Approximate Nearest Neighbor (ANN) Search - II

Sanjiv Kumar, Google Research, NY

EECS-6898, Columbia University - Fall, 2010

Two popular ANN approaches

Tree approaches

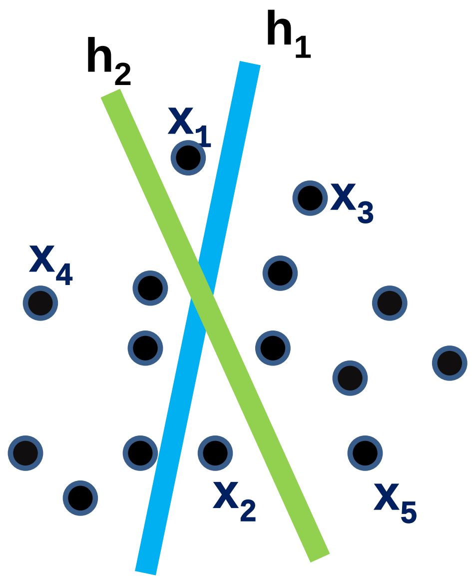
- Recursively partition the data: Divide and Conquer
- Expected query time: O(log n) (with constants exponential in dimension)
- Performance degrades with high-dimensional data
- Large storage needs
- Original data is required at run-time

Hashing approaches

- Each item in database represented as a code
- Significant reduction in storage
  - For 64 bit codes, just 8GB storage instead of 40TB
- Expected query time: O(1) or sublinear in n
  - Search in 64-bit hamming space: ~13 sec instead of ~15 hrs/query
- Compact codes preferred

Example: Binary Codes

Linear projection (hyperplane) based partitioning



X	x_1	x_2	x_3	x_4	x_5
y_1	0	1	1	0	1
y_2	1	0	1	0	1
…	…	…	…	…	…
y_m	…	…	…	…	…
	010…	100…	111…	001…	110…

No recursive partitioning unlike trees!

Hashing: Main Steps

1. Training
   - Define a model to convert an input item in a code
   - Learn the parameters of the model
     - Possibly using a subset of randomly sampled database items

Hashing: Main Steps

1. Training
   - Define a model to convert an input item in a code
   - Learn the parameters of the model
     - Possibly using a subset of randomly sampled database items

Given input x Learn $h(x) = \{h_1(x), h_2(x), \ldots, h_m(x)\}$ $h_k(x) \in Z$

Example: Binary codes using linear projections

$$h_k(x) = \mathrm{sgn}(w_k^T x + b_k) \qquad h_k(x) \in \{-1, 1\}$$

equivalent to $y_k(x) = (1 + h_k(x))/2$ $y_k(x) \in \{0, 1\}$

Hashing: Main Steps

1. Training
   - Define a model to convert an input item in a code
   - Learn the parameters of the model
     - Possibly using a subset of randomly sampled database items

Given input x Learn $h(x) = \{h_1(x), h_2(x), \ldots, h_m(x)\}$ $h_k(x) \in Z$

Example: Binary codes using linear projections

$$h_k(x) = \mathrm{sgn}(w_k^T x + b_k) \qquad h_k(x) \in \{-1, 1\}$$

equivalent to $y_k(x) = (1 + h_k(x))/2$ $y_k(x) \in \{0, 1\}$

Training goal: To learn parameters for m hash functions

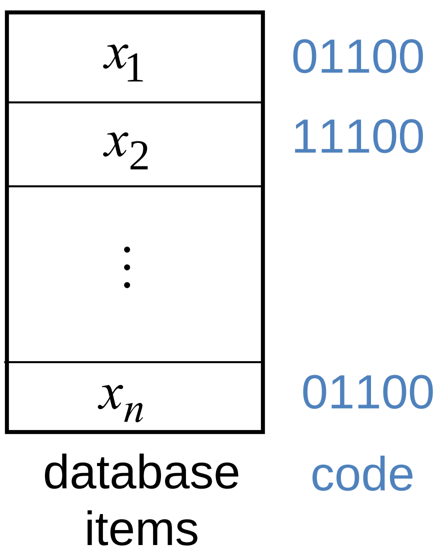
$$\{w_k, b_k\}_{k=1,\ldots,m}$$

Hashing: Main Steps

2. Indexing
   - Represent each item in the database as a code
   - In some cases, organize all the codes in a hash table (inverse-lookup): For a given code, return all the points with the same code

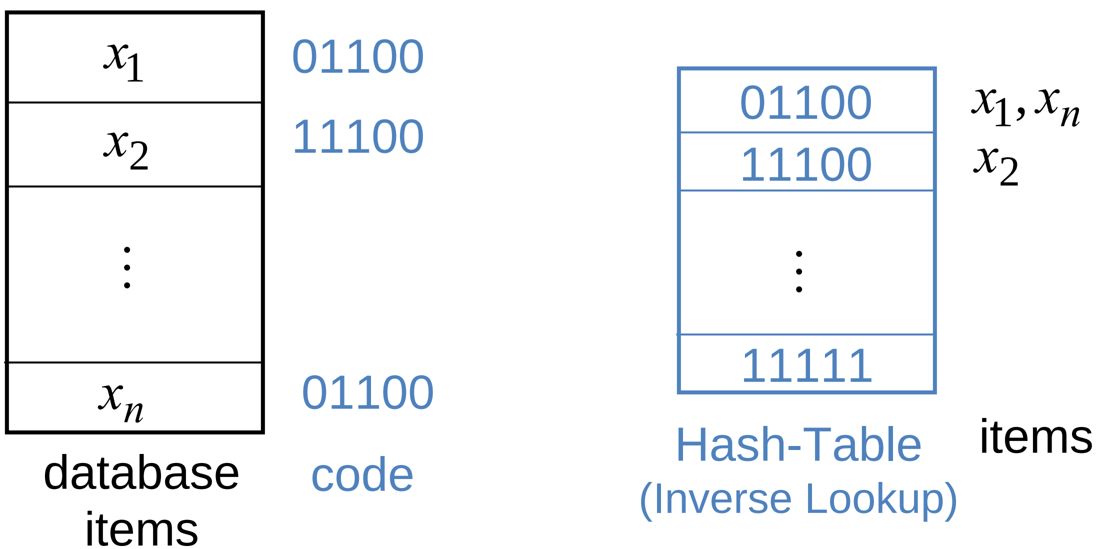
Hashing: Main Steps

2. Indexing
   – Represent each item in the database as a code
   – In some cases, organize all the codes in a hash table (inverse-lookup): For a given code, return all the points with the same code



Hashing: Main Steps

2. Indexing
   - Represent each item in the database as a code
   - In some cases, organize all the codes in a hash table (inverse-lookup): For a given code, return all the points with the same code

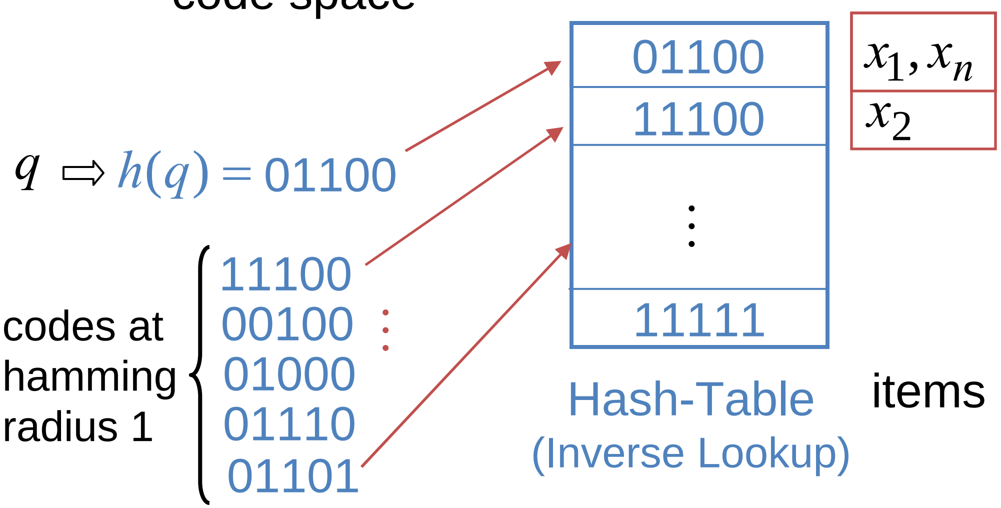


Hashing: Main Steps

3. Querying
   - Convert the query to code
   - Find all items with the same code in database using hash table
     - Return all points within a small radius of query in code space
     - Use multiple codes (and tables) to increase recall
   - Rank all the database items according to their distance from query in code space

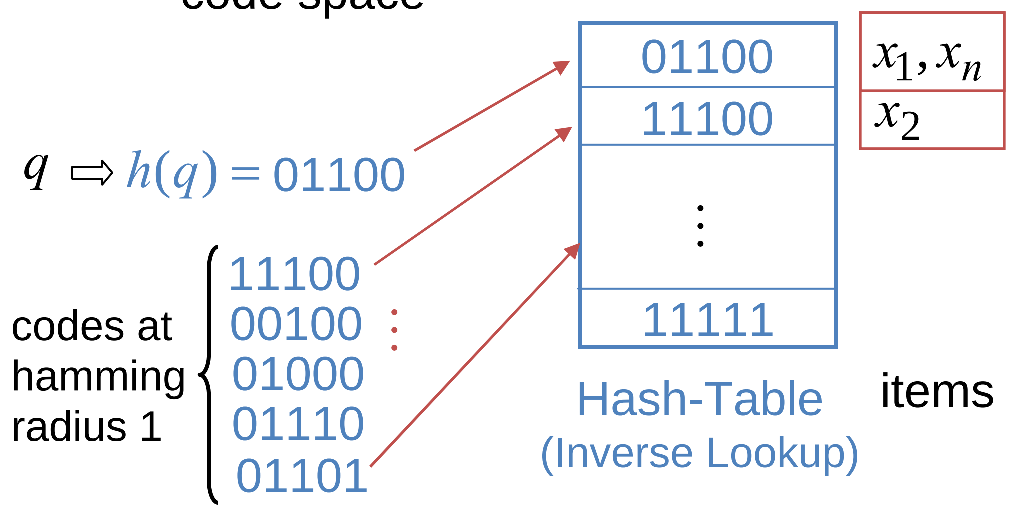
Hashing: Main Steps

3. Querying
   - Convert the query to code
   - Find all items with the same code in database using hash table
     - Return all points within a small radius of query in code space
     - Use multiple codes (and tables) to increase recall
   - Rank all the database items according to their distance from query in code space



Hashing: Main Steps

3. Querying
   - Convert the query to code
   - Find all items with the same code in database using hash table
     - Return all points within a small radius of query in code space
     - Use multiple codes (and tables) to increase recall
   - Rank all the database items according to their distance from query in code space



$O(nm)$ space

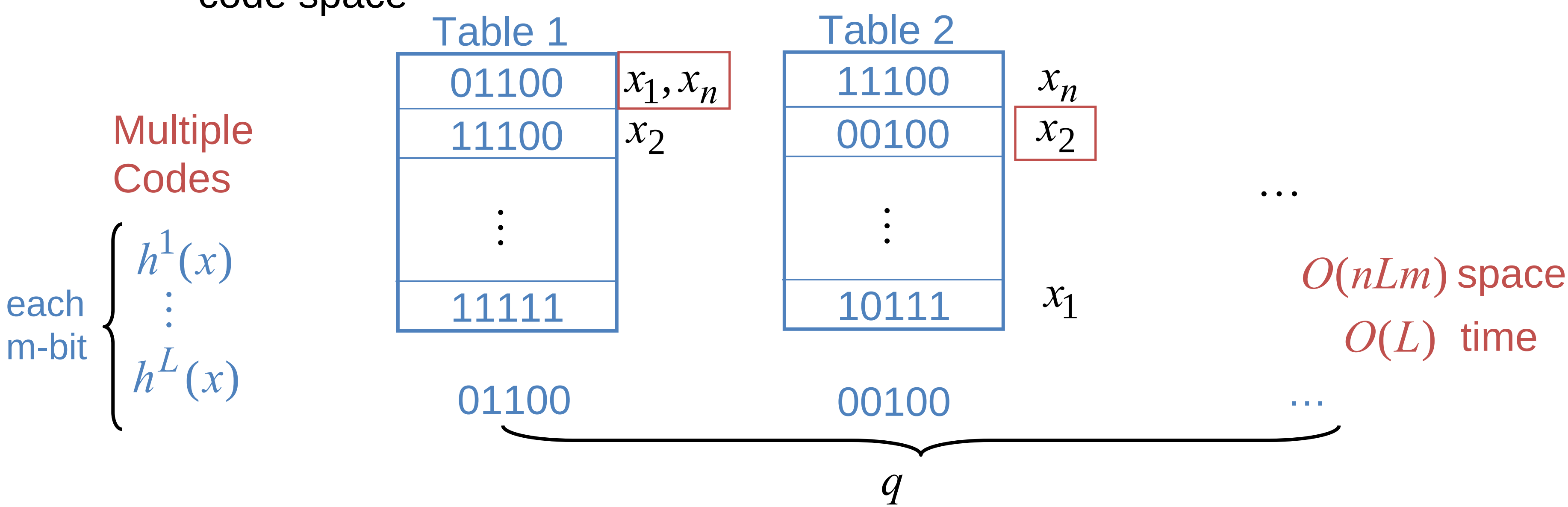
$O(1)$ time

Number of codes to search at radius r : $O(m^r)$

Buckets for many codes may be empty: with high probability for large m !

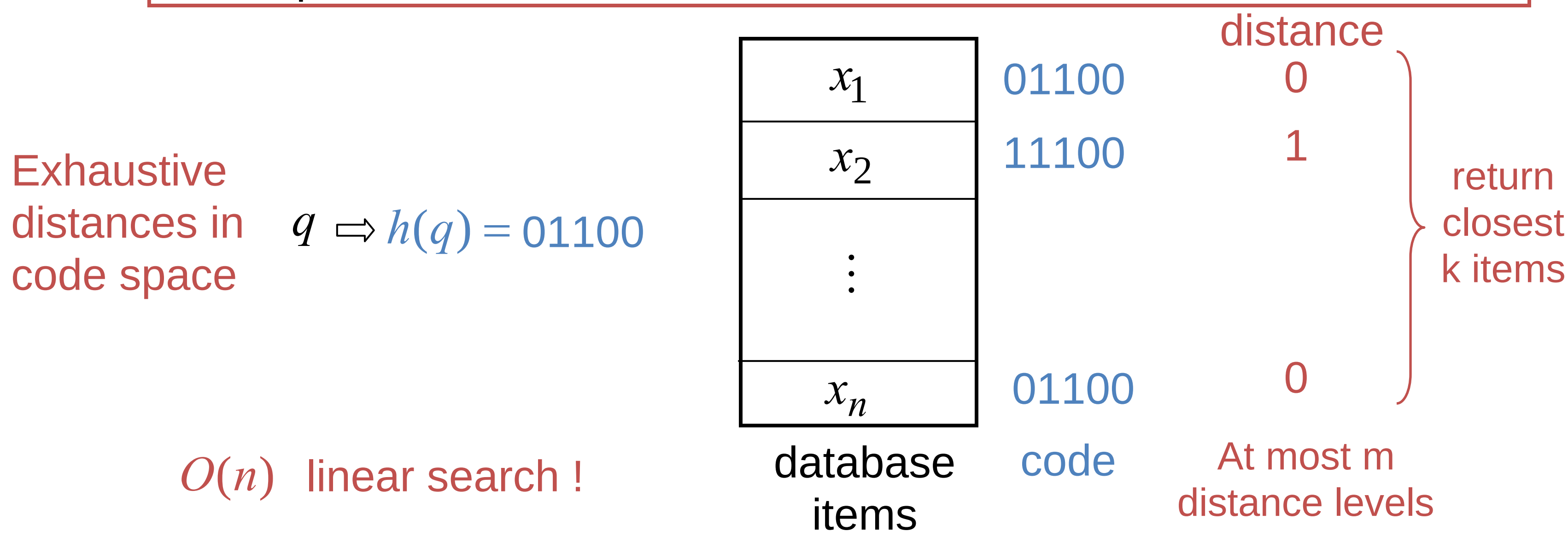
Hashing: Main Steps

3. Querying
   - Convert the query to code
   - Find all items with the same code in database using hash table
     - Return all points within a small radius of query in code space
     - Use multiple codes (and tables) to increase recall
   - Rank all the database items according to their distance from query in code space



Hashing: Main Steps

3. Querying
   - Convert the query to code
   - Find all items with the same code in database using hash table
     - Return all points within a small radius of query in code space
     - Use multiple codes (and tables) to increase recall
   - Rank all the database items according to their distance from query in code space



Hashing Techniques

1. Unsupervised – use unlabeled data to learn hash functions
   - Locality Sensitive Hashing (LSH), PCA Hashing, Spectral Hashing, Min-Hashing, Kernel-LSH, …
2. Supervised – use labeled pairs to learn hash functions
   - Boosted Hashing, Binary Reconstructive Embedding,…
3. Semi-Supervised – use labeled pairs and unlabeled data both
   - Sequential Learning,…
4. Type of Hash Function
   - Linear/Quasi-linear: LSH, Min-Hash, SH, PCA-Hash, …
   - Nonlinear: KLSH, RBM, BRE,…

Locality Sensitive Hashing (LSH)

A family of hash functions $H = \{h : X \rightarrow Z\}$ is called (r_1, r_2, p_1, p_2)- sensitive if for any $x_1, x_2 \in X$

$$\text{if } d(x_1, x_2) \le r_1 \text{ then } \Pr[h(x_1) = h(x_2)] \ge p_1,$$

$$\text{if } d(x_1, x_2) > r_2 \text{ then } \Pr[h(x_1) = h(x_2)] \le p_2.$$

where $r_1 < r_2$ and $p_1 > p_2$

Locality Sensitive Hashing (LSH)

A family of hash functions $H = \{h : X \to Z\}$ is called (r_1, r_2, p_1, p_2)- sensitive if for any $x_1, x_2 \in X$

$$\text{if } d(x_1, x_2) \le r_1 \text{ then } \Pr[h(x_1) = h(x_2)] \ge p_1,$$
$$\text{if } d(x_1, x_2) > r_2 \text{ then } \Pr[h(x_1) = h(x_2)] \le p_2.$$

where $r_1 < r_2$ and $p_1 > p_2$

A simple LSH family

$$h_k(x) = \left\lfloor (w_k^T x + b_k)/t \right\rfloor \quad w_k \sim P_s(w) \quad b_k \sim U[0, t]$$

s-stable distribution

Locality Sensitive Hashing (LSH)

A family of hash functions $H = \{h : X \rightarrow Z\}$ is called (r_1, r_2, p_1, p_2)- sensitive if for any $x_1, x_2 \in X$

$$\text{if } d(x_1, x_2) \le r_1 \text{ then } \Pr[h(x_1) = h(x_2)] \ge p_1,$$

$$\text{if } d(x_1, x_2) > r_2 \text{ then } \Pr[h(x_1) = h(x_2)] \le p_2.$$

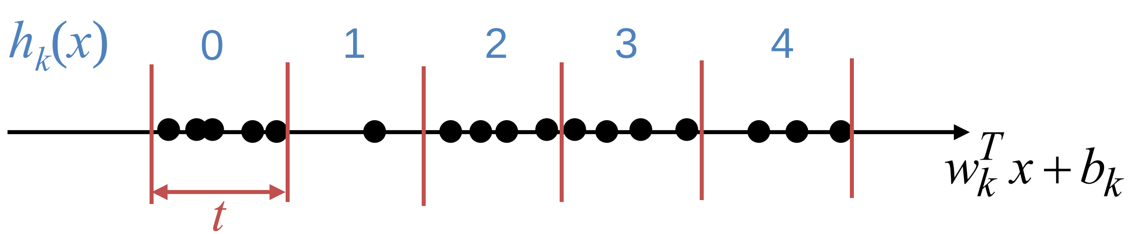
where $r_1 < r_2$ and $p_1 > p_2$

A simple LSH family

$$h_k(x) = \left\lfloor (w_k^T x + b_k)/t \right\rfloor \quad w_k \sim P_s(w) \quad b_k \sim U[0, t]$$

s-stable distribution

Special case:
binary hashing



s-Stable Distributions

A distribution $P_s(r)$ is called s-stable if there exists an $s \geq 0$ such that for any $x \in \Re^d$, and any w with $i.i.d.\ w^i \sim P_s$, then

$$x^T w \sim \|x\|_s w^i$$

$$\Rightarrow (x_1 - x_2)^T w \sim \|(x_1 - x_2)\|_s w^i$$

Neighboring points tend to have similar projections → Binning projections has LSH property !

s-Stable Distributions

A distribution $P_s(r)$ is called s-stable if there exists an $s \geq 0$ such that for any $x \in \Re^d$, and any w with $i.i.d.\ w^i \sim P_s$, then

$$x^T w \sim \|x\|_s w^i$$

$$\Rightarrow (x_1 - x_2)^T w \sim \|(x_1 - x_2)\|_s w^i$$

Neighboring points tend to have similar projections → Binning projections has LSH property !

Special Case: s = 2 (Euclidean distance)

$$w^i \sim P_s = N(0,1) \Rightarrow w \sim N(0, I)$$

$$E[x^T w] = 0$$

$$Var[x^T w] = E[x^T w w^T x] = x^T E[w w^T] x = \|x\|_2^2$$

$$x^T w \sim \|x\|_2 w^i$$

Gaussian distribution is 2-stable !

Which distribution is 1-stable? Cauchy !

One can find s-stable distribution for all s $\in$(0, 2]

Collision Probability

Suppose $u = \|(x_1 - x_2)\|_s$ and $f_s(a)$ is pdf of absolute of s-stable random variable, i.e., $a = |w^i|$, then probability of collision,

$$p(u) = \Pr[h(x_1) = h(x_2)] = \int_0^t (1/u) f_s(a/u)(1 - a/t)da$$

$p(u)$ increases as u decreases !

Collision Probability

Suppose $u = \|(x_1 - x_2)\|_s$ and $f_s(a)$ is pdf of absolute of s-stable random variable, i.e., $a = |w^i|$, then probability of collision,

$$p(u) = \Pr[h(x_1) = h(x_2)] = \int_0^t (1/u) f_s(a/u)(1 - a/t)\,da$$

$p(u)$ increases as u decreases !

How to choose t ?

$$\hat{t} = \arg\min_t \rho = \arg\min_t [\log(1/p_1)/\log(1/p_2)]$$

Can be computed analytically for s = 1, 2

Collision Probability

Suppose $u = \|(x_1 - x_2)\|_s$ and $f_s(a)$ is pdf of absolute of s-stable random variable, i.e., $a = |w^i|$, then probability of collision,

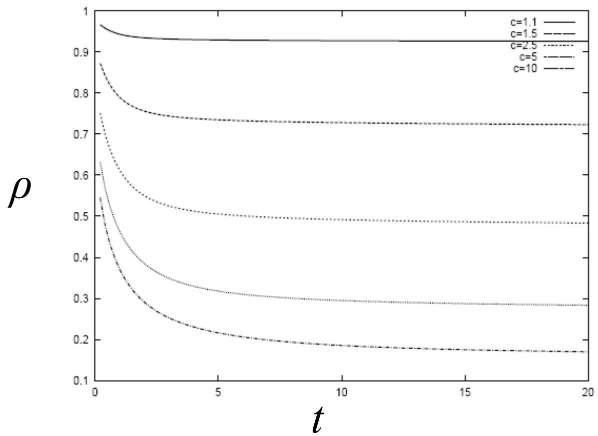
$$p(u) = \Pr[h(x_1) = h(x_2)] = \int_0^t (1/u) f_s(a/u)(1 - a/t)\, da$$

$p(u)$ increases as u decreases !

How to choose t ?

$$\hat{t} = \arg\min_t \rho = \arg\min_t [\log(1/p_1)/\log(1/p_2)]$$

Can be computed analytically for s = 1, 2



ρ not too sensitive to t if sufficiently away from 0 !

Datar et al.[5]

Parameter selection in LSH

How many tables (L) and how many bits per table (m)?

Given a query q and its near-neighbor x', suppose each hash function satisfies

$$\Pr[h_k(q) = h_k(x')] \geq p_1$$

Parameter selection in LSH

How many tables (L) and how many bits per table (m)?

Given a query q and its near-neighbor x' , suppose each hash function satisfies

$$\Pr[h_k(q) = h_k(x')] \geq p_1$$

For m-bit code $$\Pr[h(q) = h(x')] \geq p_1^m$$

Probability of collision falls exponentially with m !

Parameter selection in LSH

How many tables (L) and how many bits per table (m)?

Given a query q and its near-neighbor x', suppose each hash function satisfies

$$\Pr[h_k(q) = h_k(x')] \geq p_1$$

For m-bit code $\Pr[h(q) = h(x')] \geq p_1^m$

Probability of collision falls exponentially with m !

Probability that q and x' will fail to collide for all L tables

$$\leq (1 - p_1^m)^L$$

Bound the probability that q and x' will collide for at least one of L tables

$$1 - (1 - p_1^m)^L \leq 1 - \delta$$

$$\boxed{L \geq -\log(1/\delta) / \log(1 - p_1^m)}$$

Precision-Recall Tradeoff

- For high precision, longer codes (i.e. large m) preferred
- Large m reduces the probability of collision exponentially → low recall
- Many tables (large L) necessary to get good recall → Large storage

Precision-Recall Tradeoff

- For high precision, longer codes (i.e. large m) preferred
- Large m reduces the probability of collision exponentially → low recall
- Many tables (large L) necessary to get good recall → Large storage

Design L and m such that run-time is minimized for a given application:

$$T_{total} = T_h + T_r$$

T_h: To compute L m-bit hash functions and retrieve points from tables via lookups

T_r: To compute exact distance with retrieved points and return top k (sublinear in n)

- Larger m increases T_h but decreases T_r
- Empirically estimate total time averaged over many queries
- In some cases, T_r is simply vote over how many tables returned an item

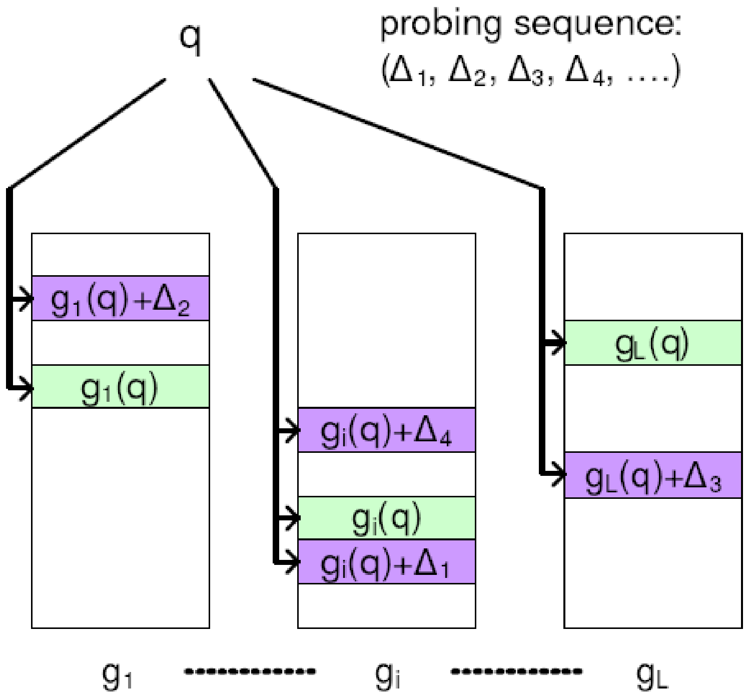
How to avoid large number of tables?

Multi-Probe LSH

Strategy to increase recall without using large number of tables

Basic idea

- If two neighbors do not fall in the same bucket, they should fall in a nearby one, e.g., within a hamming distance of 1



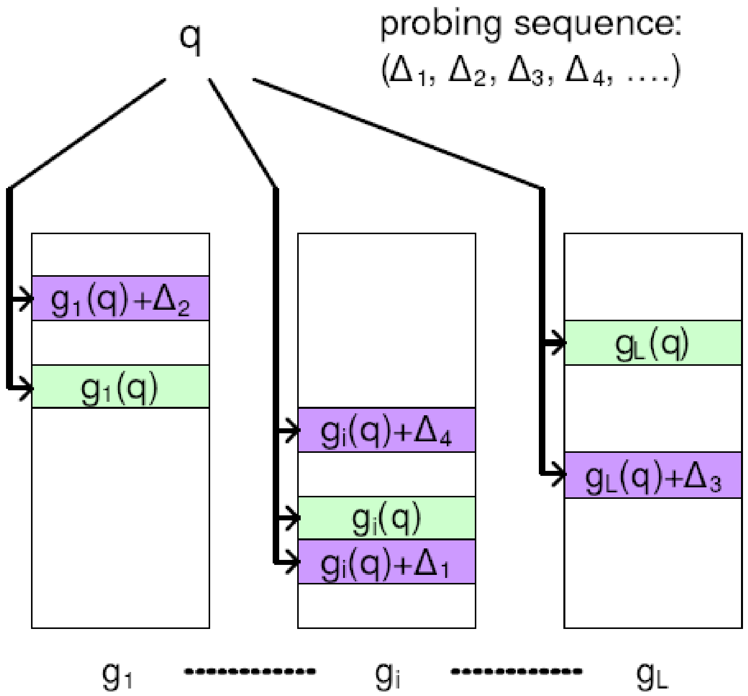
Lv et al.[7]

Multi-Probe LSH

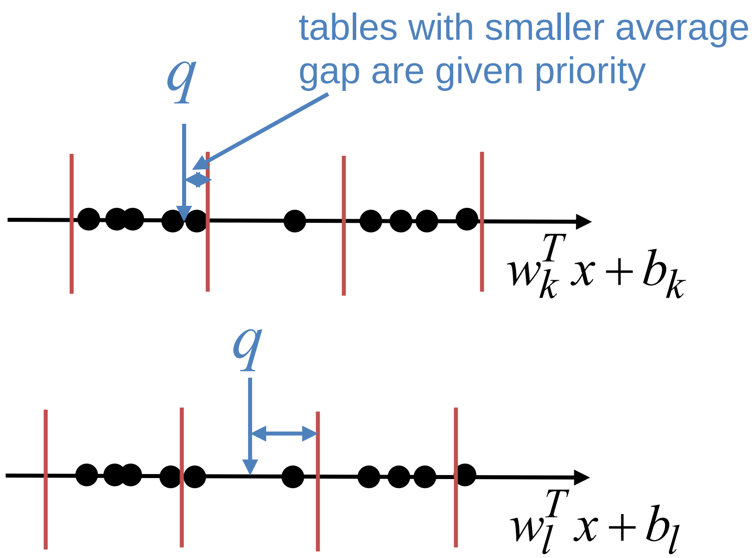
Strategy to increase recall without using large number of tables

Basic idea

- If two neighbors do not fall in the same bucket, they should fall in a nearby one, e.g., within a hamming distance of 1



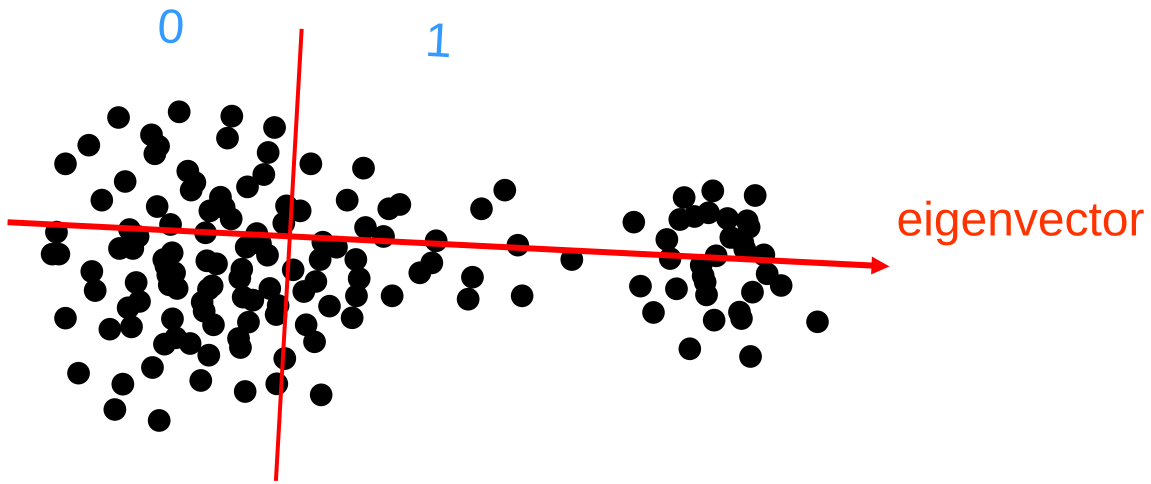
How to choose the sequence?



Lv et al.[7]

Data-Dependent Projections

– PCA-Hash: Let's focus on binary codes

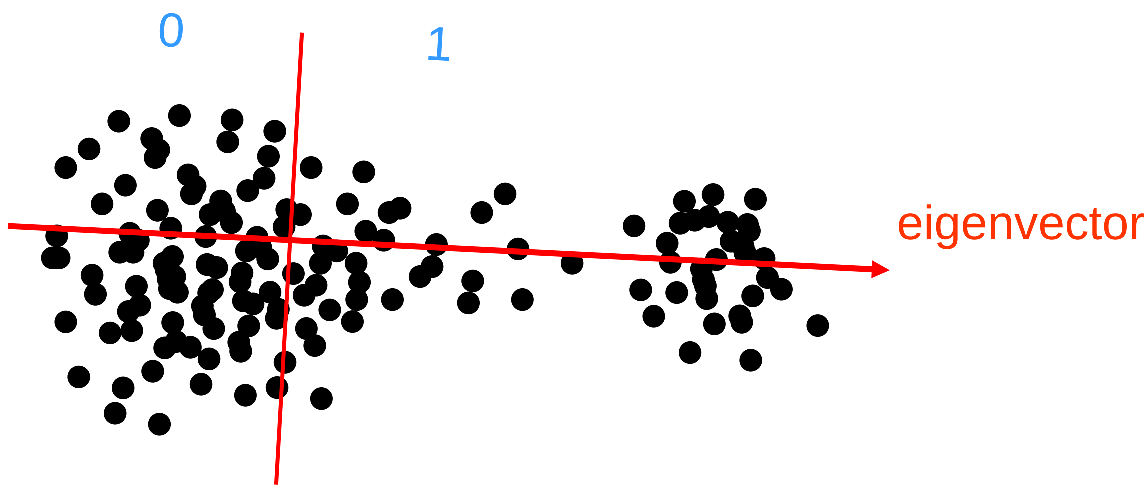


$$h_k(x) = \text{sgn}(w_k^T x + b_k) \qquad w_k \sim \text{eigenvec}(Cov(X))$$

Projection on max variance directions followed by median threshold

Data-Dependent Projections

– PCA-Hash: Let's focus on binary codes



$$h_k(x) = \mathrm{sgn}(w_k{}^T x + b_k) \qquad w_k \sim \mathrm{eigenvec}\,(Cov(X))$$

Projection on max variance directions followed by median threshold

Performance degrades with larger number of bits

- Variance decreases rapidly for most real-world data
- Can one reuse the high variance directions?

Spectral Hash

Data-dependent learning of binary codes $h(x)$ such that

similarity between x_i, x_j

$$\min \sum_{i,j} W_{ij} \left\| h(x_i) - h(x_j) \right\|^2$$

subject to

$$\sum_i h_k(x_i) = 0 \quad \forall\, k$$

balanced partitioning $h_k(x) \in \{-1, 1\}$

$$\sum_i h_k(x_i) h_l(x_i) = 0 \quad \forall\, k \neq l$$

uncorrelated

Spectral Hash

Data-dependent learning of binary codes $h(x)$ such that

similarity between x_i, x_j

$$\min \sum_{i,j} W_{ij} \left\| h(x_i) - h(x_j) \right\|^2$$

Graph Laplacian → O(n²)

subject to

$$\sum_i h_k(x_i) = 0 \quad \forall\, k$$

balanced partitioning $h_k(x) \in \{-1, 1\}$

$$\sum_i h_k(x_i) h_l(x_i) = 0 \quad \forall\, k \neq l$$

uncorrelated

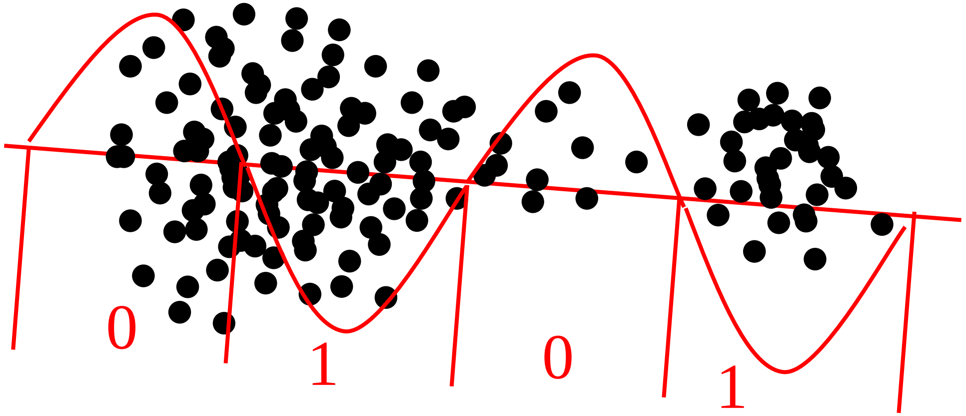
Issues

- Computationally extremely expensive (needs complete NN search)
- Balanced graph partitioning problem even with single bit → NP hard

Approximation

- Assumes uniform data distribution and solves 1D Laplacian eigenfunctions analytically

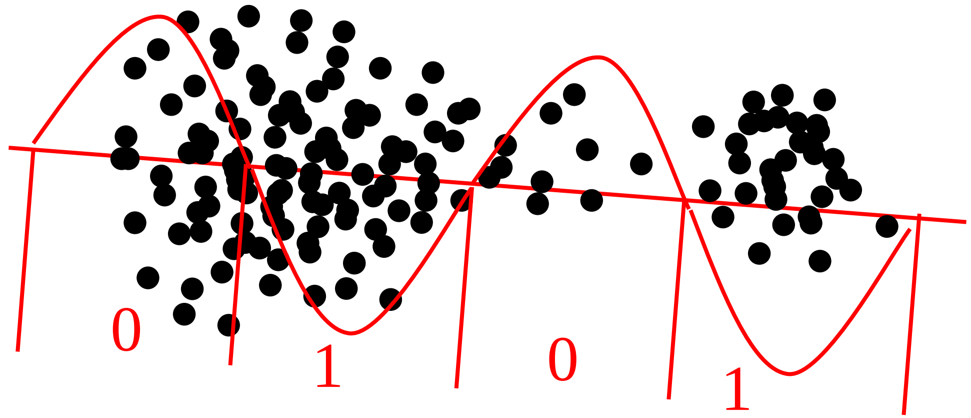
Spectral Hash



Three main steps

- Extract max-variance directions using PCA
- Select which direction to pick next based on modes of 1D-Laplacian
  - High variance PCA directions may be picked again
- Create bits by thresholding sinusoidal eigenfunctions at zero
  - slower than simple thresholding

Spectral Hash



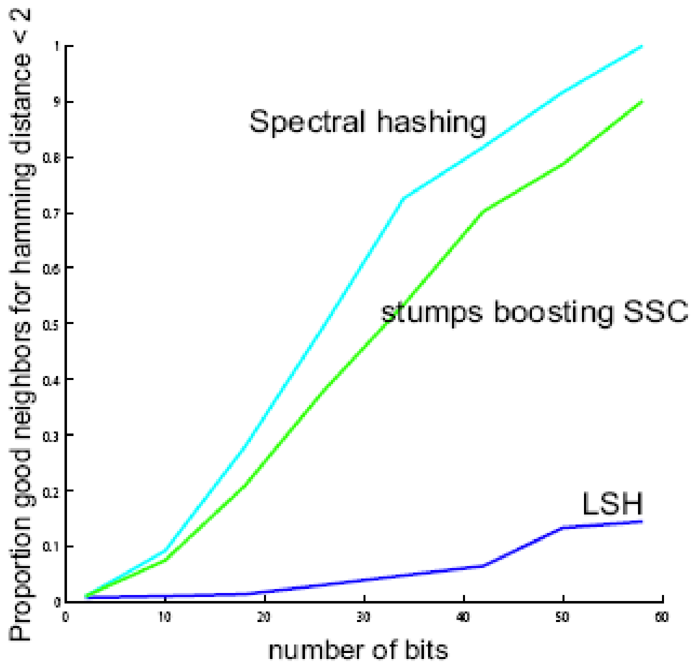
$$h_k(x) = \text{sgn}(\cos(\alpha w_k{}^T x)) \quad w_k \sim \text{eigenvec}\,(Cov(X))$$

Three main steps

- Extract max-variance directions using PCA
- Select which direction to pick next based on modes of 1D-Laplacian
  - High variance PCA directions may be picked again
- Create bits by thresholding sinusoidal eigenfunctions
  - slower than simple thresholding

In practice, PCA-hash with median threshold may do better but both suffer from low-variance directions

Spectral Hash Experiment



- Testing using Hamming radius around the query
- Dense 384-dim vector → PCA-Hash gives similar or better performance
- For Hamming-radius testing, better to use LSH with median threshold

Weiss et al.[9]

Shift-Invariant Kernel Embedding

Use random projections along with sinusoidal thresholding

Key idea

- Suppose similarity between a pair of points is given by a shift-invariant kernel, i.e.,

$$s(x,y) = K(x,y) = K(x-y) \le 1 \qquad K(x-x) = K(0) = 1$$

Examples $s(x,y) = \exp(-\gamma \|x-y\|^2 / 2)$ L_2 distance

$s(x,y) = \exp(-\gamma \|x-y\|_1)$ L_1 distance

Shift-Invariant Kernel Embedding

Use random projections along with sinusoidal thresholding

Key idea

- Suppose similarity between a pair of points is given by a shift-invariant kernel, i.e.,

$$s(x,y) = K(x,y) = K(x-y) \le 1 \qquad K(x-x) = K(0) = 1$$

Examples $s(x,y) = \exp(-\gamma \|x-y\|^2 / 2)$ L_2 distance

$s(x,y) = \exp(-\gamma \|x-y\|_1)$ L_1 distance

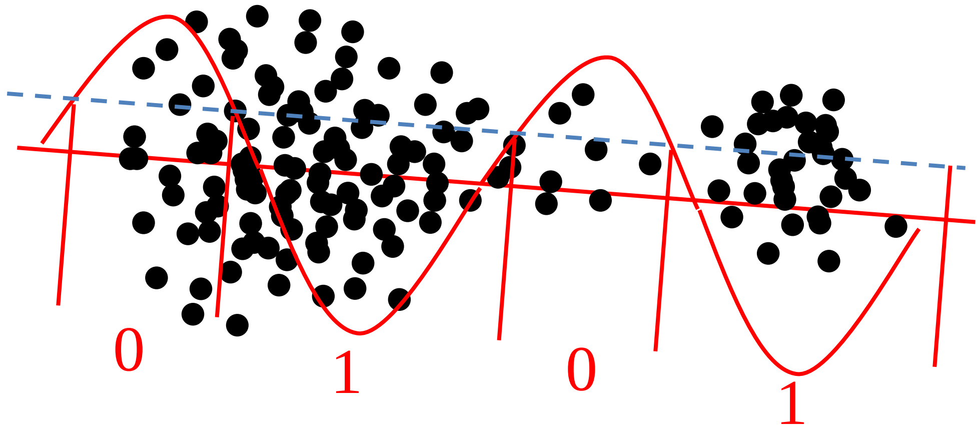
Want to learn m-bit code $h(x)$ such that

$$f_1(K(x-y)) \le \underbrace{(1/m) d_H(h(x), h(y))}_{\text{normalized Hamming distance}} \le f_2(K(x-y))$$

decreasing functions → small for similar points

$$f_1(1) = f_2(1) = 0, \; f_1(0) = f_2(0) = c > 0$$

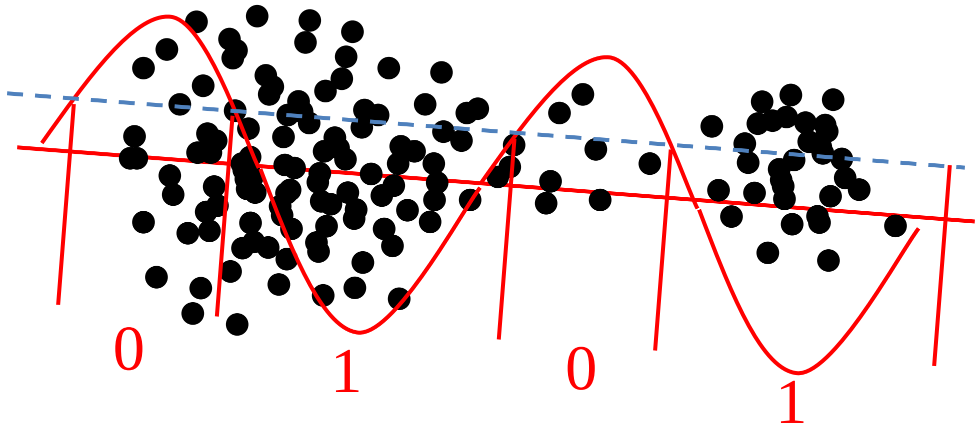
Shift-Invariant Kernel Embedding



Main steps

- Approximate shift-invariant kernels as dot products of random fourier features
- Pick directions from distribution induced by kernel → similar to s-stable directions
- Create bits by thresholding sinusoidal eigenfunctions

Shift-Invariant Kernel Embedding



standard conversion into 0/1

$$h_k(x) = \mathrm{sgn}(\cos(w_k^T x + b_k) + t_k) \qquad w_k \sim N(0, \gamma I) \;\; b_k \sim U[0, 2\pi] \;\; t_k \sim U[-1, 1]$$

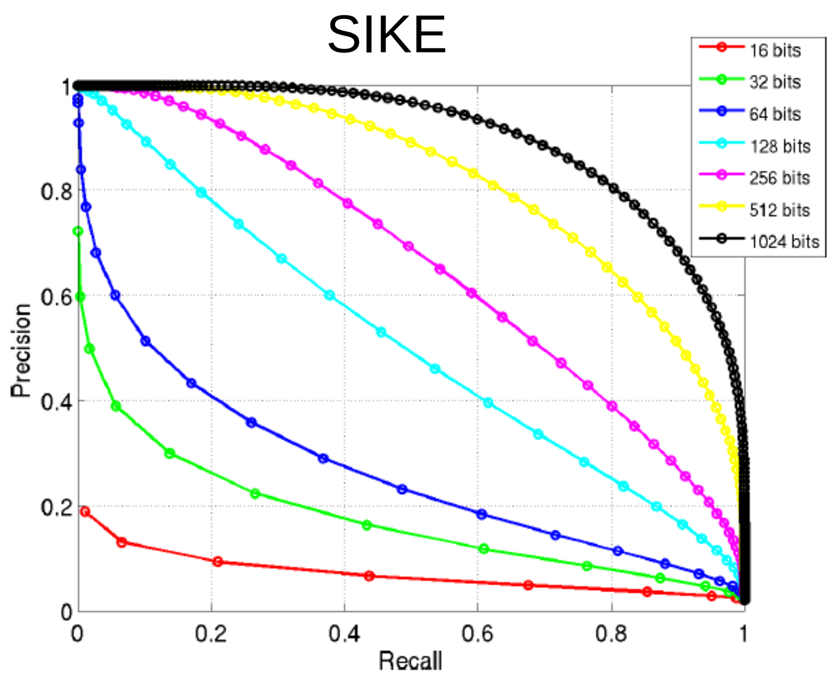
critical for performance

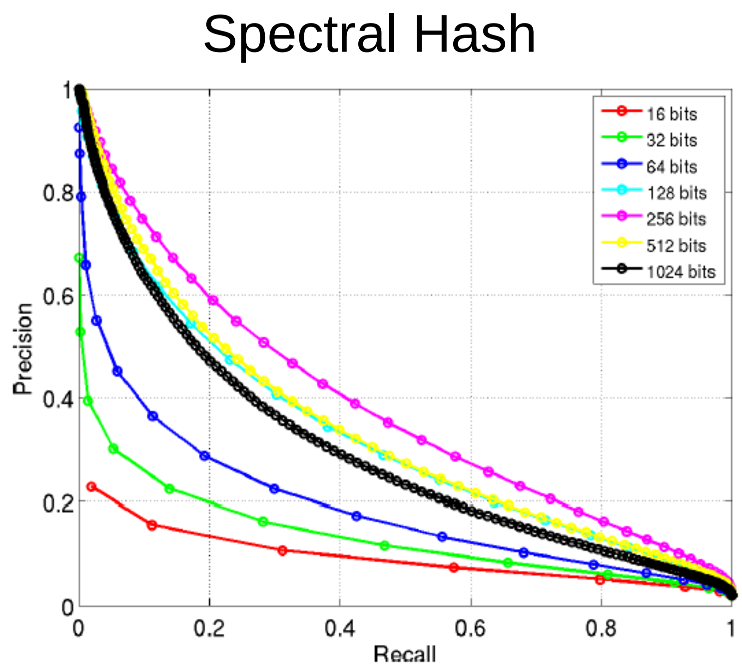
Main steps

- Approximate shift-invariant kernels as dot products of random fourier features
- Pick directions from distribution induced by kernel → similar to s-stable directions
- Create bits by thresholding sinusoidal eigenfunctions

Performance (with hamming ranking) better if large number of bits are used !

Shift-Invariant Kernel Embedding





- Test using exhaustive hamming ranking with all database items
- Dense 384-dim vectors
- After 256 bits, performance of Spectral Hash falls
- Even regular LSH quite powerful if large number of bits are used

Raginsky et al.[11]

Min-Hash

A method to estimate Jaccard similarity between sets (or vectors)

- Jaccard similarity between two sets (A, B) or two vectors (x, y)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad J(x,y) = \frac{\sum_i \min(x^i, y^i)}{\sum_i \max(x^i, y^i)} \quad \forall\, x^i, y^i \geq 0$$

Min-Hash

A method to estimate Jaccard similarity between sets (or vectors)

- Jaccard similarity between two sets (A, B) or two vectors (x, y)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad J(x,y) = \frac{\sum_i \min(x^i, y^i)}{\sum_i \max(x^i, y^i)} \quad \forall\ x^i, y^i \geq 0$$

relation with L_1 ?

- The above two definitions equivalent → represent each set as a binary vector of length $|A \cup B|$

Min-Hash

A method to estimate Jaccard similarity between sets (or vectors)

- Jaccard similarity between two sets (A, B) or two vectors (x, y)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad J(x,y) = \frac{\sum_i \min(x^i, y^i)}{\sum_i \max(x^i, y^i)} \quad \forall\, x^i, y^i \geq 0$$

relation with L_1 ?

- The above two definitions equivalent → represent each set as a binary vector of length $|A \cup B|$
- Suppose $h_k(.)$ is a random function that maps each item to a real number

$$h_k(x^i) \neq h_k(x^j) \quad \text{and} \quad \Pr[h_k(x^i) < h_k(x^j)] = 0.5$$

simple choice $h_k(x^i) = U[0,1]$

Min-Hash

A method to estimate Jaccard similarity between sets (or vectors)

- Jaccard similarity between two sets (A, B) or two vectors (x, y)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad J(x,y) = \frac{\sum_i \min(x^i, y^i)}{\sum_i \max(x^i, y^i)} \quad \forall\, x^i, y^i \geq 0$$

relation with L_1 ?

- The above two definitions equivalent → represent each set as a binary vector of length $|A \cup B|$
- Suppose $h_k(.)$ is a random function that maps each item to a real number

$$h_k(x^i) \neq h_k(x^j) \quad \text{and} \quad \Pr[h_k(x^i) < h_k(x^j)] = 0.5$$

simple choice $h_k(x^i) = U[0,1]$

min-hash $m(A, h_k) = \arg\min_{x^i \in A} h_k(x^i)$

$$\Pr[m(A,h_k) = m(B,h_k)] = J(A,B)$$

Min-Hash

min-hash $m(A, h_k) = \arg\min_{x^i \in A} h_k(x^i)$

suppose $m(A \bigcup B, h_k) = x^u$

if $x^u \in A \bigcap B \Rightarrow x^u = m(A, h_k) \;\&\; x^u = m(B, h_k)$

Thus $\Pr[m(A, h_k) = m(B, h_k)] = \frac{|A \bigcap B|}{|A \bigcup B|}$

Min-Hash

min-hash $m(A, h_k) = \arg\min_{x_i \in A} h_k(x_i)$

suppose $m(A \bigcup B, h_k) = x^u$

if $x^u \in A \bigcap B \Rightarrow x^u = m(A, h_k) \,\&\, x^u = m(B, h_k)$

Thus $\Pr[m(A, h_k) = m(B, h_k)] = \frac{|A \bigcap B|}{|A \bigcup B|}$

Sketches

- For retrieval efficiency, min-hashes are grouped in s-tuples. For s random functions $(h_1, \ldots, h_s)$,

$$sketch\,(A) = (m(A, h_1), \ldots, m(A, h_s))$$

$$\Pr[sketch\,(A) = sketch\,(B)] = J(A, B)^s$$

- In practice, many sketches are created and sets (i.e. vectors) that have at least k sketches in common are retrieved for further testing.
- Generalizations to non-binary vectors, continuous valued vectors possible
- Good performance for high-dim (but mostly sparse) vectors

Kernel Locality Sensitive Hashing (KLSH)

Learn LSH-type codes but when only kernel similarity, $k(x,y)$, is known

- Data may not be given in explicit vector space

A different view of LSH

$$\Pr[h(x) = h(y)] = sim\,(x, y)$$

Kernel Locality Sensitive Hashing (KLSH)

Learn LSH-type codes but when only kernel similarity, $k(x,y)$, is known

- Data may not be given in explicit vector space

A different view of LSH

$$\Pr[h(x) = h(y)] = sim(x, y) \quad \rightarrow [0,1]$$

query-time to find (1+ε)-neighbor $O(n^{1/(1+\varepsilon)})$

Example

$$sim(x, y) = x^T y \qquad h_k(x) = \begin{cases} 1, & \text{if } r^T x > 0 \\ 0 & \text{otherwise} \end{cases} \quad r \sim N(0, I)$$

$$\Pr[\text{sgn}(r^T x) = \text{sgn}(r^T y)] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{x^T y}{\|x\|\|y\|}\right)$$

Kernel Locality Sensitive Hashing (KLSH)

Learn LSH-type codes but when only kernel similarity, $k(x,y)$, is known

- Data may not be given in explicit vector space

A different view of LSH

$$\Pr[h(x) = h(y)] = sim(x, y) \quad \rightarrow [0, 1]$$

query-time to find (1+ε)-neighbor $O(n^{1/(1+\varepsilon)})$

Example

$$sim(x, y) = x^T y \qquad h_k(x) = \begin{cases} 1, & \text{if } r^T x > 0 \\ 0 & \text{otherwise} \end{cases} \quad r \sim N(0, I)$$

$$\Pr[\mathrm{sgn}(r^T x) = \mathrm{sgn}(r^T y)] = 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{x^T y}{\|x\| \|y\|} \right)$$

Suppose we are given only similarity

implicit (unknown) feature vector

$$sim(x, y) = k(x, y) = \Phi(x)^T \Phi(y)$$

How to compute the hash function when vector is not known ?

Kernel Locality Sensitive Hashing (KLSH)

Goal: To find appropriate random projection in implicit feature space $r^T \Phi(x)$

From RKHS argument

$$r = \sum_{i=1}^{n} w_i \Phi(x_i) \approx \sum_{i=1}^{p} w_i \Phi(x_i)$$

$$= \Phi w$$

randomly chosen $p << n$

Kernel Locality Sensitive Hashing (KLSH)

Goal: To find appropriate random projection in implicit feature space $r^T \Phi(x)$

From RKHS argument

$$r = \sum_{i=1}^{n} w_i \Phi(x_i) \approx \sum_{i=1}^{p} w_i \Phi(x_i) = \Phi w$$

randomly chosen $p << n$

Find w such that $E[r] = 0,\ E[rr^T] = I$

$$w = K^{-1/2} e_s$$

$K = \Phi^T \Phi,\ \ e_s = \underbrace{[0,0,1,0,1\ldots]^T}$

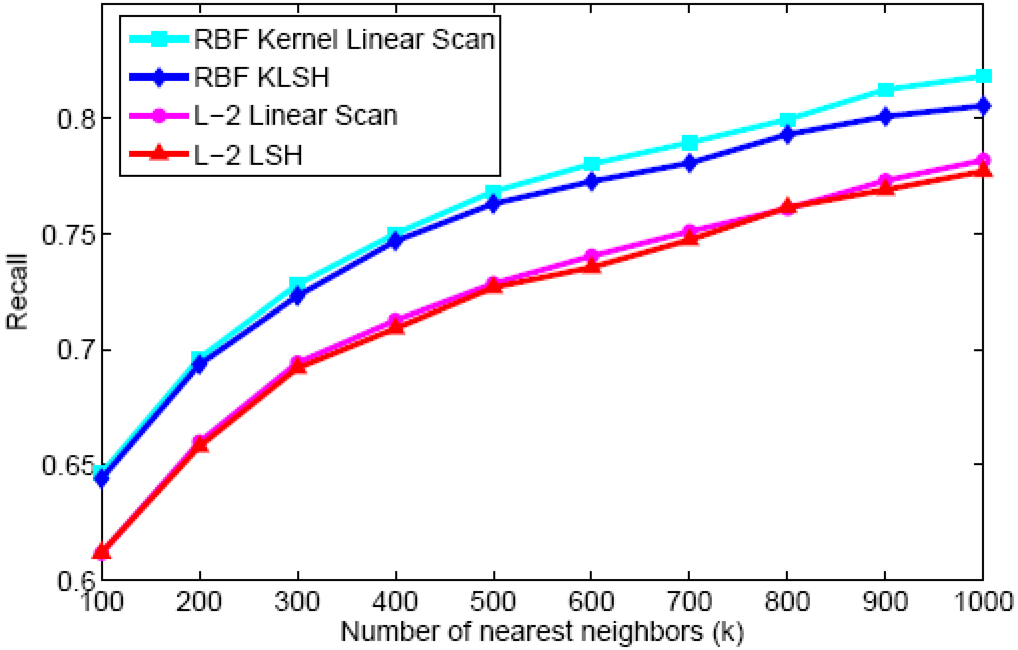
\# of 1's is a parameter

$$h(\Phi(x)) = \text{sgn}(r^T \Phi(x) = \text{sgn} \sum_{i=1}^{p} w_i k(x, x_i)$$

non-linear hashing

usually slower run-time !

KLSH vs LSH



n = 100K image patches

Kulis et al.[12]

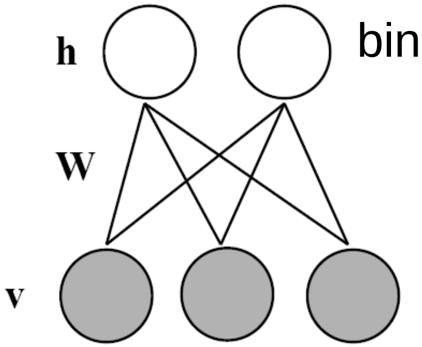
Semantic Hashing (RBM)

Nonlinear method to create binary codes using Restricted Boltzmann Machines (RBMs)

– Special type of Markov Random Fields

$$p(v,h) = \exp\{-E(v,h)\} / Z$$

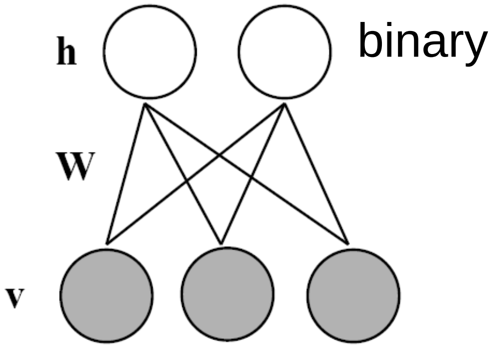
$$p(v) = \sum_h p(v,h)$$



Semantic Hashing (RBM)

Nonlinear method to create binary codes using Restricted Boltzmann Machines (RBMs)

– Special type of Markov Random Fields



$$p(v,h) = \exp\{-E(v,h)\}/Z$$

$$p(v) = \sum_h p(v,h)$$

$$p(h_j = 1 | v) = \sigma(b_j + \sum_i w_{ij} v_i) \qquad \sigma(x) = 1/(1+e^{-x})$$

$$p(v_i | h) = N(f(h), \tau I)$$

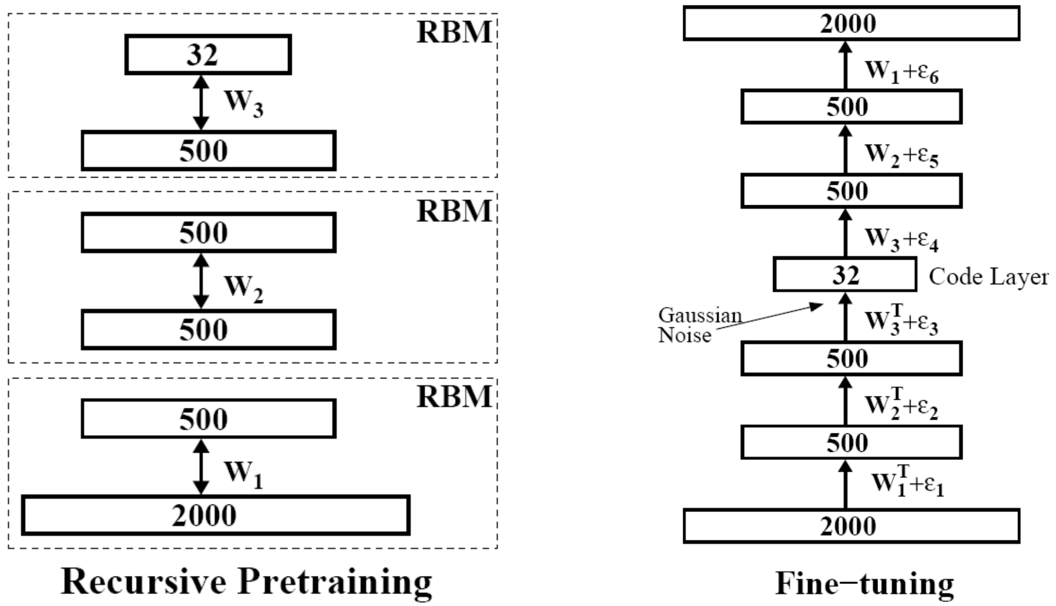
Learn parameters (W and b) using (approx) max-likelihood $p(v)$

How to learn codes ? → Stack multiple RBMs (Deep Belief Networks)

Semantic Hashing (RBM)

Deep Belief Networks

– Similar functionality as multi-layer Neural Nets



Output of each stage used as input to next

Back-propagation (coordinate descent) with Auto-encoding objective

Many parameters, architecture choices, usually slow to train and to apply !

Salakhutdinov et al.[8]

Binary Reconstructive Embedding

Construct binary codes by minimizing difference between original (metric) distance and hamming distance

$$h_k(x) = \text{sgn}(w_k^T v(x)) \quad \text{where} \quad v(x) = [1, K(x_{k_1}, x), \ldots, K(x_{k_s}, x)]^T$$

kernel

chosen randomly or learned

Binary Reconstructive Embedding

Construct binary codes by minimizing difference between original (metric) distance and hamming distance

$$h_k(x) = \text{sgn}(w_k^T v(x)) \quad \text{where} \quad v(x) = [1, K(x_{k_1}, x), \ldots, K(x_{k_s}, x)]^T$$

kernel

chosen randomly or learned

$$\hat{W} = \arg\min_{W} \sum_{(x_i, x_j) \in T} [d_M(x_i, x_j) - d_H(x_i, x_j)]^2$$

Binary Reconstructive Embedding

Construct binary codes by minimizing difference between original (metric) distance and hamming distance

$$h_k(x) = \mathrm{sgn}(w_k^T v(x)) \quad \text{where} \quad v(x) = [1, K(x_{k_1}, x), \ldots, K(x_{k_s}, x)]^T$$

kernel

chosen randomly or learned

$$\hat{W} = \arg\min_{W} \sum_{(x_i, x_j) \in T} [d_M(x_i, x_j) - d_H(x_i, x_j)]^2$$

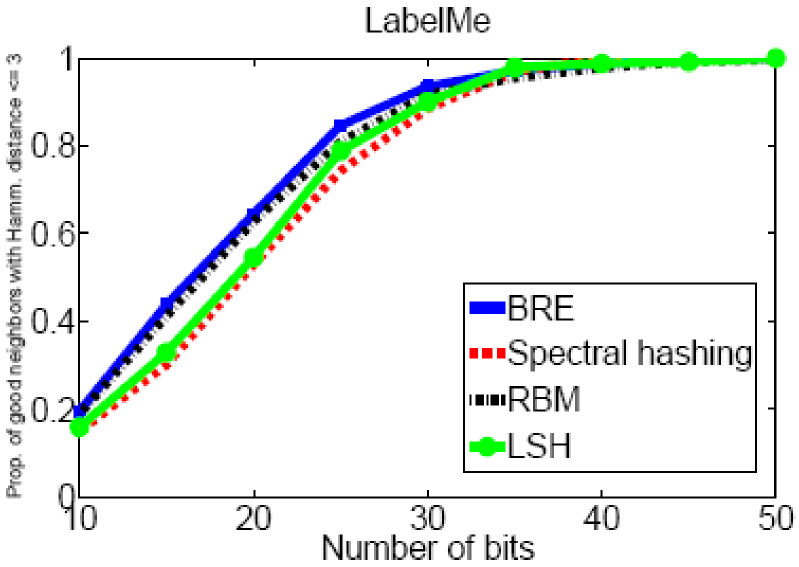
$(1/2)\|x_i - x_j\|^2$

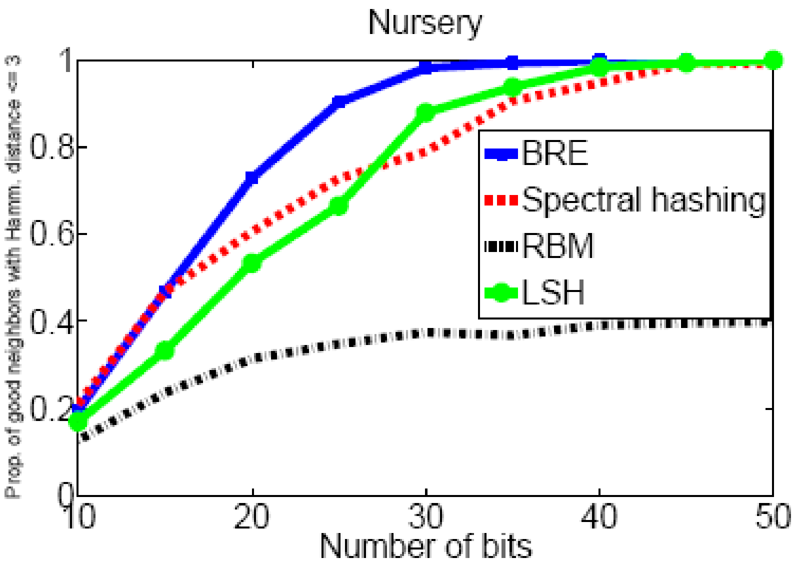
$(1/4m)\sum_{k=1}^{m}[h_k(x_i) - h_k(x_j)]^2$

- Data scaled to unit norm to remove the effect of scale (but this changes the distance between points), uniform scaling better
- Coordinate Descent for optimization, deals with discontinuities
- Expensive to compute codes, learning appropriate anchors for kernel representations hard

Binary Reconstructive Embedding

Testing based on points retrieved within hamming radius 3



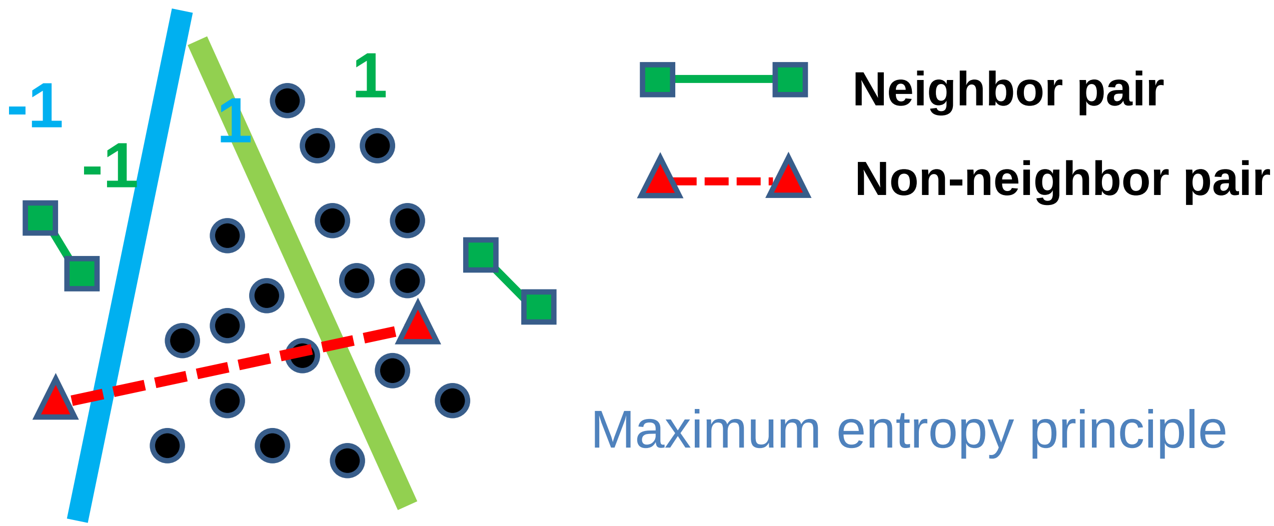


LSH is quite close even for moderate number of bits !

Kulis et al.[13]

Semi-supervised Hashing

Suppose a few neighbor pairs and a few non-neighbor pairs are given



Maximum entropy principle

Semi-supervised Formulation

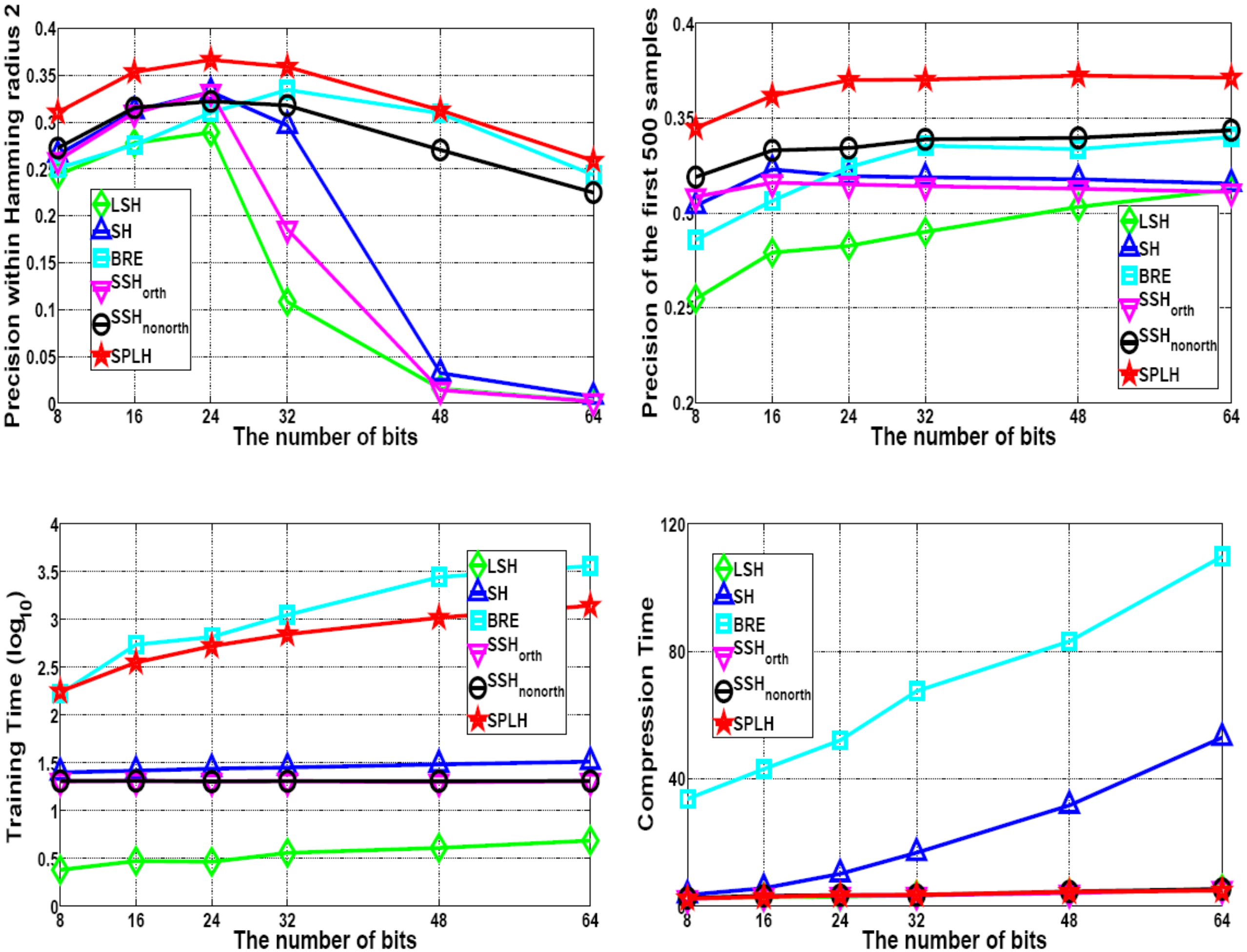
$$\max_{h_k} \{ J(h_k(\mathbf{X}_l)) + \eta \cdot Entropy(h_k(\mathbf{X})) \}$$

Empirical fitness (labeled data)

Regularizer (all data)

Wang et al.[14]

Flickr-270K



Wang et al.[15]

References

1. A. Broder, "On the resemblance and containment of documents," In Compression and Complexity of Sequences, 1997.
2. P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. of 30th ACM Symposium on Theory of Computing*, 1998
3. A. Gionis, P. Indyk, R. motwani, "Similarity Search in High Dimensions via Hashing," In VLDB, 1999.
4. G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proc. of the IEEE International Conference on Computer Vision*, 2003.
5. M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Annual Symposium on Computational Geometry*, 2004.
6. G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.
7. Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multiprobe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large databases*, 2007
8. R. Salakhutdinov and G. Hinton, "Semantic Hashing," ACM SIGIR, 2007.
9. Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008.
10. O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. BMVC., 2008.
11. M. Raginsky and S. Lazebnik, "Locality-Sensitive Binary Codes from Shift-Invariant Kernels," in *Proc. of Advances in neural information processing systems*, 2009.
12. B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," *Proc. of the IEEE International Conference on Computer Vision*, 2009.
13. B. Kulis and T. Darrell, "Learning to Hash with Binary Reconstructive Embeddings," in *Proc. of Advances in Neural Information Processing Systems*, 2009
14. J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
15. J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proceedings of the 27th International Conference on Machine Learning, 2010.*