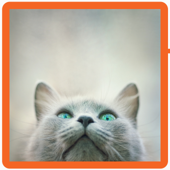
Neighbor-Sensitive Hashing

Yongjoo Park
Michael Cafarella
Barzan Mozafari

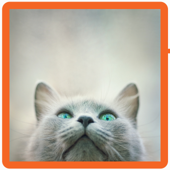
University of Michigan, Ann Arbor

k-Nearest Neighbors Problem (kNN)



query

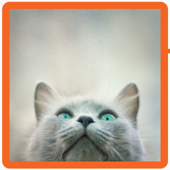
k-Nearest Neighbors Problem (kNN)



query

database

k-Nearest Neighbors Problem (kNN)

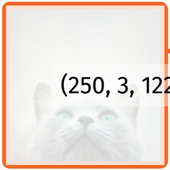


query

What are the k most similar items?

database

k-Nearest Neighbors Problem (kNN)



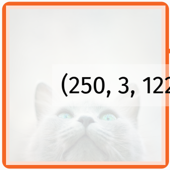
query

(250, 3, 122, 130, 68, . . .)

What are the k most similar items?

database

k-Nearest Neighbors Problem (kNN)



query

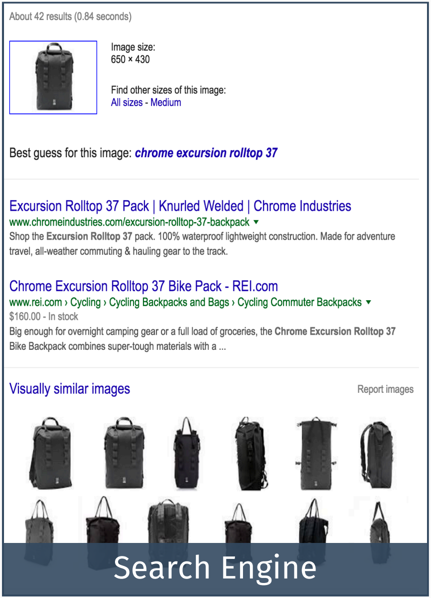
(250, 3, 122, 130, 68, . . .)

What are the k most similar items?

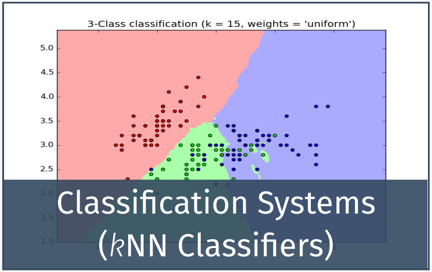
(109, 33, 92, 87, 161, . . .),
(50, 83, 22, 230, 98, . . .),
(2, 183, 59, 18, 178, . . .),
(221, 183, 259, 88, 112, . . .),
. . ..

database

kNN is Heart of Key Applications



Search Engine

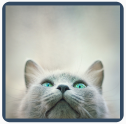


Classification Systems (kNN Classifiers)

Recommender Systems (Collaborative Filtering)

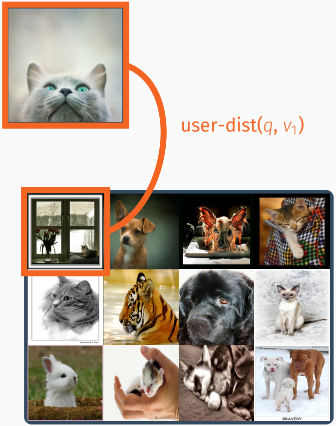
Naïve Approach to *k*NN

Naïve Approach: *linear search* with the original representations



Naïve Approach to kNN

Naïve Approach: *linear search* with the original representations



Naïve Approach to *k*NN

Naïve Approach: *linear search* with the original representations

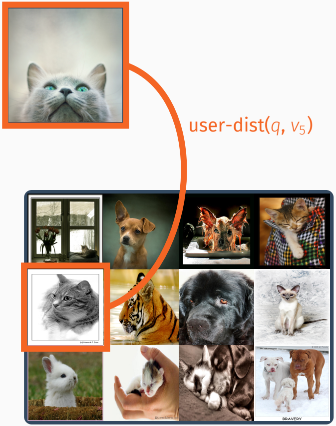
Naïve Approach to kNN

Naïve Approach: *linear search* with the original representations

Naïve Approach: *linear search* with the original representations

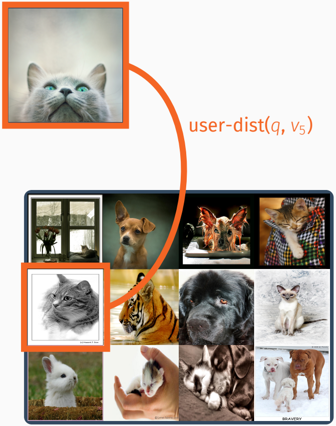
Naïve Approach to kNN

Naïve Approach: *linear search* with the original representations



Naïve Approach to kNN

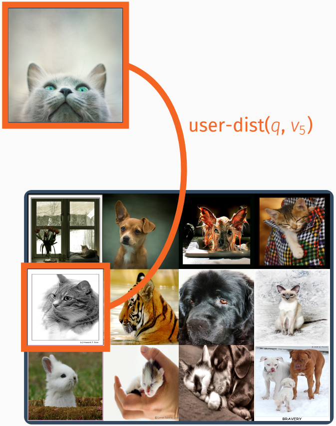
Naïve Approach: *linear search* with the original representations



Pick the items with the k smallest user-defined distances

Naïve Approach to kNN

Naïve Approach: *linear search* with the original representations

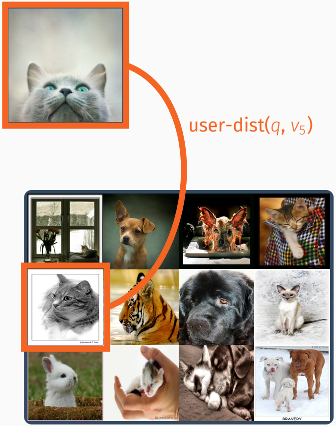


Pick the items with the k smallest user-defined distances

Extremely slow

Naïve Approach to kNN

Naïve Approach: *linear search* with the original representations



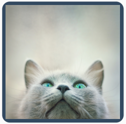
Pick the items with the k smallest user-defined distances

Extremely slow

Note: No known fast exact algorithms for *dense, high-dimensional* vectors

Locality-Sensitive Hashing for kNN

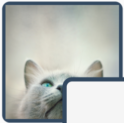
LSH: Use similarity-preserving hash functions

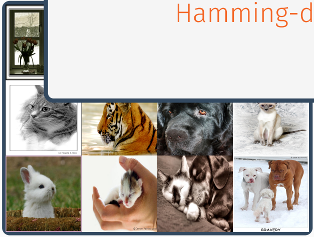


First proposed by [Datar et al., 2004] and [Charikar, 2002]

Locality-Sensitive Hashing for kNN

LSH: Use similarity-preserving hash functions





Let $h(\cdot)$ be a function that produces a hashcode. Then,

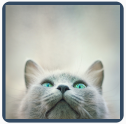
$$\text{Hamming-dist}(h(q), h(v_i)) \quad \propto \quad \text{user-dist}(q, v_i)$$

First proposed by [Datar et al., 2004] and [Charikar, 2002]

Locality-Sensitive Hashing for kNN

LSH: Use similarity-preserving hash functions

Hamming-dist$(h(q), h(v_i)) \propto$ user-dist(q, v_i)

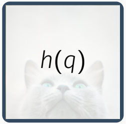


First proposed by [Datar et al., 2004] and [Charikar, 2002]

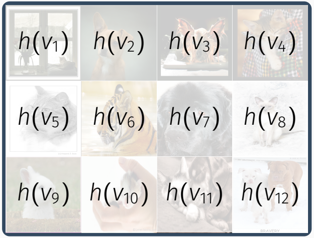
Locality-Sensitive Hashing for kNN

LSH: Use similarity-preserving hash functions

$$\text{Hamming-dist}(h(q), h(v_i)) \propto \text{user-dist}(q, v_i)$$



Hashed Query

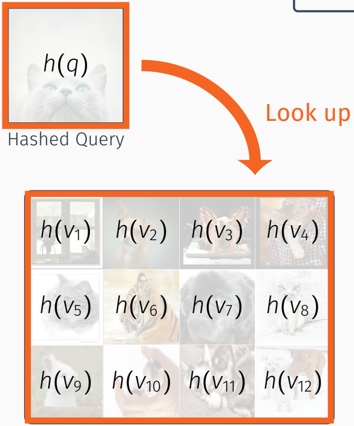


Hashed DB

First proposed by [Datar et al., 2004] and [Charikar, 2002]

LSH: Use similarity-preserving hash functions

Hamming-dist$(h(q), h(v_i)) \propto$ user-dist(q, v_i)



Hashed DB

First proposed by [Datar et al., 2004] and [Charikar, 2002]

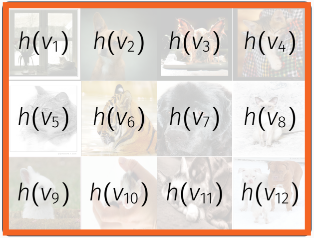
Locality-Sensitive Hashing for kNN

LSH: Use similarity-preserving hash functions

Hamming-dist$(h(q), h(v_i)) \propto$ user-dist(q, v_i)

Hashed Query

Look up



Hashed DB

hashcodes $\rightarrow$ lookup operations in a hash table $\rightarrow$ fast.

First proposed by [Datar et al., 2004] and [Charikar, 2002]

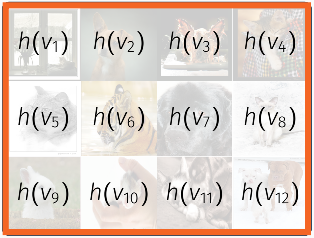
Locality-Sensitive Hashing for kNN

LSH: Use similarity-preserving hash functions

$\text{Hamming-dist}(h(q), h(v_i)) \propto \text{user-dist}(q, v_i)$

Hashed Query

Look up



Hashed DB

hashcodes $\rightarrow$ lookup operations in a hash table $\rightarrow$ fast.

Perfect hash functions may not exist, or extremely hard to find $\rightarrow$ approximate.

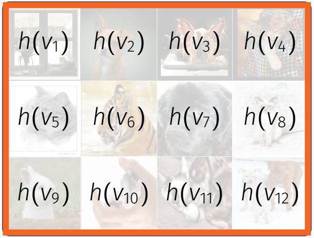
First proposed by [Datar et al., 2004] and [Charikar, 2002]

LSH: Use similarity-preserving hash functions

$$\text{Hamming-dist}(h(q), h(v_i)) \propto \text{user-dist}(q, v_i)$$

Hashed Query

Look up



Hashed DB

hashcodes $\rightarrow$ lookup operations in a hash table $\rightarrow$ fast.

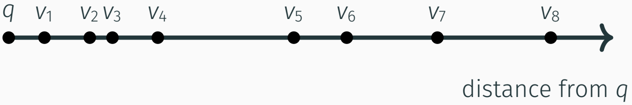
Perfect hash functions may not exist, or extremely hard to find $\rightarrow$ approximate.

Note: Longer hashcode makes the searching slower.

First proposed by [Datar et al., 2004] and [Charikar, 2002]

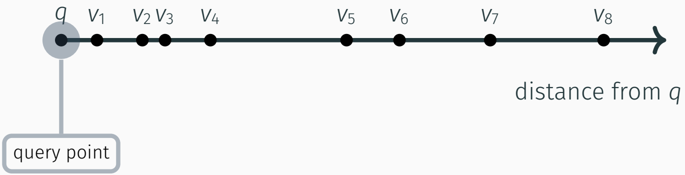
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



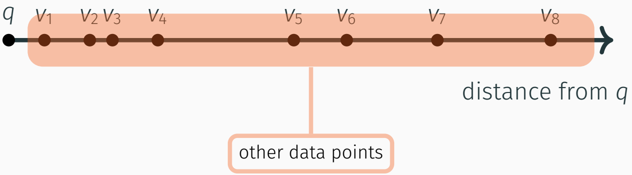
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



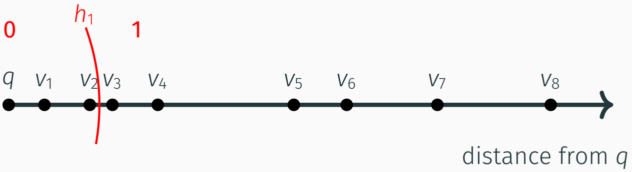
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



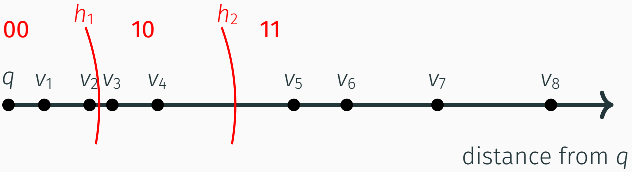
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



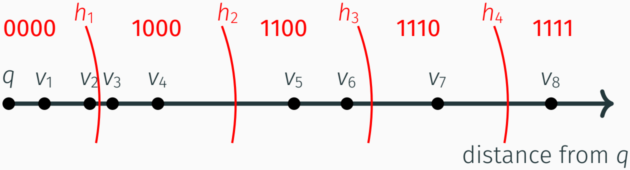
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



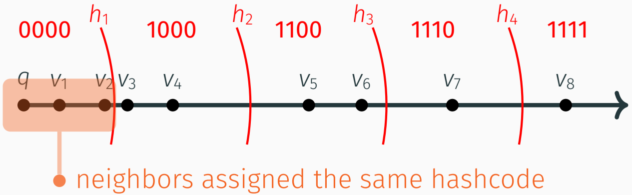
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



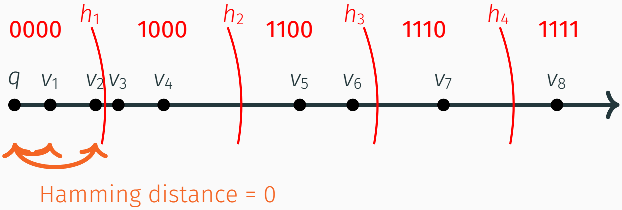
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



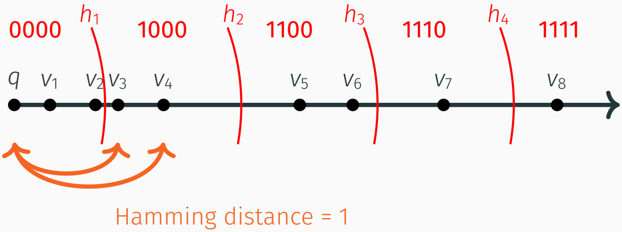
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



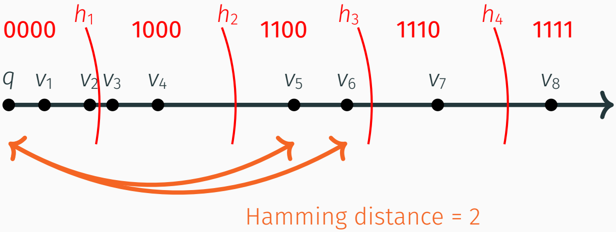
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



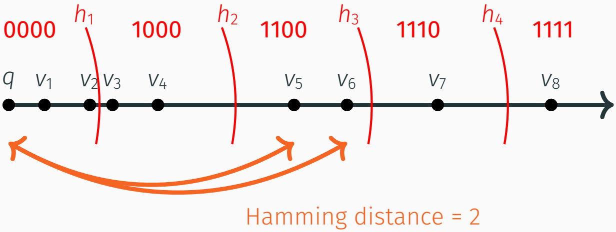
Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



Hashcodes Generation for LSH

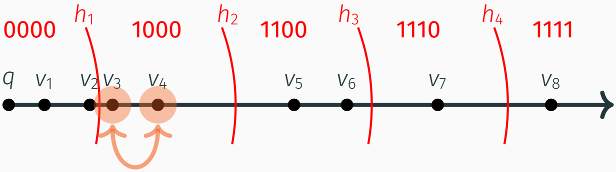
Suppose LSH generates hashcodes of length 4.



Hashcodes as a proxy

Hashcodes Generation for LSH

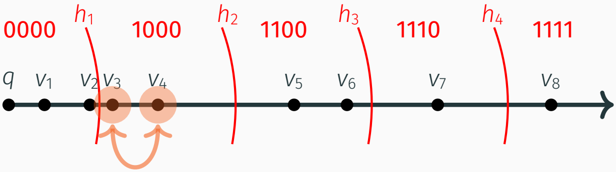
Suppose LSH generates hashcodes of length 4.



We can't distinguish these two

Hashcodes Generation for LSH

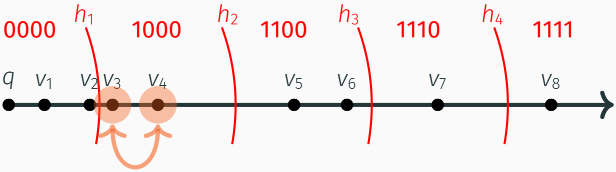
Suppose LSH generates hashcodes of length 4.



We can't distinguish these two
$\rightarrow$ For 3-NN, approximate

Hashcodes Generation for LSH

Suppose LSH generates hashcodes of length 4.



We can't distinguish these two
$\rightarrow$ For 3-NN, approximate

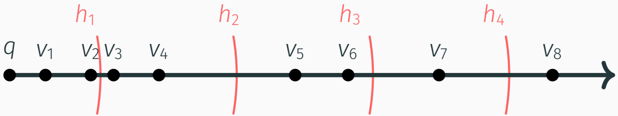
Motivation

A new scheme able to distinguish v_3 and v_4
based on their hashcodes?

Outline

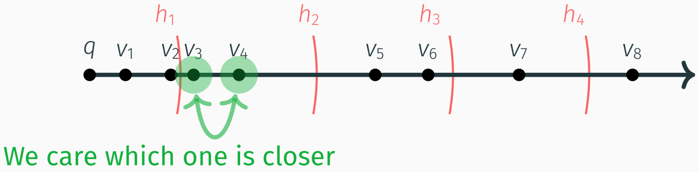
Neighbor-Sensitive Hashing Intuition

We are interested in 3-NN. Hash functions by LSH.



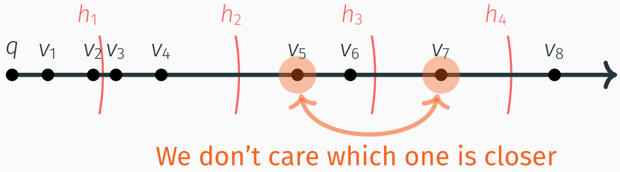
Neighbor-Sensitive Hashing Intuition

We are interested in 3-NN. Hash functions by LSH.



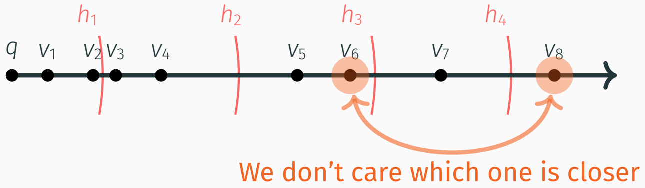
Neighbor-Sensitive Hashing Intuition

We are interested in 3-NN. Hash functions by LSH.



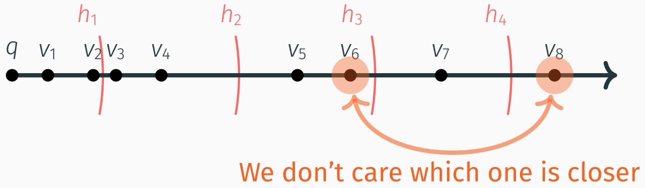
Neighbor-Sensitive Hashing Intuition

We are interested in 3-NN. Hash functions by LSH.



Neighbor-Sensitive Hashing Intuition

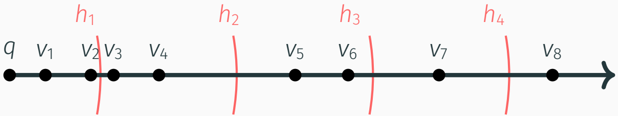
We are interested in 3-NN. Hash functions by LSH.



Observation: h_3 and h_4 are wasted (for 3-NN).

Neighbor-Sensitive Hashing Intuition

We are interested in 3-NN. Hash functions by LSH.



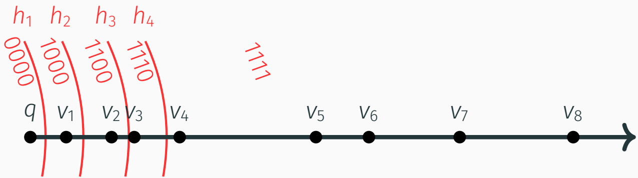
Observation: h_3 and h_4 are wasted (for 3-NN).

Our Idea

Generating hash functions **close to the query**
so that we can **better distinguish** the close items.

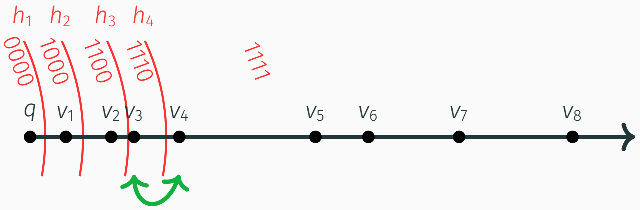
Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.



Neighbor-Sensitive Hashing Intuition (cont'd)

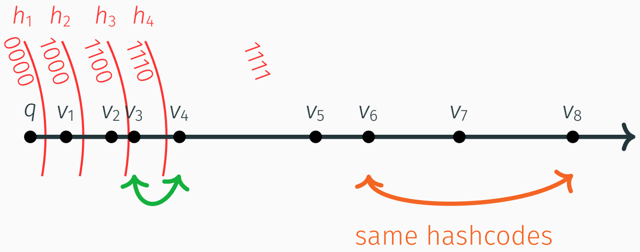
Suppose we could (somehow) generate hash functions in this way.



We could **distinguish v_3 and v_4** based on their hashcodes.
(thus, able to solve 3-NN accurately)

Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.

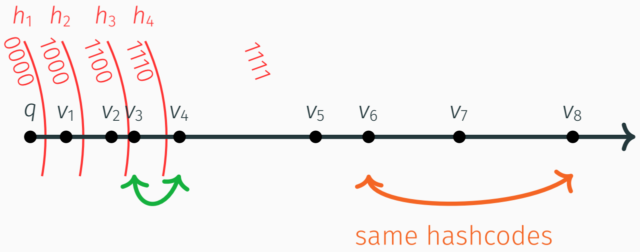


We could **distinguish v_3 and v_4** based on their hashcodes.
(thus, able to solve 3-NN accurately)

Note: **Could not** distinguish v_6 and v_8 based on their hashcodes.

Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.



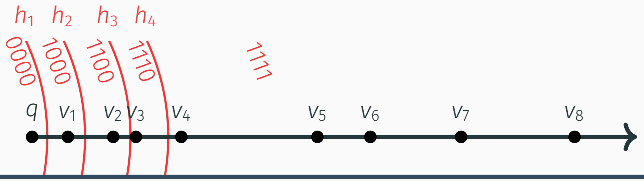
We could **distinguish v_3 and v_4** based on their hashcodes.
(thus, able to solve 3-NN accurately)

Note: **Could not** distinguish v_6 and v_8 based on their hashcodes.

Not an issue for 3-NN

Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.



We
(th

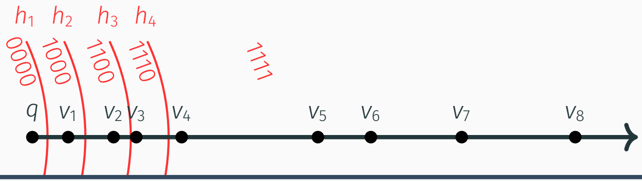
No

Difference in NSH's Intuition:

A decade of existing work: small Hamming distance between close data items

Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.



We

(th

No

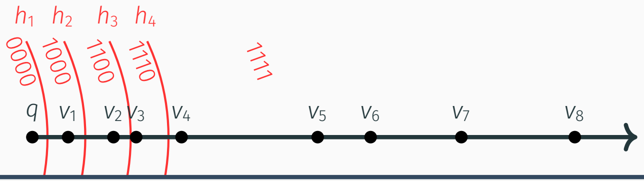
Difference in NSH's Intuition:

A decade of existing work: small Hamming distance between close data items

NSH: larger Hamming distance between close items

Neighbor-Sensitive Hashing Intuition (cont'd)

Suppose we could (somehow) generate hash functions in this way.



We

(th

No

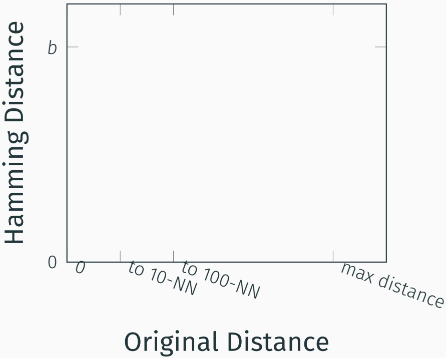
Difference in NSH's Intuition:

A decade of existing work: small Hamming distance between close data items

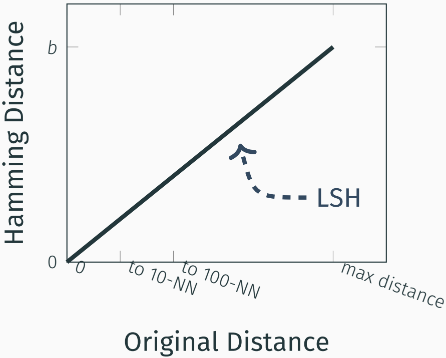
NSH: larger Hamming distance between close items

Seemingly **counter-intuitive**; however, our paper proves that larger Hamming distance leads to higher accuracy in general.

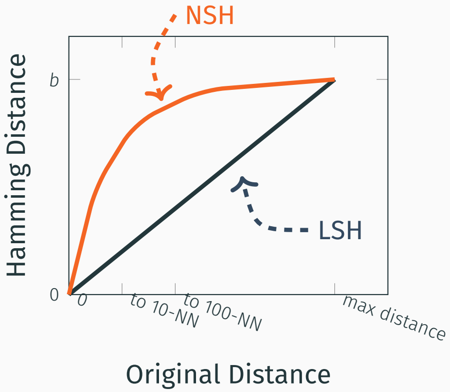
Important Difference between LSH and NSH



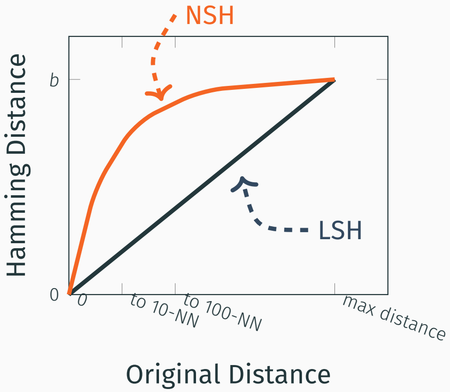
Important Difference between LSH and NSH



Important Difference between LSH and NSH

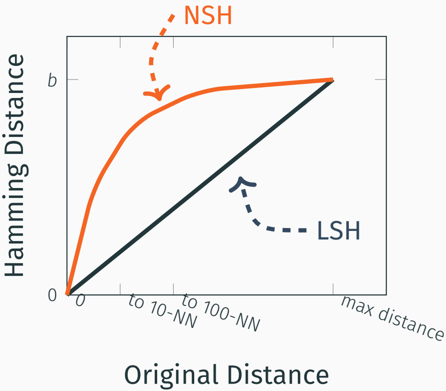


Important Difference between LSH and NSH



A larger slope indicates higher distinguishing-power based on hashcodes.

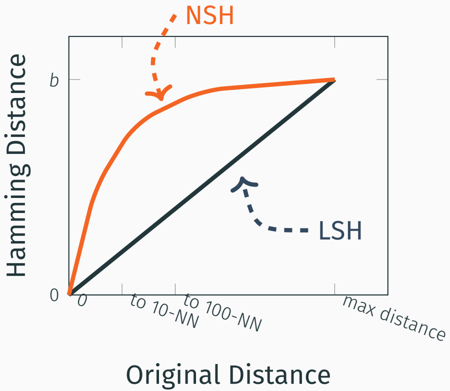
Important Difference between LSH and NSH



A larger slope indicates higher distinguishing-power based on hashcodes.

LSH: **uniform** distinguishing-power over all distance ranges.

Important Difference between LSH and NSH

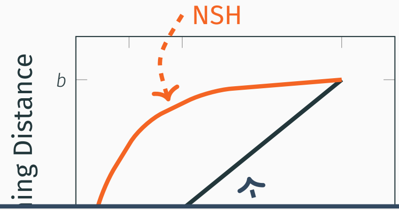


A larger slope indicates higher distinguishing-power based on hashcodes.

LSH: **uniform** distinguishing-power over all distance ranges.

NSH: Higher distinguishing-power for the points
that are close each other.

Important Difference between LSH and NSH



Key Challenge

How to enlarge the Hamming distances *selectively* for close data items?

Original Distance

A larger slope indicates higher distinguishing-power based on hashcodes.

LSH: **uniform** distinguishing-power over all distance ranges.

NSH: Higher distinguishing-power for the points that are close each other.

Outline

Neighbor-Sensitive Hashing Overview

Neighbor-Sensitive Hashing Overview

Transform data points to expand the space around the query.

(before generating hash functions)

Neighbor-Sensitive Hashing Overview

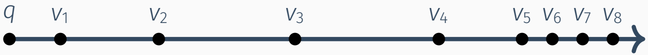
Transform data points to expand the space around the query.

(before generating hash functions)

Neighbor-Sensitive Hashing Overview

Transform data points to expand the space around the query.

(before generating hash functions)

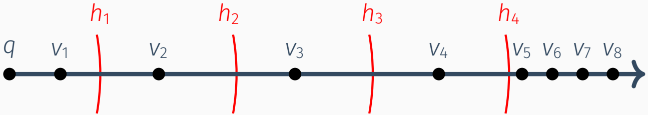


We call this new space the transformed space.

Neighbor-Sensitive Hashing Overview

Transform data points to expand the space around the query.

(before generating hash functions)



We call this new space the transformed space.

Then, generate hash functions on this transformed space.

(thus, convert data points to hashcodes accordingly)

Neighbor-Sensitive Hashing Overview

Key Questions

How can we expand the space around a query?

Then, generate hash functions on this transformed space.

(thus, convert data points to hashcodes accordingly)

Neighbor-Sensitive Hashing Overview

Key Questions

How can we expand the space around a query?

Is it easier if we know the query *a priori* ?

Then, generate hash functions on this transformed space.

(thus, convert data points to hashcodes accordingly)

Neighbor-Sensitive Hashing Overview

Key Questions

How can we expand the space around a query?

Is it easier if we know the query *a priori* ?

How can we expand the space around an arbitrary query?

Then, generate hash functions on this transformed space.

(thus, convert data points to hashcodes accordingly)

Neighbor-Sensitive Transformation

We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).

(e.g., $f(v_1)$, $f(v_2)$, ...)

We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).
(e.g., $f(v_1)$, $f(v_2)$, ...)

Visual illustration of NST

Neighbor-Sensitive Transformation

We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).

(e.g., $f(v_1)$, $f(v_2)$, ...)

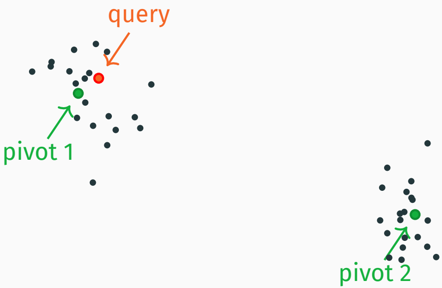
Visual illustration of NST

Neighbor-Sensitive Transformation

We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).

(e.g., $f(v_1)$, $f(v_2)$, ...)

Visual illustration of NST



Neighbor-Sensitive Transformation

We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).

(e.g., $f(v_1)$, $f(v_2)$, ...)

Visual illustration of NST

Neighbor-Sensitive Transformation

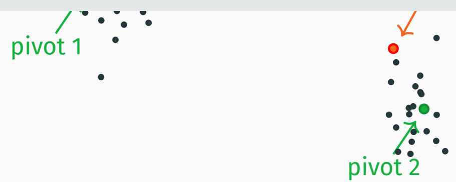
We expand the space around an *arbitrary query* using our proposed Neighbor-Sensitive Transformation (NST).

(e.g., $f(v_1)$, $f(v_2)$, ...)

Visual illustration of NST

Our Formal Claim (Theorem 2)

Using NST with regular hash functions produces higher accuracy than LSH.

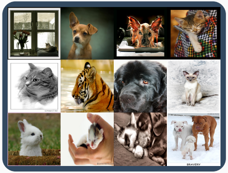


Big Picture: NSH Workflow

Offline Processing

Big Picture: NSH Workflow

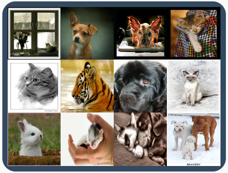
Offline Processing



Original Database

Big Picture: NSH Workflow

Offline Processing



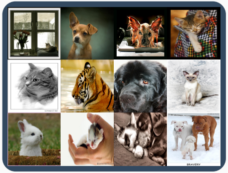
Original Database

NST →

Transformed DB

Big Picture: NSH Workflow

Offline Processing

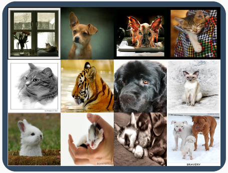


Original Database

NST → Transformed DB → hash → Hashed DB

Big Picture: NSH Workflow

Offline Processing



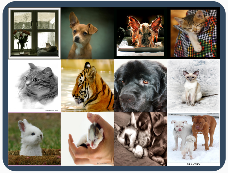
Original Database

NST → Transformed DB → hash → Hashed DB

Online Processing

Big Picture: NSH Workflow

Offline Processing



Original Database

NST → Transformed DB

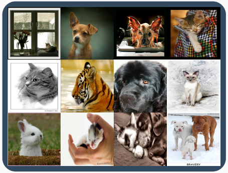
hash → Hashed DB

Online Processing

Original Query

Big Picture: NSH Workflow

Offline Processing



Original Database

NST → Transformed DB — hash → Hashed DB

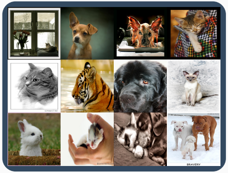
Online Processing

Original Query

NST → Transformed Query

Big Picture: NSH Workflow

Offline Processing



Original Database

NST → Transcribed DB → hash → Hashed DB

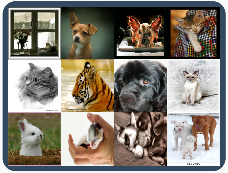
Online Processing

Original Query

NST → Transformed Query → hash → Hashed Query

Big Picture: NSH Workflow

Offline Processing



Original Database

NST → Transformed DB → hash → Hashed DB

Search

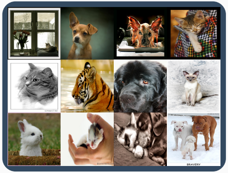
Online Processing

Original Query

NST → Transformed Query → hash → Hashed Query

Big Picture: NSH Workflow

Offline Processing



Original Database

NST →

Transformed DB

LSH Workflow

hash →

Hashed DB

Search ↑

Online Processing

Original Query

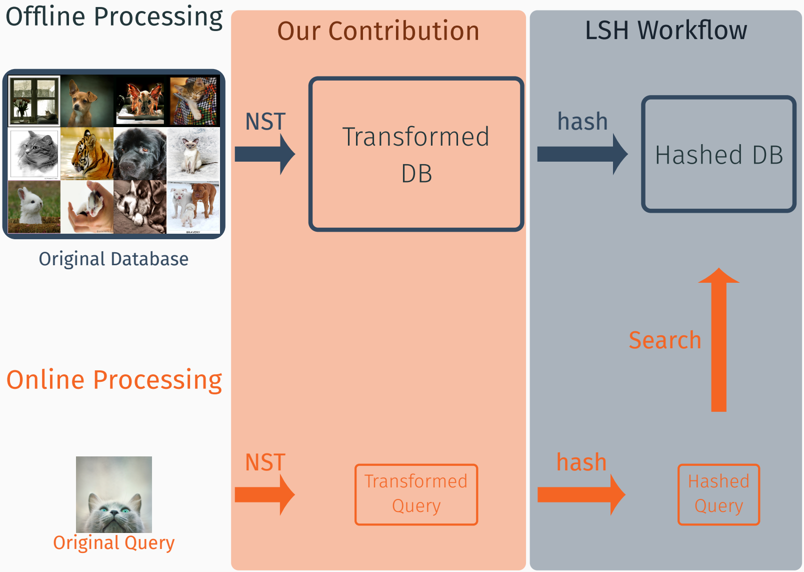
NST →

Transformed Query

hash →

Hashed Query

Big Picture: NSH Workflow



Neighbor-Sensitive Hashing Visualized

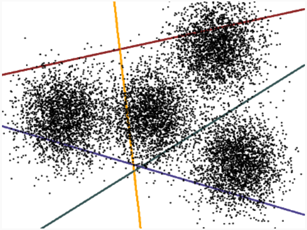
We visualized the hash functions
in the original space.

Dataset: five 2D normal distributions, Generated 4 hash functions

Hash functions for NSH were generated in the transformed space.

Neighbor-Sensitive Hashing Visualized

We visualized the hash functions
in the original space.



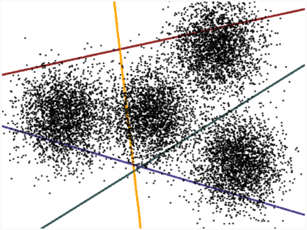
LSH

Dataset: five 2D normal distributions, Generated 4 hash functions

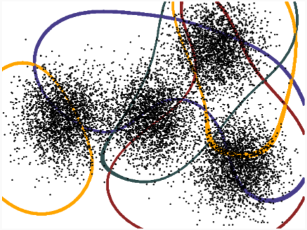
Hash functions for NSH were generated in the transformed space.

Neighbor-Sensitive Hashing Visualized

We visualized the hash functions
in the original space.



LSH



NSH

Dataset: five 2D normal distributions, Generated 4 hash functions

Hash functions for NSH were generated in the transformed space.

Experiment Setup

Quality Metric

$$recall(k)@r = \frac{(\#\text{ of true }k\text{NN in the retrieved})}{k} \times 100.$$

Experiment Setup

Quality Metric

$$recall(k)@r = \frac{(\text{\# of true } k\text{NN in the retrieved})}{k} \times 100.$$

Note: Higher recall means either

(i) more accurate searching for the same time budget, or

(ii) faster searching for the same target recall.

Experiment Setup

Quality Metric

$$recall(k)@r = \frac{(\text{\# of true } k\text{NN in the retrieved})}{k} \times 100.$$

Note: Higher recall means either

(i) more accurate searching for the same time budget, or

(ii) faster searching for the same target recall.

Compared Methods: three well-known and five state-of-the-arts

Experiment Setup

Quality Metric

$$recall(k)@r = \frac{(\text{\# of true } k\text{NN in the retrieved})}{k} \times 100.$$

Note: Higher recall means either

(i) **more accurate searching** for the same time budget, or

(ii) **faster searching** for the same target recall.

Compared Methods: three well-known and five state-of-the-arts

1. Locality Sensitive Hashing (LSH) [Datar et al., 2004]
2. Spectral Hashing (SH) [Weiss et al., 2009]
3. Spherical Hashing (SpH) [Heo et al., 2012]
4. Data Sensitive Hashing (CPH) [Gao et al., 2014]
5. Anchor Graph Hashing (AGH) [Liu et al., 2011]
6. Compressed Hashing (CH) [Lin et al., 2013]
7. Complementary Projection Hashing (CPH) [Jin et al., 2013]
8. Kernelized Supervised Hashing (KSH) [Kulis and Grauman, 2012]

Experiment Setup

Quality Metric

$$recall(k)@r = \frac{(\#\text{ of true }k\text{NN in the retrieved})}{k} \times 100.$$

Note: Higher recall means either

(i) more accurate searching for the same time budget, or

(ii) faster searching for the same target recall.

Compared Methods: three well-known and five state-of-the-arts

1. Locality Sensitive Hashing (LSH) [Datar et al., 2004]
2. Sp

Involves data transformations for different purposes

3. Spherical Hashing (SpH) [Heo et al., 2012]
4. Data Sensitive Hashing (CPH) [Gao et al., 2014]
5. Anchor Graph Hashing (AGH) [Liu et al., 2011]
6. Compressed Hashing (CH) [Lin et al., 2013]
7. Complementary Projection Hashing (CPH) [Jin et al., 2013]
8. Kernelized Supervised Hashing (KSH) [Kulis and Grauman, 2012]

Experimental Claim and Datasets

Our Experimental Claim:

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved “larger Hamming distances between close data items”

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved “larger Hamming distances between close data items”
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved “larger Hamming distances between close data items”
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.
3. NSH showed **faster search speed** (for target recalls) than compared methods.

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved "larger Hamming distances between close data items"
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.
3. NSH showed **faster search speed** (for target recalls) than compared methods.
4. NSH's hash function generation was reasonably **fast**.

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved "larger Hamming distances between close data items"
2. NSH showed higher recalls (for fixed hashcode sizes) than compared methods.
3. NSH showed faster search speed (for target recalls) than compared methods.
4. NSH's hash function generation was reasonably fast.

Real-World Datasets:

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved “larger Hamming distances between close data items”
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.
3. NSH showed **faster search speed** (for target recalls) than compared methods.
4. NSH’s hash function generation was reasonably **fast**.

Real-World Datasets:

1. MNIST: 69K hand-written digit images

Experimental Claim and Datasets

Our Experimental Claim:

1. NSH achieved "larger Hamming distances between close data items"
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.
3. NSH showed **faster search speed** (for target recalls) than compared methods.
4. NSH's hash function generation was reasonably **fast**.

Real-World Datasets:

1. MNIST: 69K hand-written digit images
2. TINY: 80 million image (GIST) descriptors

Experimental Claim and Datasets

Our Experimental Claim:

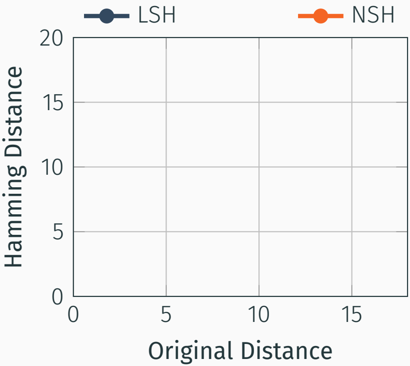
1. NSH achieved "larger Hamming distances between close data items"
2. NSH showed **higher recalls** (for fixed hashcode sizes) than compared methods.
3. NSH showed **faster search speed** (for target recalls) than compared methods.
4. NSH's hash function generation was reasonably **fast**.

Real-World Datasets:

1. MNIST: 69K hand-written digit images
2. TINY: 80 million image (GIST) descriptors
3. SIFT: 50 million image (SIFT) descriptors

Neighbor-Sensitive Hashing Property

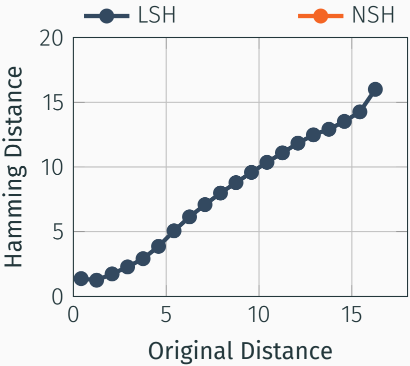
We measured the relationship between
(i) the **original distances** between pairs of original data items,
(ii) the **Hamming distance** between pairs of hashcodes.



Dataset: MNIST

Neighbor-Sensitive Hashing Property

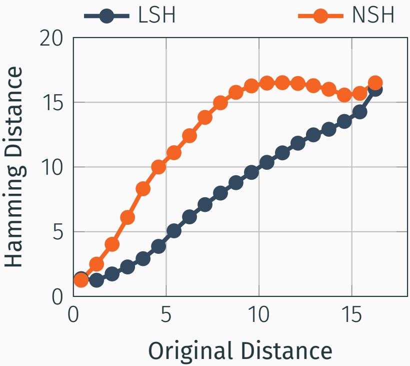
We measured the relationship between
(i) the **original distances** between pairs of original data items,
(ii) the **Hamming distance** between pairs of hashcodes.



Dataset: MNIST

Neighbor-Sensitive Hashing Property

We measured the relationship between
(i) the **original distances** between pairs of original data items,
(ii) the **Hamming distance** between pairs of hashcodes.



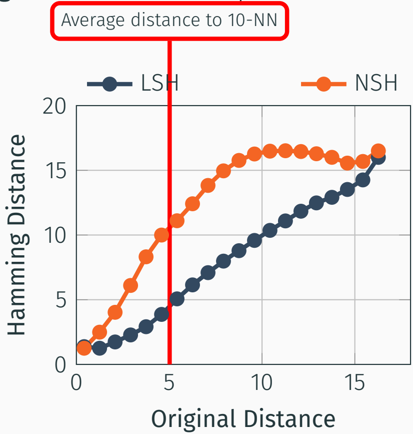
Dataset: MNIST

Neighbor-Sensitive Hashing Property

We measured the relationship between

(i) the **original distances** between pairs of original data items,

(ii) the **Hamming distance** between pairs of hashcodes.



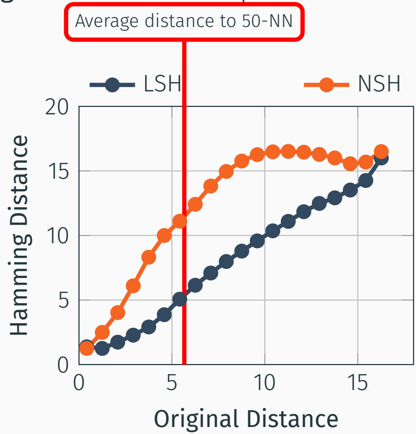
Dataset: MNIST

Neighbor-Sensitive Hashing Property

We measured the relationship between

(i) the **original distances** between pairs of original data items,

(ii) the **Hamming distance** between pairs of hashcodes.



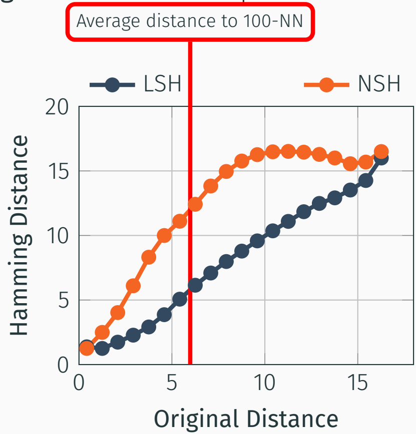
Dataset: MNIST

Neighbor-Sensitive Hashing Property

We measured the relationship between

(i) the **original distances** between pairs of original data items,

(ii) the **Hamming distance** between pairs of hashcodes.



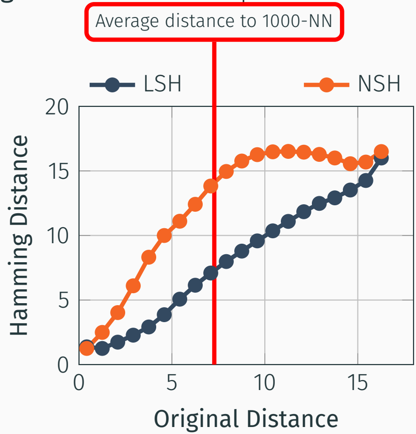
Dataset: MNIST

Neighbor-Sensitive Hashing Property

We measured the relationship between

(i) the **original distances** between pairs of original data items,

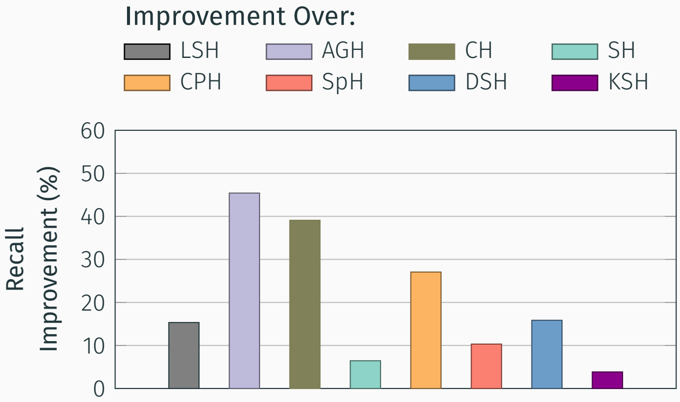
(ii) the **Hamming distance** between pairs of hashcodes.



Dataset: MNIST

Recall Improvement for Fixed Hashcode Size

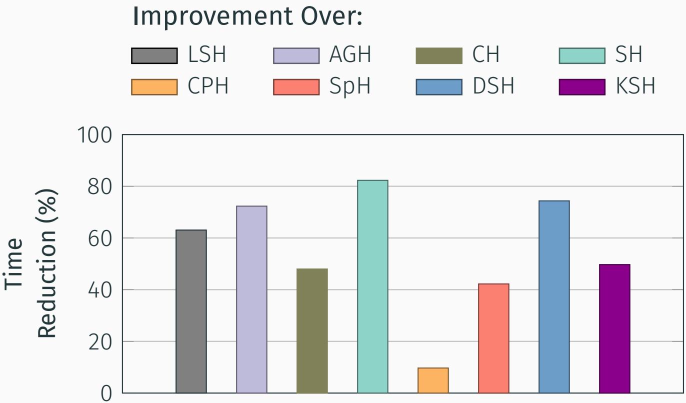
Compared search accuracy of 9 different methods (including NSH).



Dataset: TINY, Hashcode size: 64 bits

Time Reduction for Fixed Recall

Measured search time of 9 different methods (including NSH).



Dataset: SIFT, Hashcode size: 64 bits, Target recall: 50%

Offline Computation Time

Method	Hash Function Generation (sec)		Hashcode Generation (min)	
	32bit	64bit	32bit	64bit
LSH	0.38	0.29	22	23
SH	28	36	54	154
AGH	786	873	105	95
SpH	397	875	18	23
CH	483	599	265	266
CPH	34,371	63,398	85	105
DSH	3.14	1.48	24	23
KSH	2,028	3,502	24	29
NSH (Ours)	231	284	37	46

Dataset: TINY

Offline Computation Time

Method	Hash Function Generation (sec)		Hashcode Generation (min)	
	32bit	64bit	32bit	64bit
LSH	0.38	0.29	22	23
SH	28	36	54	154
AGH	786	873	105	95
SpH	397	875	18	23
CH	483	599	265	266
CPH	34,371	63,398	85	105
DSH	3.14	1.48	24	23
KSH	2,028	3,502	24	29
NSH (Ours)	231	284	37	46

Some learning-based methods (e.g., CPH, KSH) were extremely slow.

Dataset: TINY

Offline Computation Time

Method	Hash Function Generation (sec)		Hashcode Generation (min)	
	32bit	64bit	32bit	64bit
LSH	0.38	0.29	22	23
SH	28	36	54	154
AGH	786	873	105	95
SpH	397	875	18	23
CH	483	599	265	266
CPH	34,371	63,398	85	105
DSH	3.14	1.48	24	23
KSH	2,028	3,502	24	29
NSH (Ours)	231	284	37	46

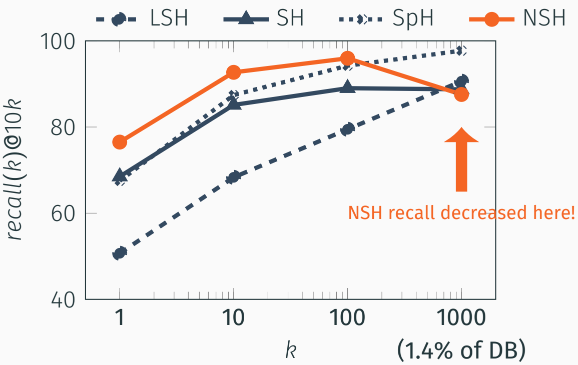
Some learning-based methods (e.g., CPH, KSH) were extremely slow.

Our method was among the fastest.

Dataset: TINY

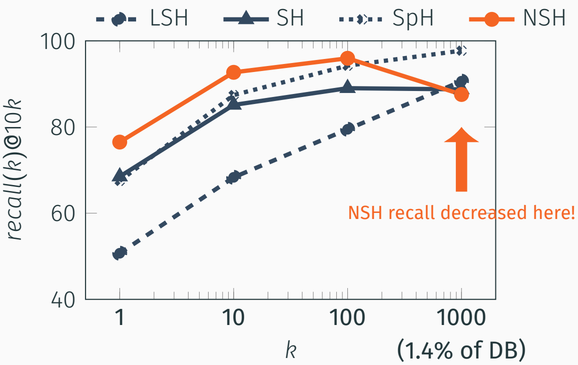
Neighbor-Sensitive Effect

NSH was more effective for relatively small k.



Neighbor-Sensitive Effect

NSH was more effective for relatively small k.



With a **bigger** dataset,
this “recall-dropping effect” was not observed.

Conclusion

1. We have formally shown that counter-intuitive idea can lead to improved kNN accuracy.

Conclusion

1. We have formally shown that counter-intuitive idea can lead to improved kNN accuracy.

2. Based on the idea, we have proposed a novel hashing-based search method—Neighbor-Sensitive Hashing.

Conclusion

1. We have formally shown that counter-intuitive idea can lead to improved kNN accuracy.

2. Based on the idea, we have proposed a novel hashing-based search method—Neighbor-Sensitive Hashing.

3. We have empirically demonstrated that our proposed method could achieve better kNN performance (faster or more accurate) compared to existing methods.

Thank You!

References I

Charikar, M. S. (2002).
Similarity estimation techniques from rounding algorithms.
In *SOTC*.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004).
Locality-sensitive hashing scheme based on p-stable distributions.
In *SoCG*.

Gao, J., Jagadish, H. V., Lu, W., and Ooi, B. C. (2014).
Dsh: data sensitive hashing for high-dimensional k-nnsearch.
In *SIGMOD*.

Heo, J.-P., Lee, Y., He, J., Chang, S.-F., and Yoon, S.-E. (2012).
Spherical hashing.
In *CVPR*.

References II

Jin, Z., Hu, Y., Lin, Y., Zhang, D., Lin, S., Cai, D., and Li, X. (2013).
Complementary projection hashing.
In *ICCV*.

Kulis, B. and Grauman, K. (2012).
Kernelized locality-sensitive hashing.
TPAM.

Lin, Y., Jin, R., Cai, D., Yan, S., and Li, X. (2013).
Compressed hashing.
In *CVPR*.

Liu, W., Wang, J., Kumar, S., and Chang, S.-F. (2011).
Hashing with graphs.
In *ICML*.

Weiss, Y., Torralba, A., and Fergus, R. (2009).
Spectral hashing.
In *NIPS*.

We expand the space around a query using
Neighbor-Sensitive Transformation.

Backup: Neighbor-Sensitive Transformation

We expand the space around a query using Neighbor-Sensitive Transformation.

A Neighbor-Sensitive Transformation (NST) is simply a function of data points (e.g., $f(v_1), f(v_2), \ldots$)

Backup: Neighbor-Sensitive Transformation

We expand the space around a query using
Neighbor-Sensitive Transformation.

A Neighbor-Sensitive Transformation (NST) is simply
a function of data points
(e.g., $f(v_1), f(v_2), \ldots$)

The function $f(\cdot)$ qualifies for NST
if $f(\cdot)$ satisfies some technical requirements.
(please see our paper)

Backup: Neighbor-Sensitive Transformation

We expand the space around a query using
Neighbor-Sensitive Transformation.

A Neighbor-Sensitive Transformation (NST) is simply
a function of data points
(e.g., $f(v_1), f(v_2), \ldots$)

The function $f(\cdot)$ qualifies for NST
if $f(\cdot)$ satisfies some technical requirements.
(please see our paper)

Our Formal Claim (Theorem 2)

Using NST with regular hash functions produces
higher accuracy than LSH.

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Then, the following function works as NST for q:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right)$$

where **pivot** p is a data point close to q.

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Then, the following function works as NST for q:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right)$$

where **pivot** p is a data point close to q.

We formally prove that this function is NST

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Then, the following function works as NST for q:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right)$$

where **pivot** p is a data point close to q.

We formally prove that this function is NST

Of course, q is unknown *a priori*

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Then, the following function works as NST for q:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right)$$

where **pivot** p is a data point close to q.

We formally prove that this function is NST

Of course, q is unknown *a priori*

Then, the following function is NST for arbitrary q:

$$f(v) = (f_{p_1}(v), \ldots, f_{p_m}(v))$$

with multiple pivots $p_1, \ldots, p_m$.

Backup: Neighbor-Sensitive Transformation (cont'd)

Let us assume a query q is known

Then, the following function works as NST for q:

$$f_p(v) = \exp\left(-\frac{\|p - v\|^2}{\eta^2}\right)$$

where **pivot p** is a data point close to q.

We formally prove that this function is NST

Of course, q is unknown *a priori*

Then, the following function is NST for arbitrary q:

$$f(v) = (f_{p_1}(v), \ldots, f_{p_m}(v))$$

with multiple pivots $p_1, \ldots, p_m$.

We don't formally prove, but show empirically that this is NST for arbitrary queries.