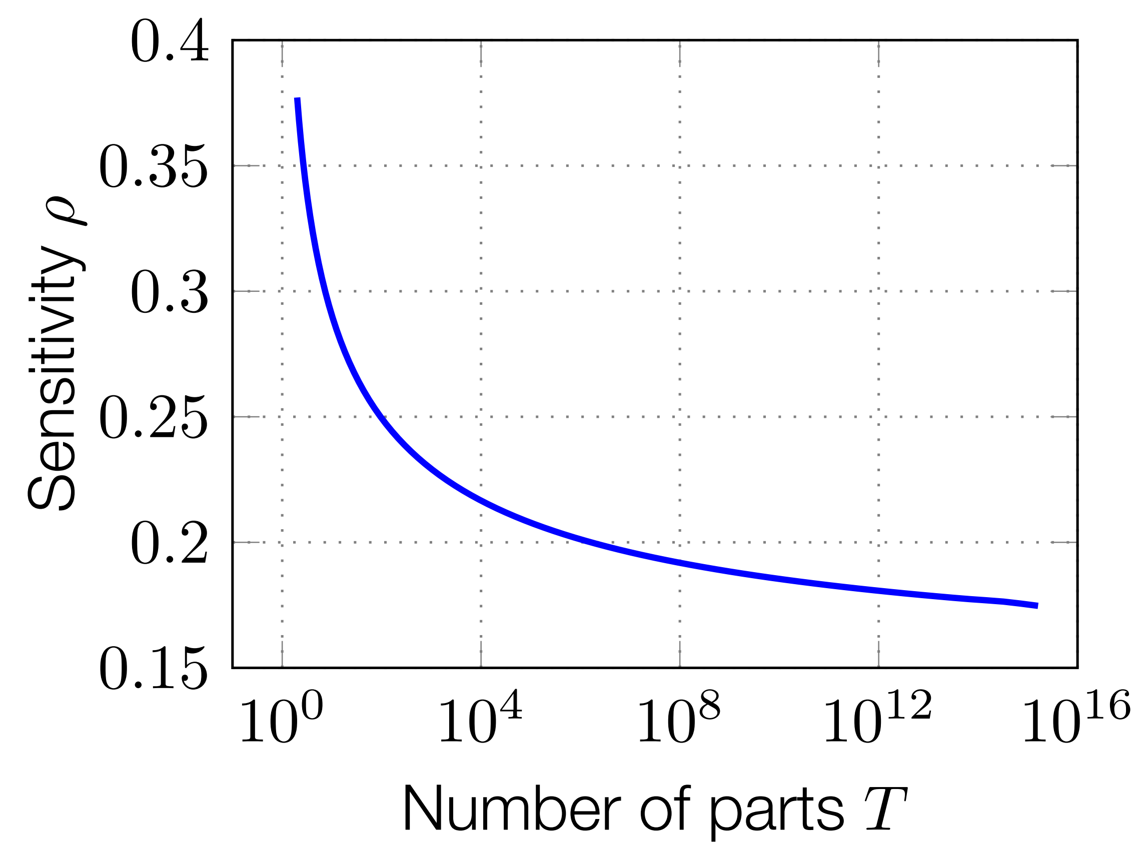
The Voronoi hash requires **many** inner products for good sensitivity ρ.

Time complexity: O(d T)



Cost of Hash Computation

The Voronoi hash requires **many** inner products for good sensitivity ρ.



Time complexity: O(d T)

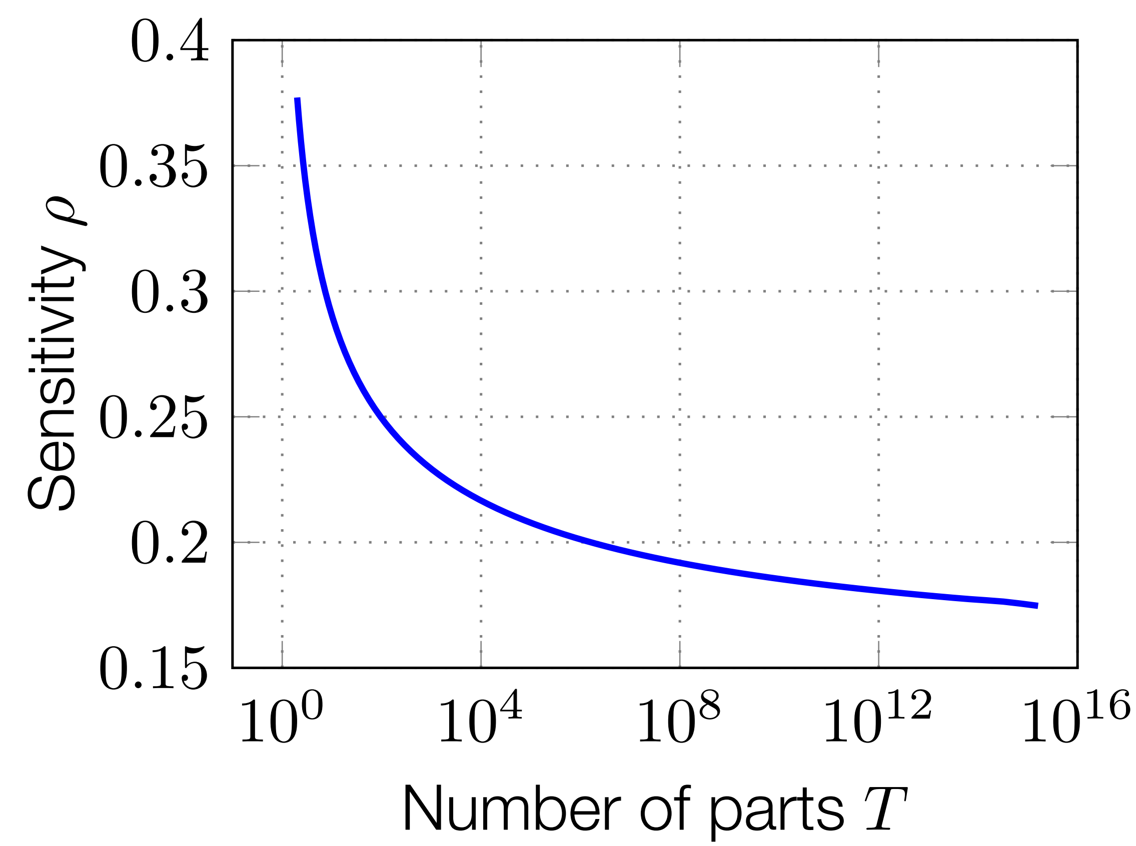
Hyperplane hash works with a **single** inner product.

→ O(d) time

Cost of Hash Computation

The Voronoi hash requires **many** inner products for good sensitivity ρ.

Time complexity: O(d T)



Hyperplane hash works with a **single** inner product.

O(d) time

Can we get fast hash functions with good sensitivity?

1. Locality-Sensitive Hashing
2. Cross-Polytope Hash
3. LSH in Neural Networks

Cross-Polytope Hash

Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere

Kengo Terasawa and Yuzuru Tanaka

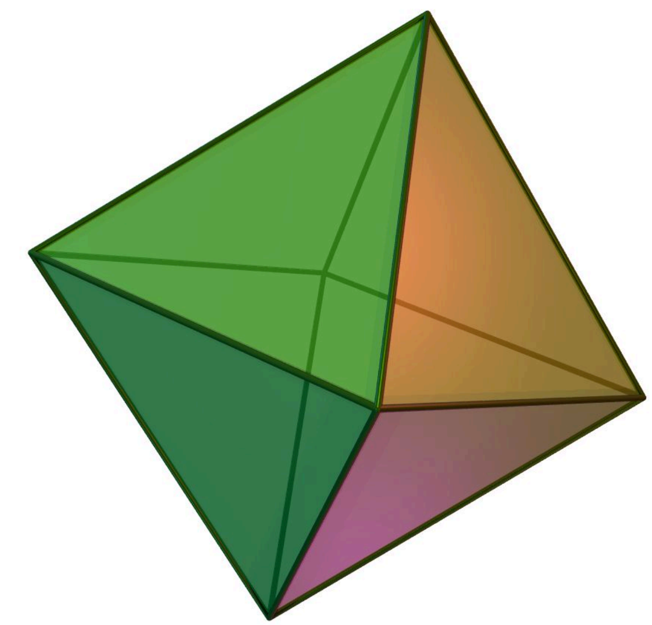
Meme Media Laboratory, Hokkaido University,
N-13, W-8, Sapporo, 060–8628, Japan
{terasawa,tanaka}@meme.hokudai.ac.jp

Abstract. LSH (Locality Sensitive Hashing) is one of the best known methods for solving the c-approximate nearest neighbor problem in high dimensional spaces. This paper presents a variant of the LSH algorithm, focusing on the special case of where all points in the dataset lie on the surface of the unit hypersphere in a d-dimensional Euclidean space. The LSH scheme is based on a family of hash functions that preserves locality of points. This paper points out that when all points are constrained to lie on the surface of the unit hypersphere, there exist hash functions that partition the space more efficiently than the previously proposed methods. The design of these hash functions uses randomly rotated regular polytopes and it partitions the surface of the unit hypersphere like a Voronoi diagram. Our new scheme improves the exponent ρ, the main indicator of the performance of the LSH algorithm.

1 Introduction

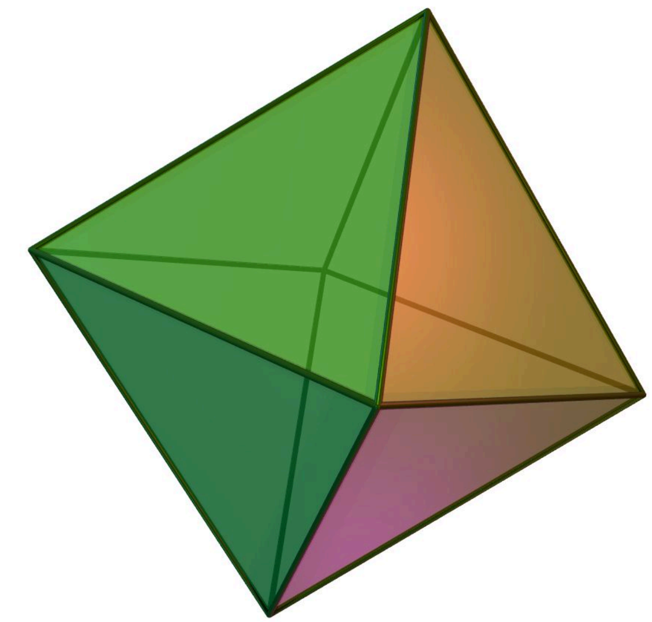
Cross-Polytope Hash

Cross-polytope = l_1 unit ball



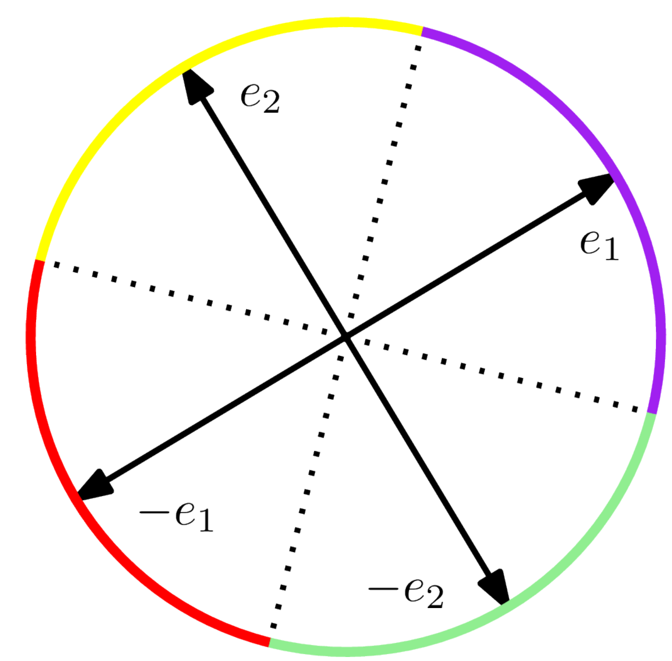
Cross-Polytope Hash

Cross-polytope = l_1 unit ball



Hash function

1. Apply random rotation to input point
2. Map to closest vertex of the cross-polytope



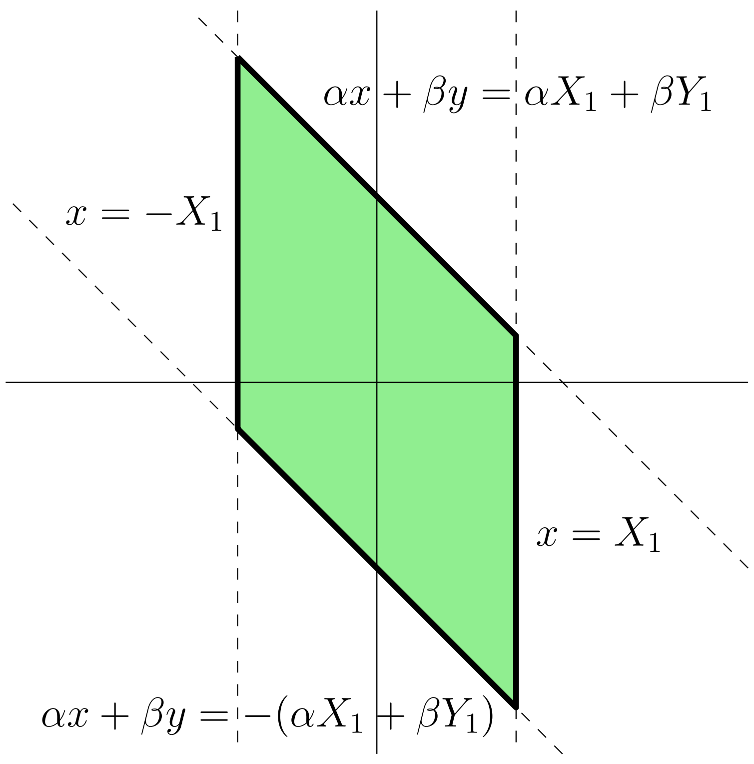
Our Contributions

1. Analyze the CP hash
2. Multiprobe for the CP hash
3. Fast Implementation

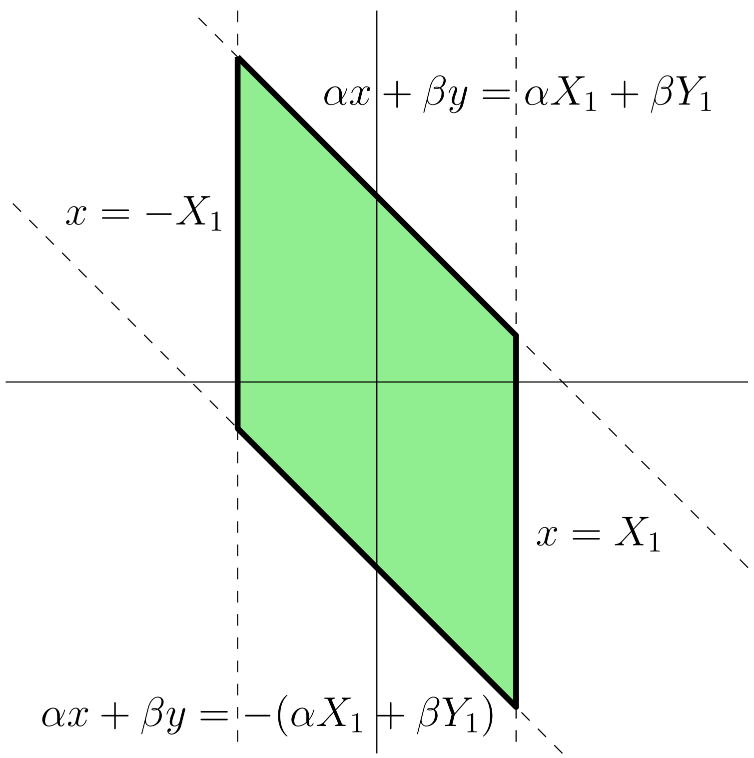
Analysis

With a Gaussian random rotation,
the CP hash has sensitivity $\rho \approx 1 / 2c^2$.

(Caveats: points on the sphere, $r_2 = \sqrt{2}$.)



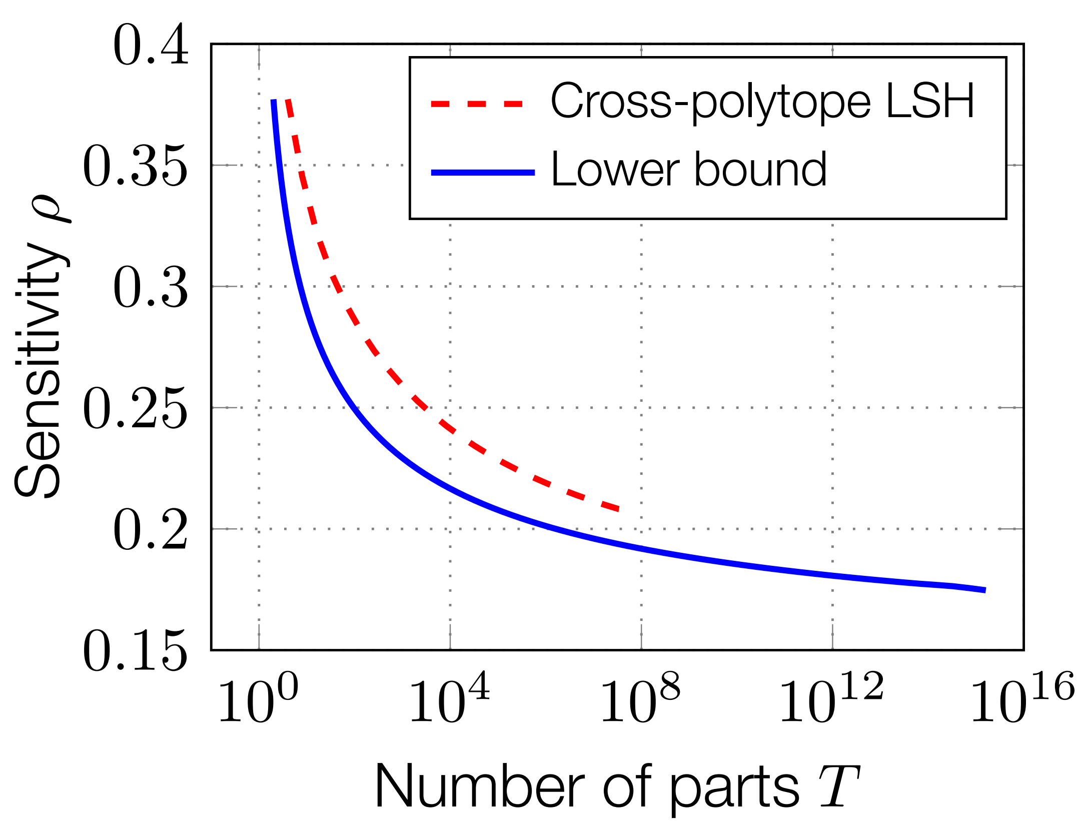
Analysis



With a Gaussian random rotation,
the CP hash has sensitivity **ρ ≈ 1 / 2c²**.

(Caveats: points on the sphere, $r_2 = \sqrt{2}$.)

Establish lower bound for
ρ vs #parts trade-off.



Multiprobe

Problem with LSH in some regimes: requires **many** tables.

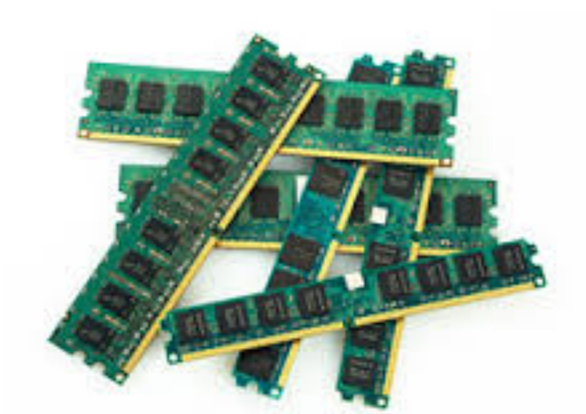
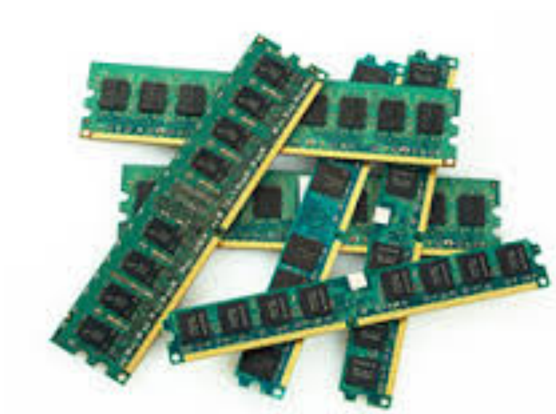
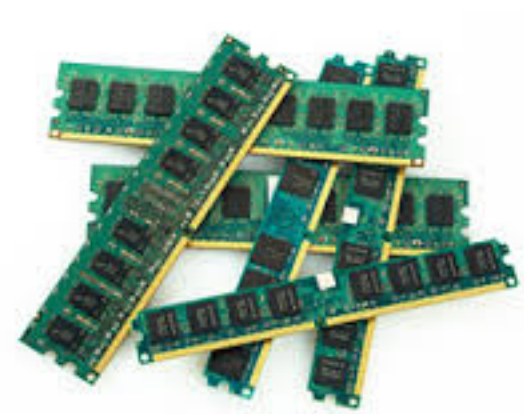
Example: for 10^6 points and queries with 45°, Hyperplane LSH needs **725 tables** for success probability 90%.

Multiprobe

Problem with LSH in some regimes: requires **many** tables.

Example: for 10^6 points and queries with 45°, Hyperplane LSH needs **725 tables** for success probability 90%.

→ More memory than the dataset itself.


Multiprobe

Problem with LSH in some regimes: requires **many** tables.

Idea: use multiple hash locations in the same few tables.
[Lv, Josephson, Wang, Charikar, Li 2007]

Multiprobe

Problem with LSH in some regimes: requires **many** tables.

Idea: use multiple hash locations in the same few tables.
[Lv, Josephson, Wang, Charikar, Li 2007]

We develop a multiprobe scheme for the CP hash

How to score hash buckets?

Fast Implementation

Algorithmic side: use fast pseudo-random rotations.

Similar to fast JL [Ailon, Chazelle 2009].

Overall O(d log d) time for one hash function.

Fast Implementation

Algorithmic side: use fast pseudo-random rotations.

Similar to fast JL [Ailon, Chazelle 2009].

Overall O(d log d) time for one hash function.

➡ Get 2d hash cells in O(d log d) time.

Analysis: [Kennedy, Ward 2016]
[Choromanski, Fagan, Gouy-Pailler, Morvan, Sarlós, Atif 2016]

Fast Implementation

Algorithmic side: use fast pseudo-random rotations.

Similar to fast JL [Ailon, Chazelle 2009].

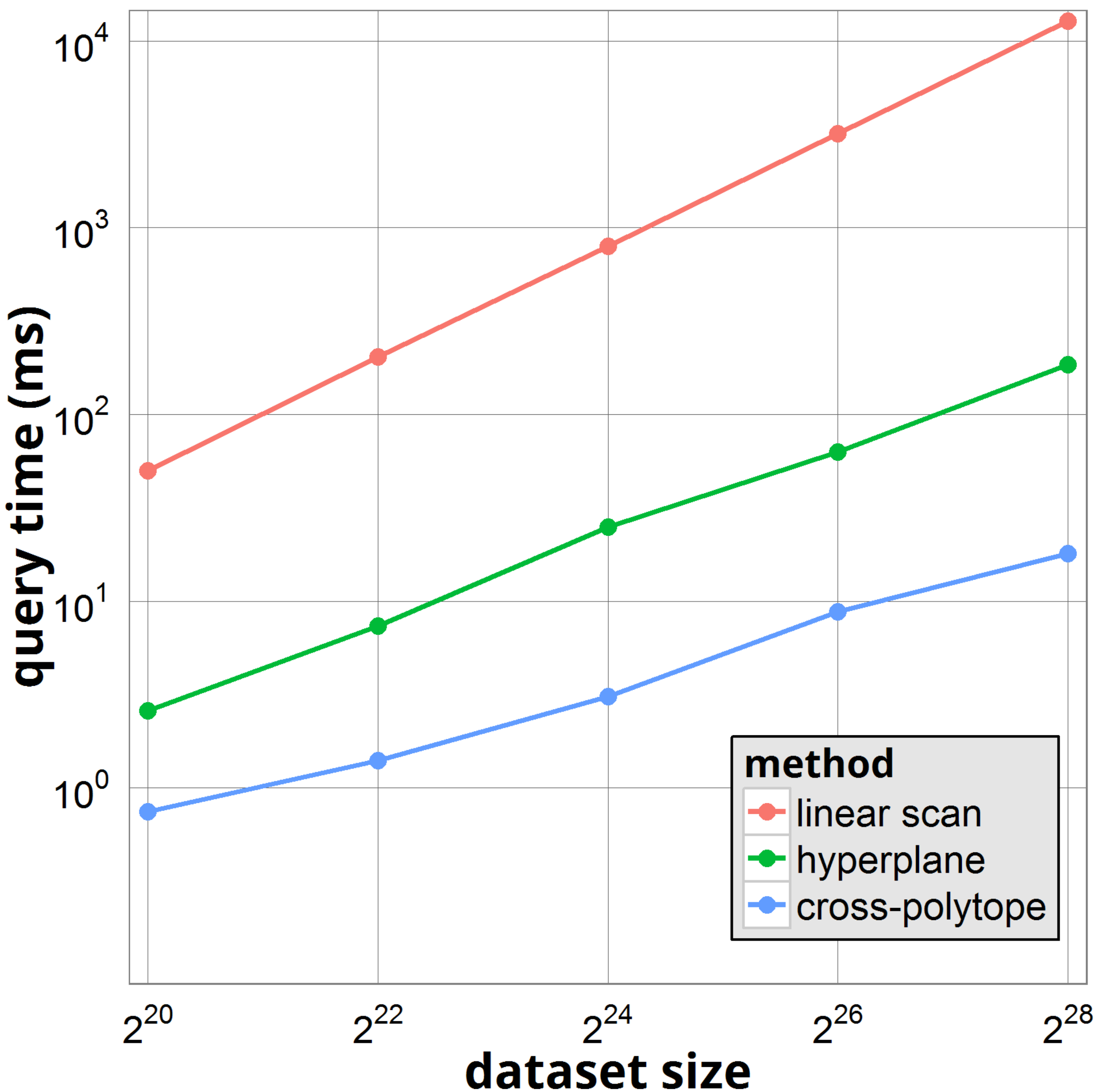
Overall O(d log d) time for one hash function.

→ Get 2d hash cells in O(d log d) time.

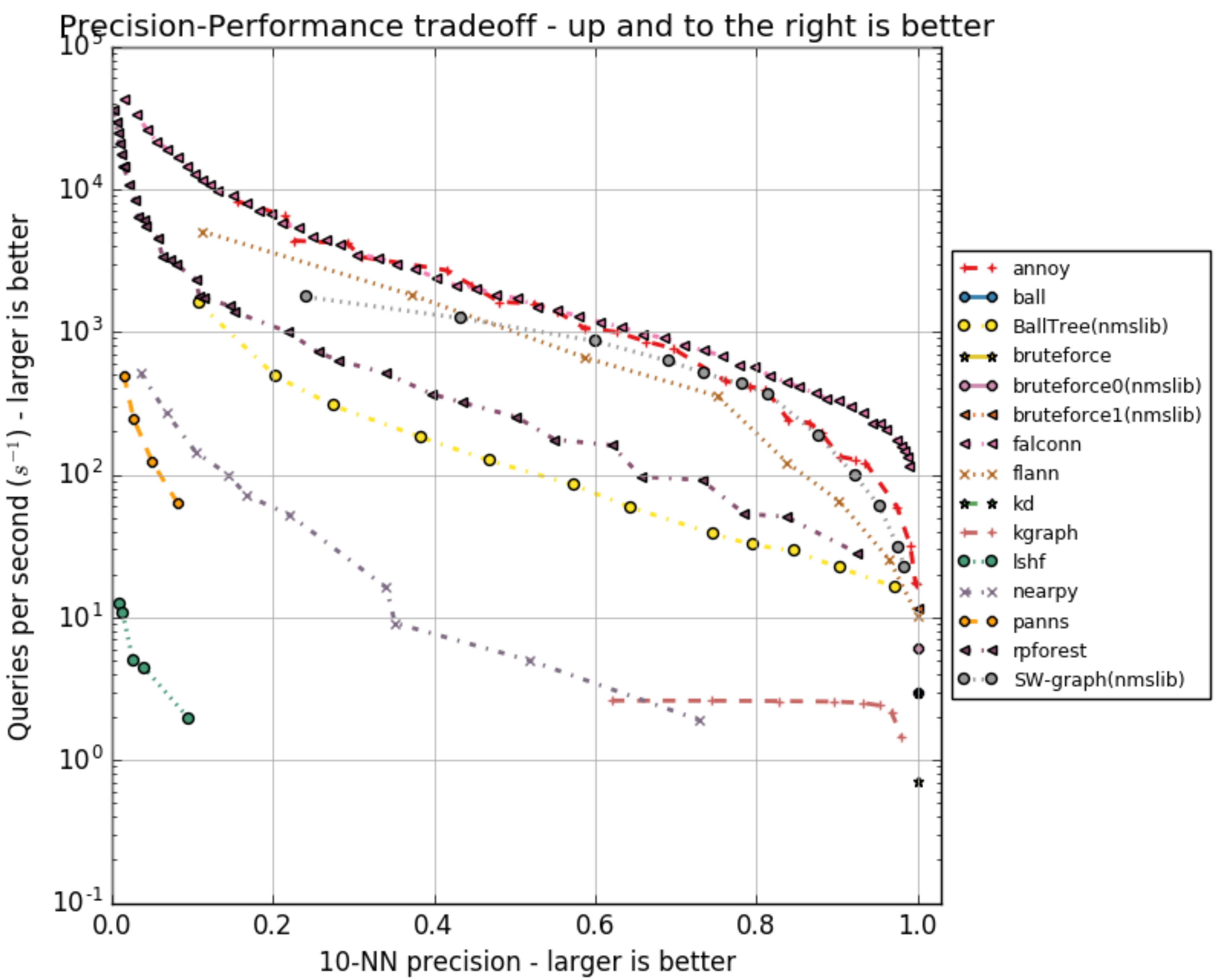
Analysis: [Kennedy, Ward 2016]
[Choromanski, Fagan, Gouy-Pailler, Morvan, Sarlós, Atif 2016]

Implementation side: C++, vectorized code (AVX), etc.

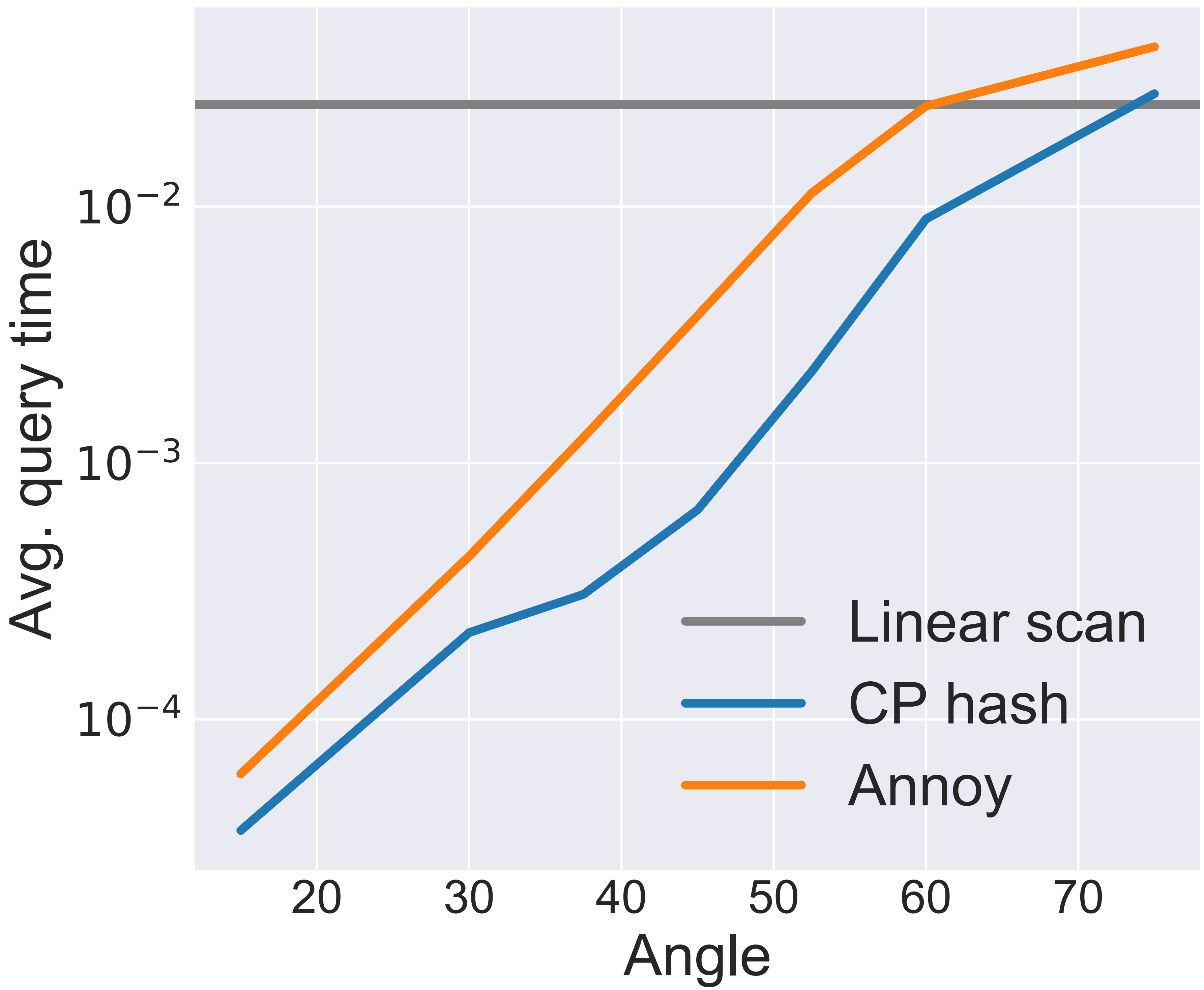
Experiments vs Hyperplane



Experiments on GloVe

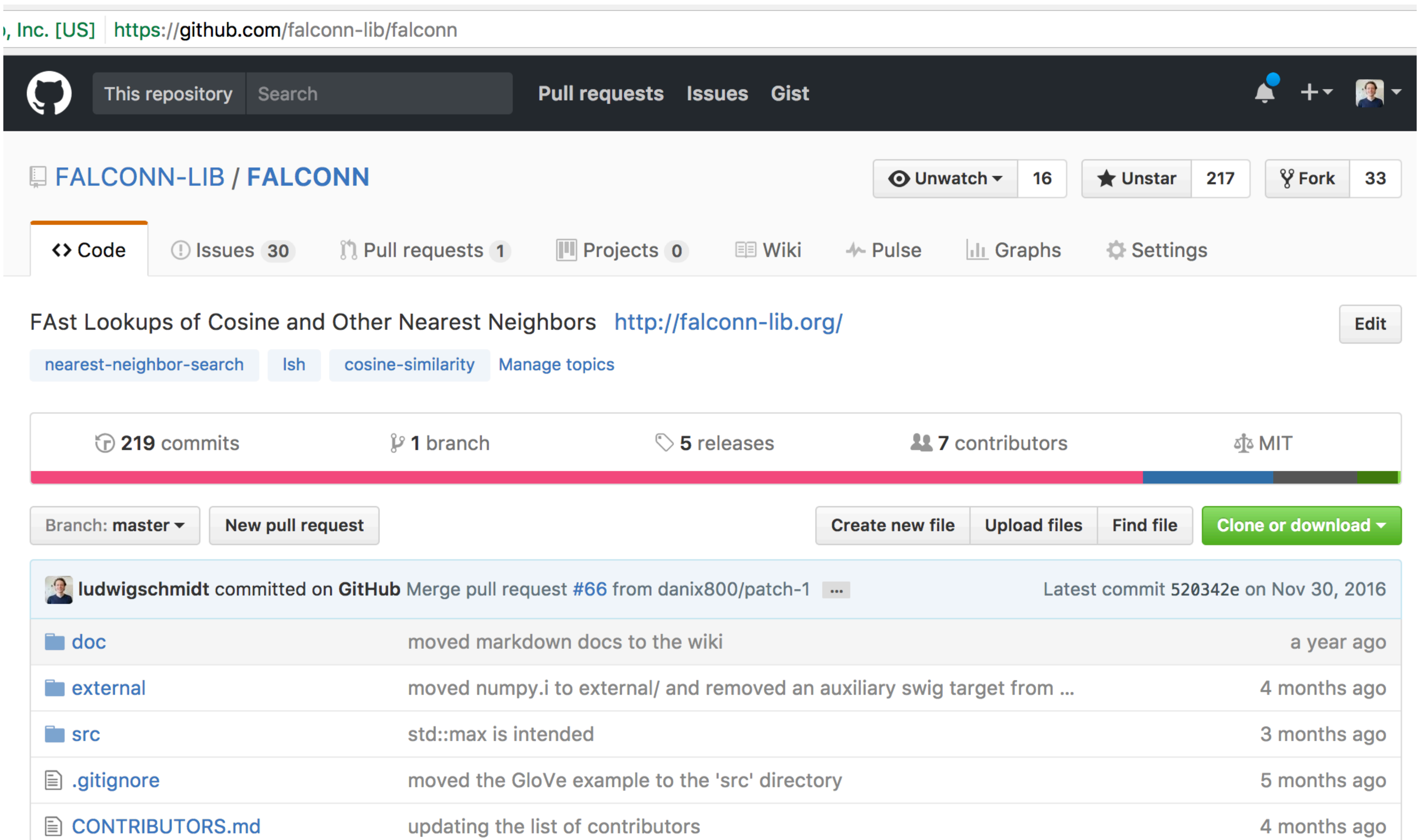


Experiments vs Annoy

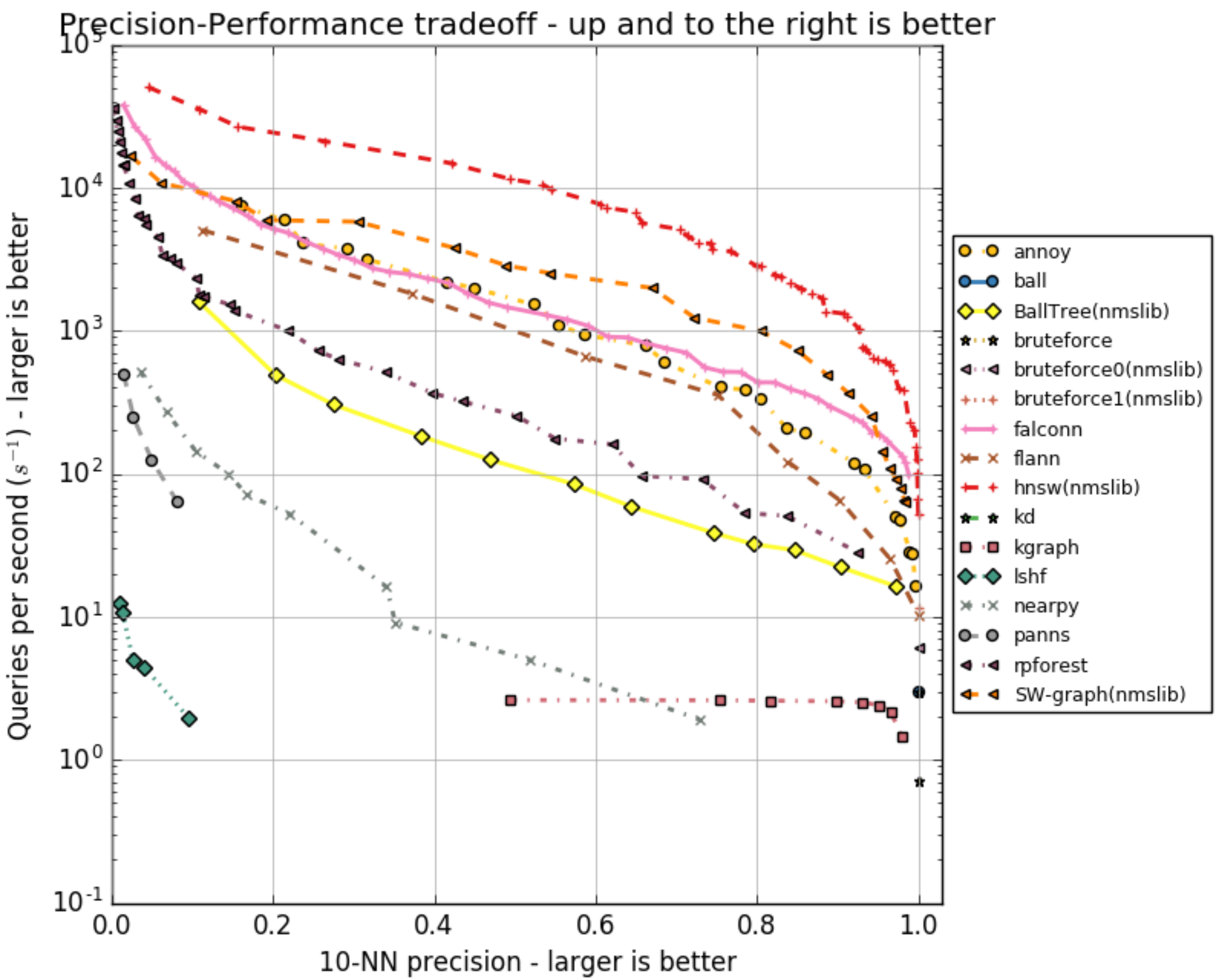


Library: FALCONN

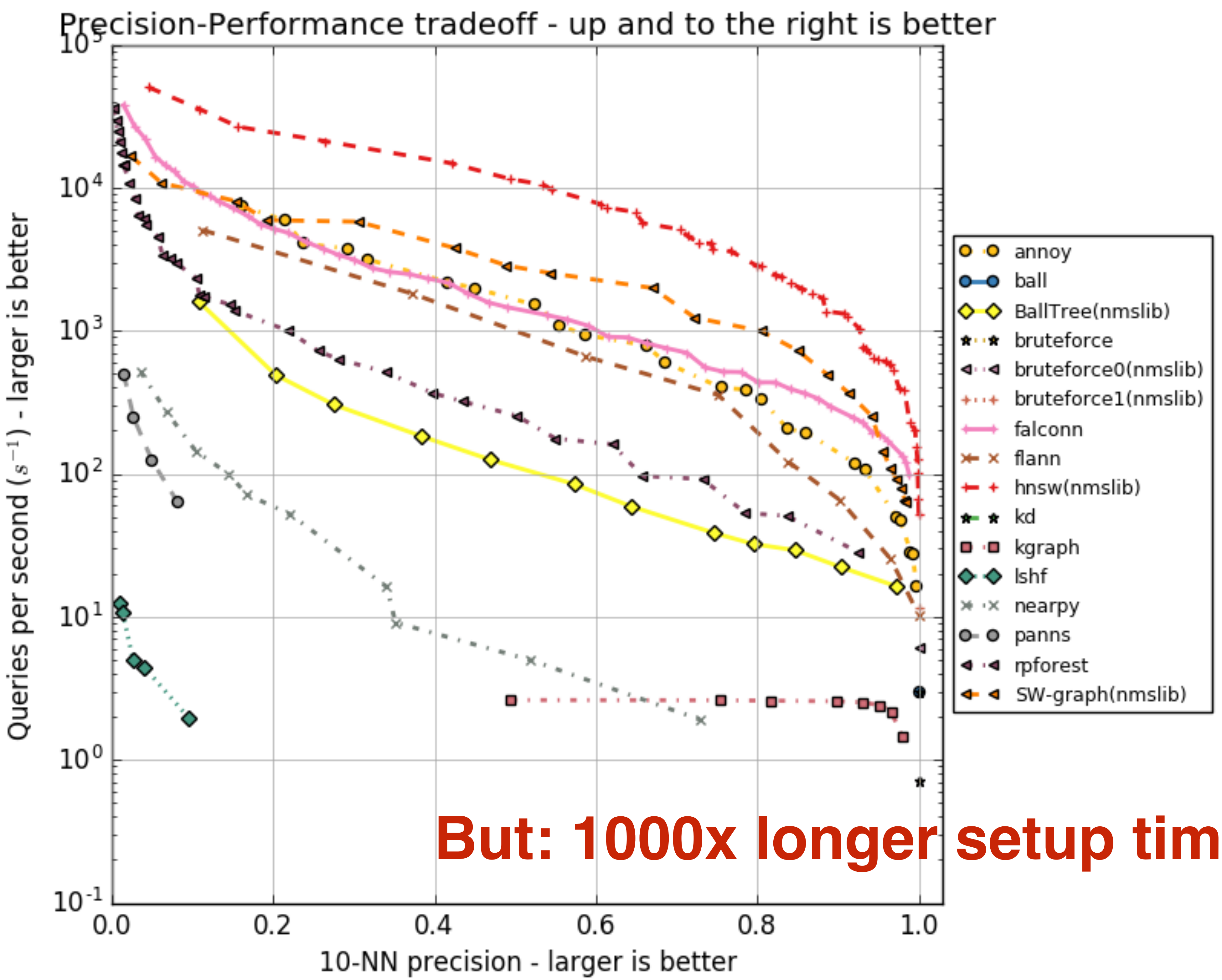
Fast Approximate Look-up of COsine Nearest Neighbors



Since then: Graph Search



Since then: Graph Search



But: 1000x longer setup time

1. Locality-Sensitive Hashing

2. Cross-Polytope Hash

3. LSH in Neural Networks

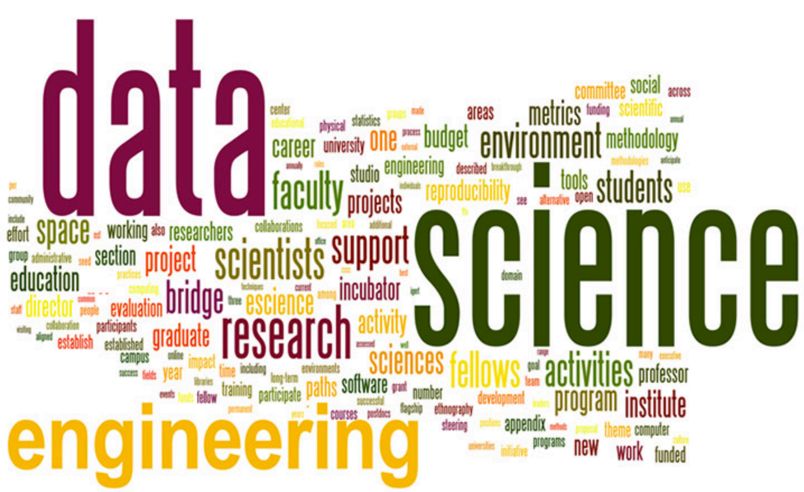
Why Nearest Neighbors and Neural Networks?

Feature vectors for images, audio, text, etc. are now often generated by deep neural networks.

Why Nearest Neighbors and Neural Networks?

Feature vectors for images, audio, text, etc. are now often generated by deep neural networks.

Neural networks are often used with **many output classes**.



Language models

YouTube

Recommender systems

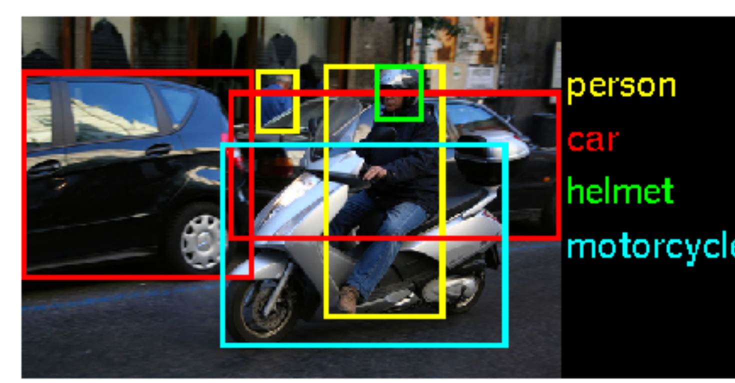
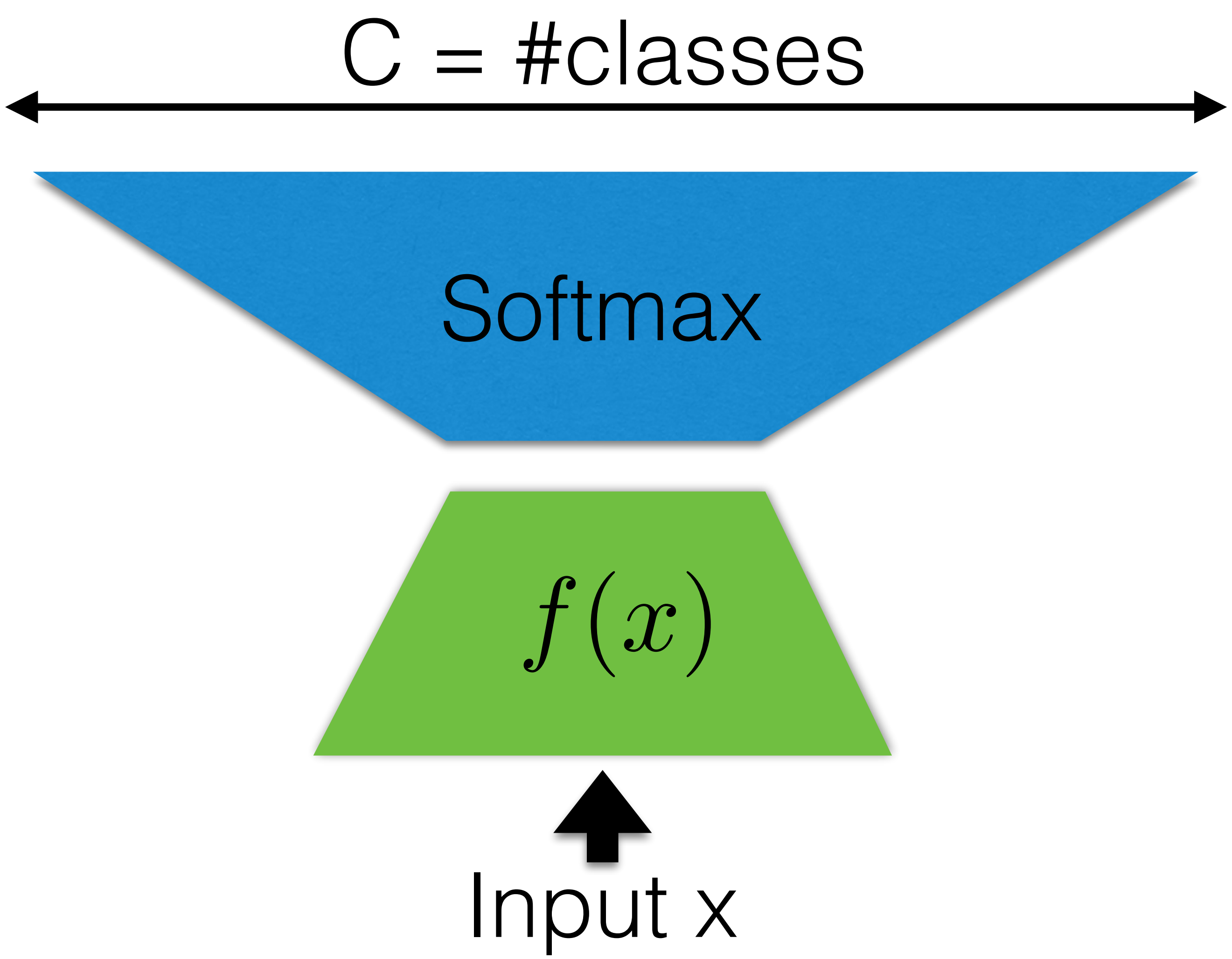


Image annotation

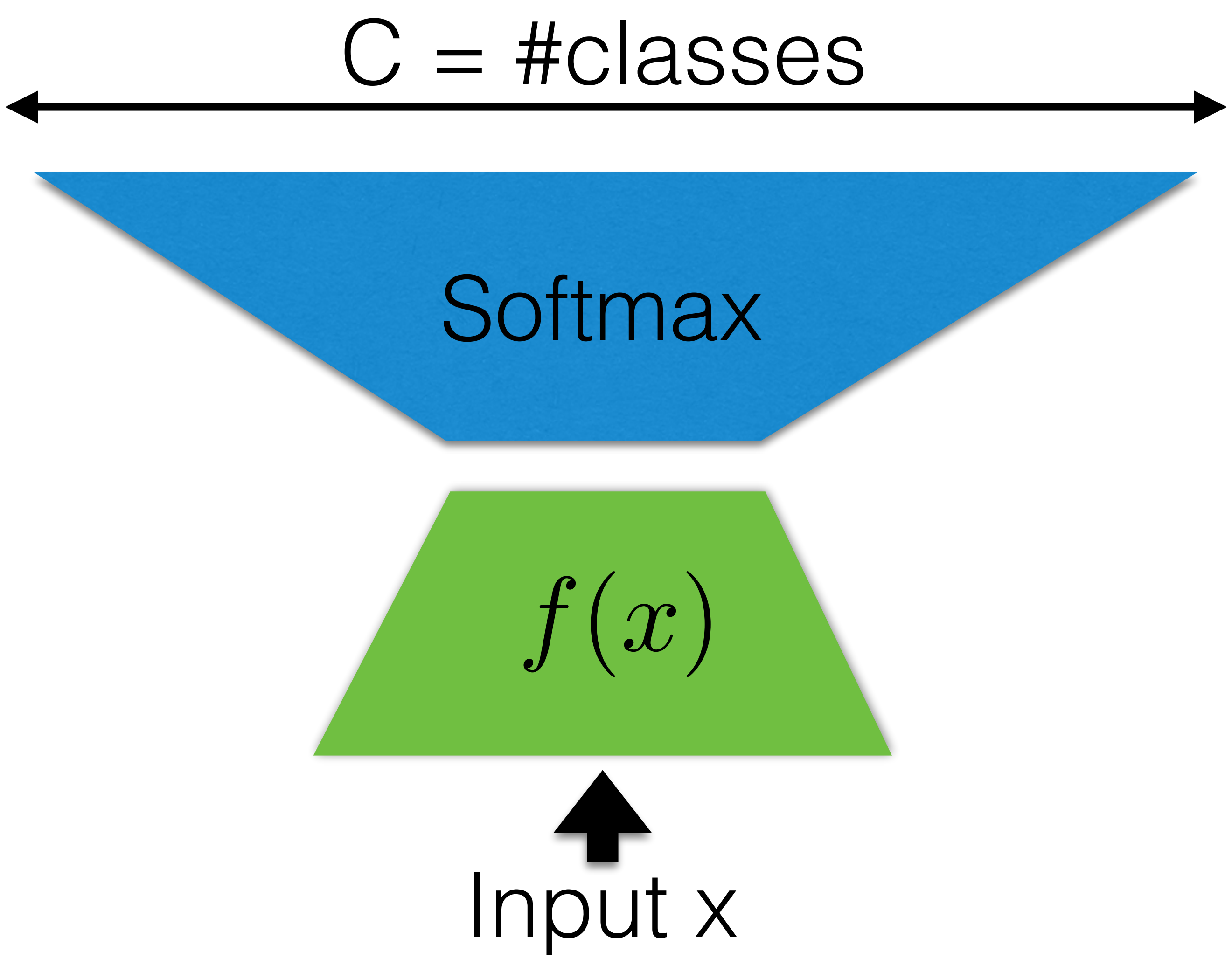
Neural Networks
with Many Output Classes



Top layer (fully connected)
one vector c_i per class.

Embedding computed
by lower layers.

Neural Networks with Many Output Classes



Top layer (fully connected) one vector c_i per class.

Embedding computed by lower layers.

Softmax function: $$p(\text{class } i \mid x) = \frac{\exp(f(x)^T c_i)}{\sum_{j=1}^{C} \exp(f(x)^T c_j)}$$

Prediction Problem

Input: **the class vectors c_i**

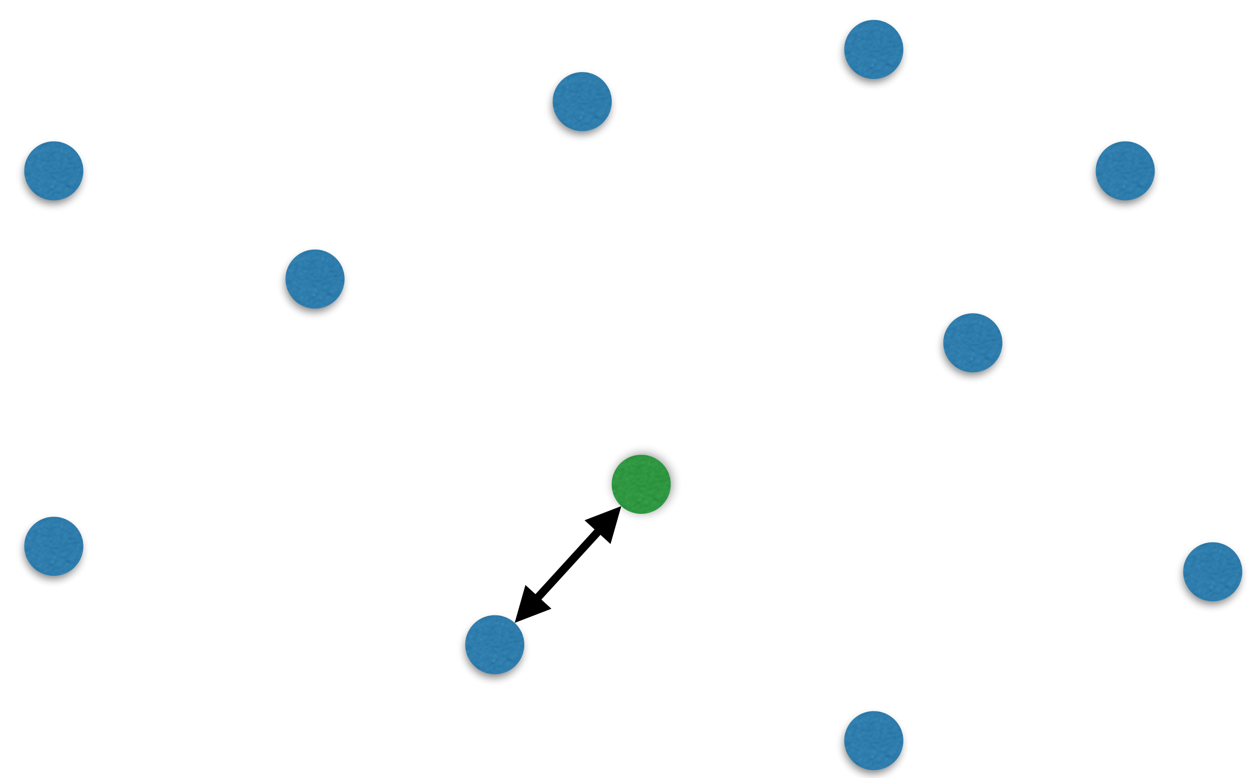
Goal: given a new **embedding f(x)**, quickly find the class vector with maximum inner product.

$$p(\text{class}\, i \mid x) = \frac{\exp(f(x)^T c_i)}{\sum_{j=1}^{C} \exp(f(x)^T c_j)}$$

Prediction Problem

Input: **the class vectors c_i**

Goal: given a new **embedding $f(x)$**, quickly find the class vector with maximum inner product.

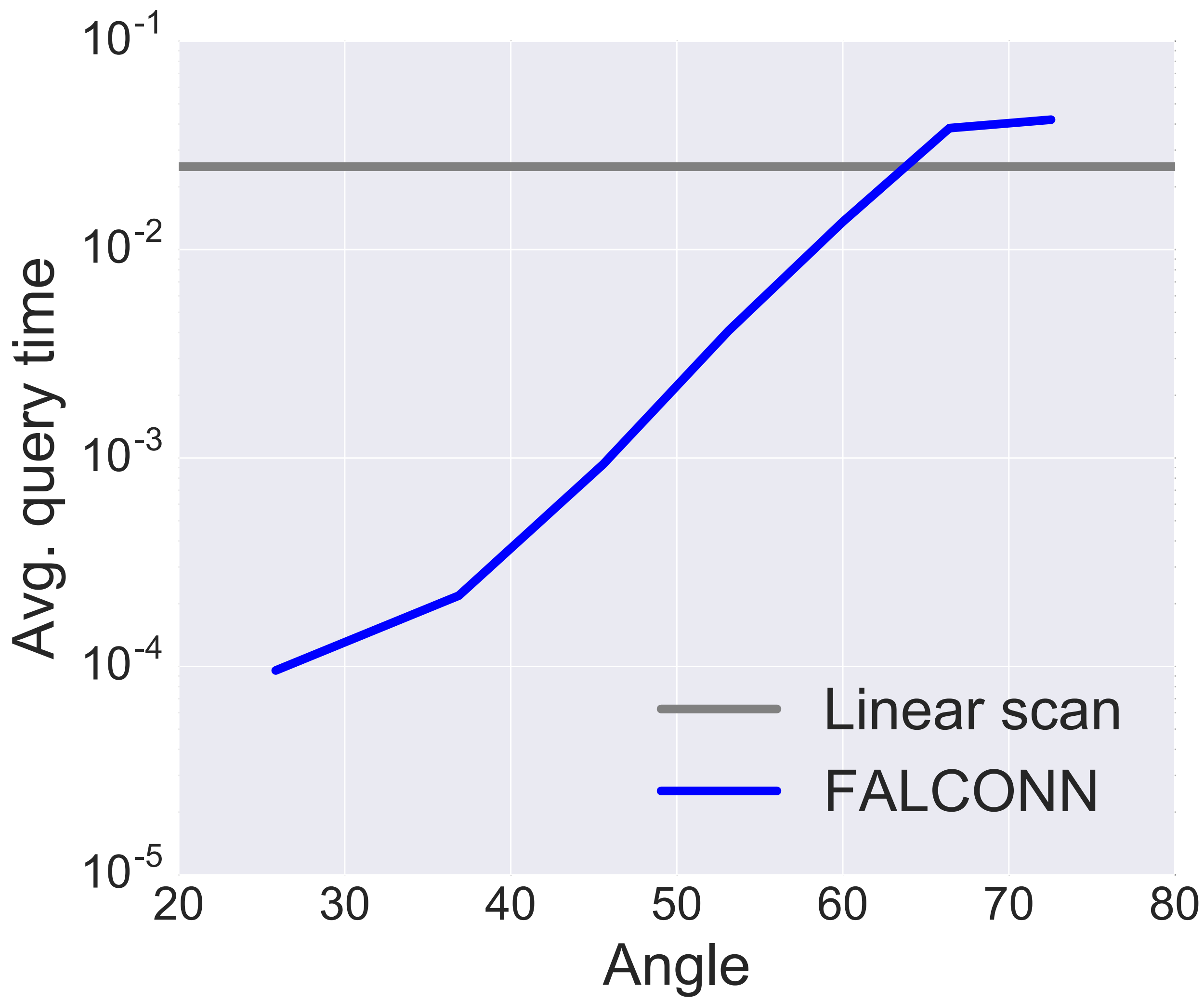


Nearest neighbor under maximum inner product "similarity"

[Vijayanarasimhan, Shlens, Monga, Yagnik 2015], [Spring, Shrivastava 2016]

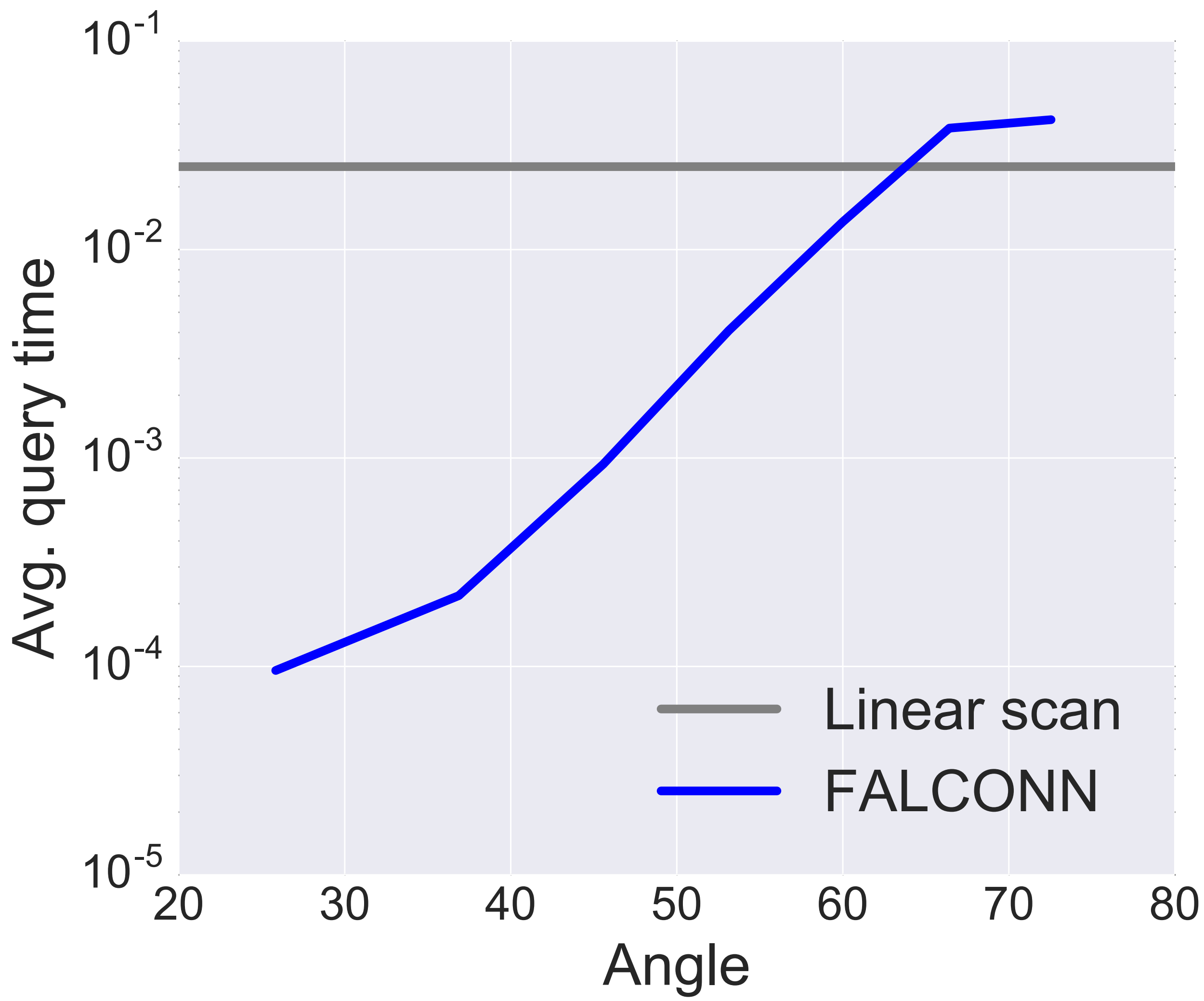
LSH in Neural Networks

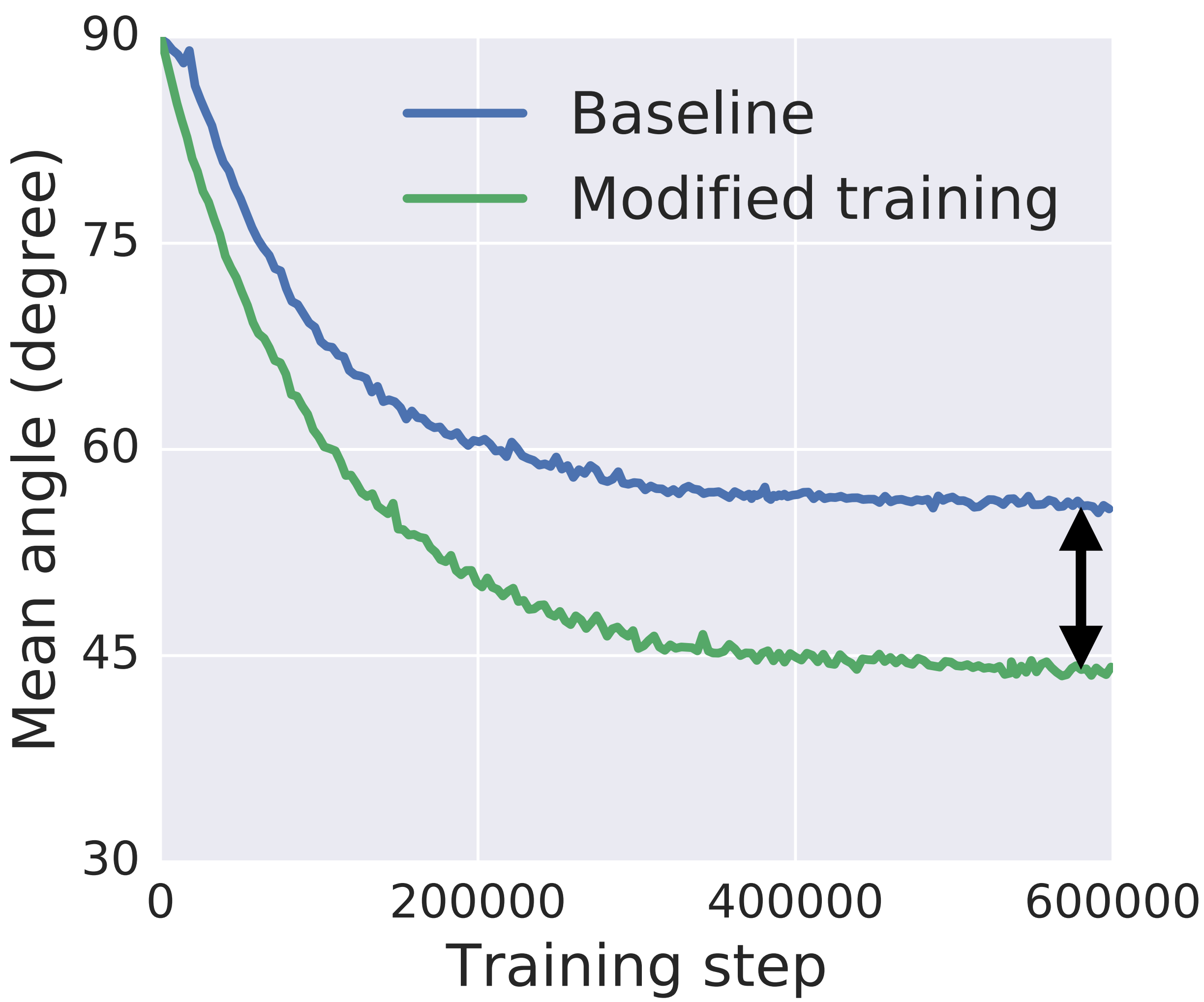
Crucial property: angle to nearest neighbor.



LSH in Neural Networks

Crucial property: angle to nearest neighbor.





→ 10x faster softmax

Training For Larger Angles

Goal of training the network: minimize cross-entropy loss

$$
\begin{aligned}
\text{loss}(x, y) &\approx -f(x)^T c_y \\
&= -\|f(x)\| \cdot \|c_y\| \cdot \cos \sphericalangle(f(x), c_y)
\end{aligned}
$$

Training For Larger Angles

Goal of training the network: minimize cross-entropy loss

$$\begin{aligned} \text{loss}(x, y) &\approx -f(x)^T c_y \\ &= -\|f(x)\| \cdot \|c_y\| \cdot \cos \sphericalangle(f(x), c_y) \end{aligned}$$

Constrain via batch normalization

Training For Larger Angles

Goal of training the network: minimize cross-entropy loss

$$\begin{aligned} \text{loss}(x, y) &\approx -f(x)^T c_y \\ &= -\|f(x)\| \cdot \|c_y\| \cdot \cos \sphericalangle(f(x), c_y) \end{aligned}$$

Constrain via batch normalization

Constrain via projected gradient descent

Training For Larger Angles

Goal of training the network: minimize cross-entropy loss

$$\begin{aligned}\text{loss}(x, y) &\approx -f(x)^T c_y \\ &= -\|f(x)\| \cdot \|c_y\| \cdot \cos \sphericalangle(f(x), c_y)\end{aligned}$$

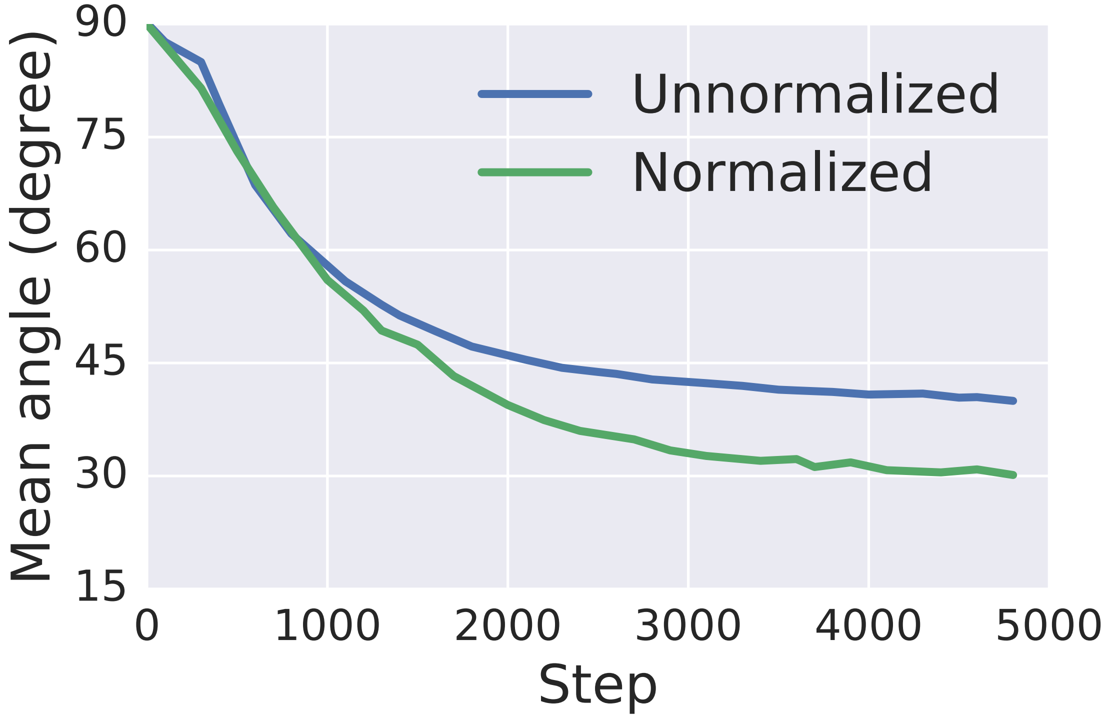
Constrain via batch normalization

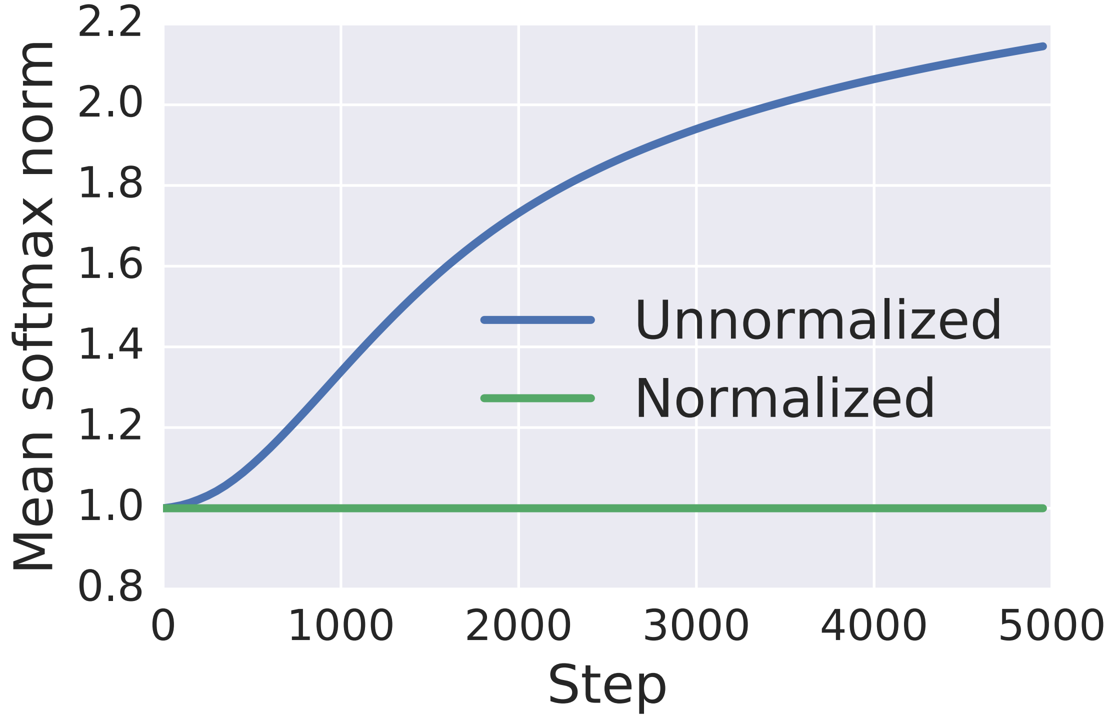
Constrain via projected gradient descent

Result: larger angles

Experiments for Softmax Normalization

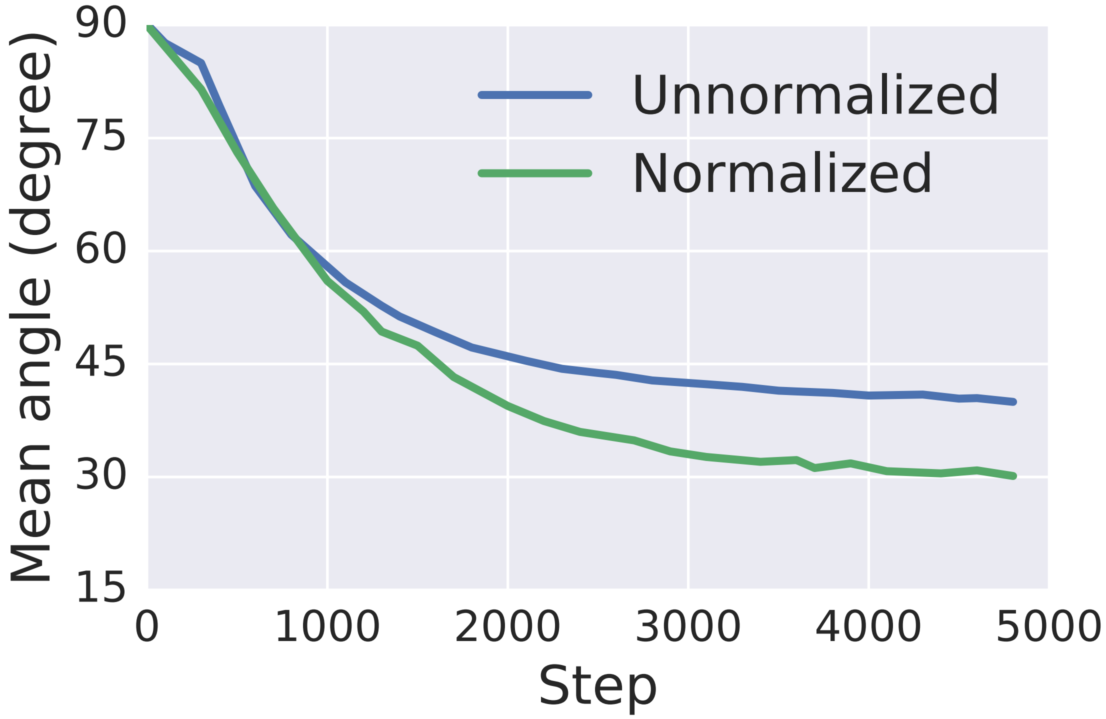
We control the norm of the class vectors c_i via projected gradient descent.

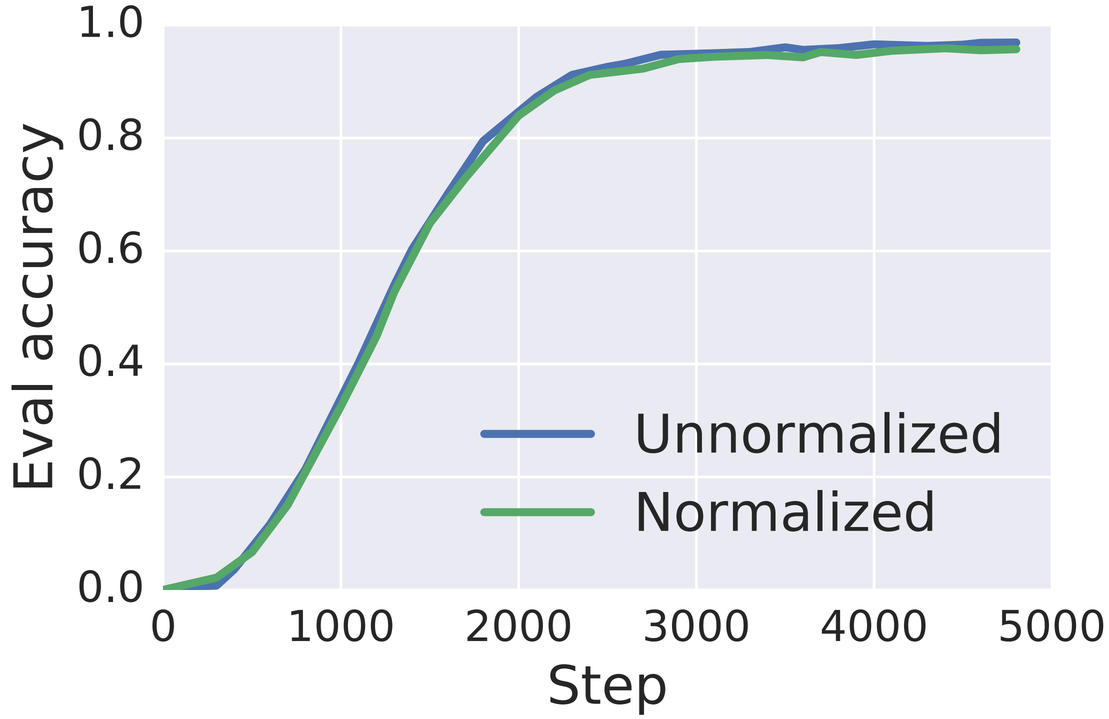




Experiments for Softmax Normalization

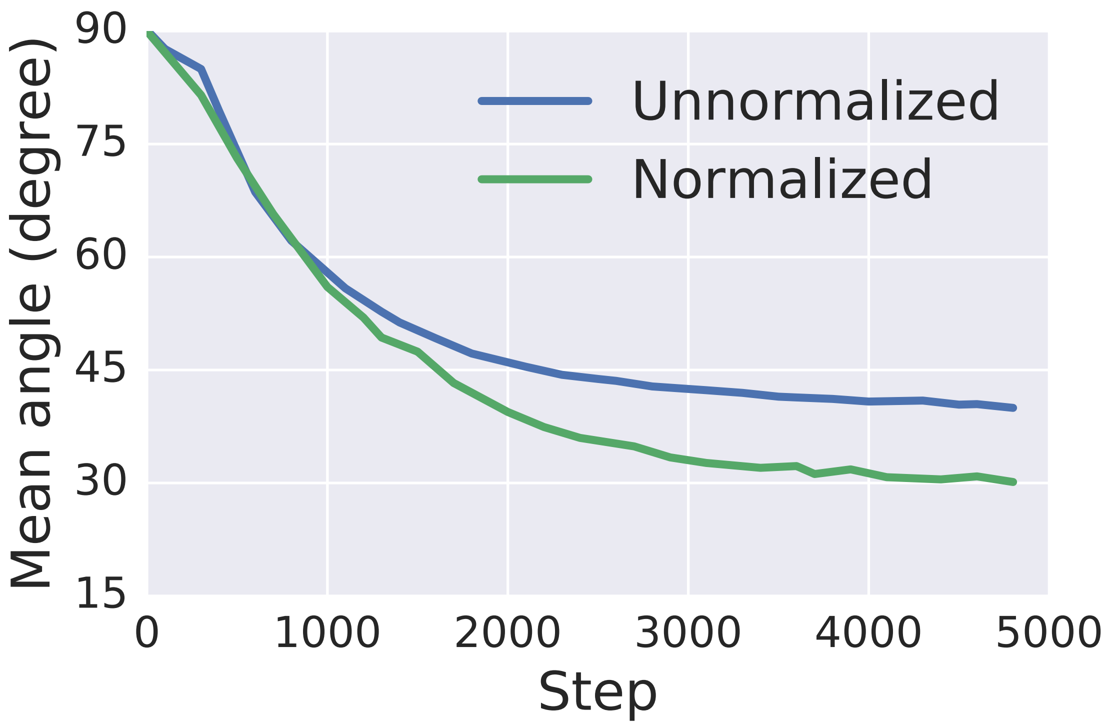
We control the norm of the class vectors c_i via projected gradient descent.

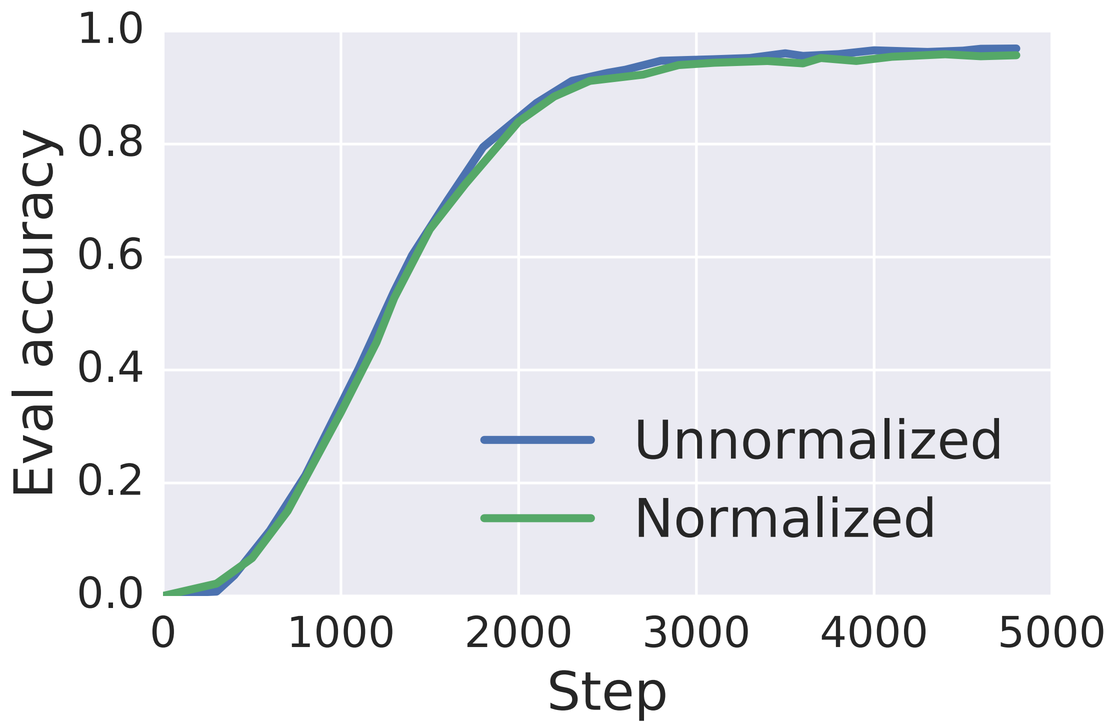




Experiments for Softmax Normalization

We control the norm of the class vectors c_i via projected gradient descent.





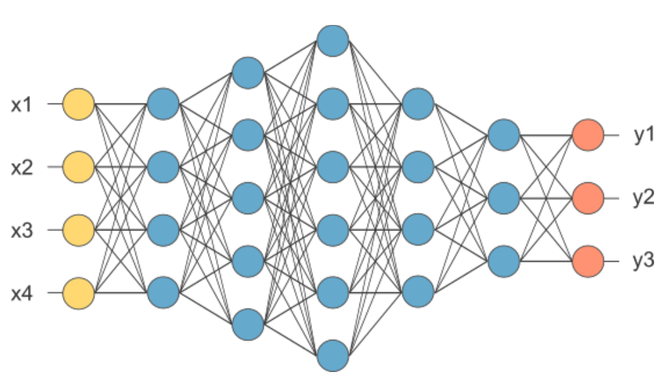
Angular distance instead of maximum inner product.

Conclusions

- Locality-sensitive hashing for nearest neighbor search.
- Cross-polytope hash has good practical performance.
- LSH can be used for fast inference in deep networks.

Conclusions

- Locality-sensitive hashing for nearest neighbor search.
- Cross-polytope hash has good practical performance.
- LSH can be used for fast inference in deep networks.



Thank You!