Multimedia Technology

Lecture 9: Nearest Neighbor Search

Lecturer: *Dr*. Wan-Lei Zhao

Autumn Semester 2020

Email: wlzhao@xmu.edu.cn,

Outline

Nearest Neighbor Search: an overview (1)

- The need of Fast Nearest Neighbor Search arises from many contexts
  1. Database, e.g. spatial-temporal database
  2. Information Retrieval
  3. Data mining, K-means, DB-SCAN
  4. Image Processing, e.g. segmentation, saliency detection
  5. Network, e.g. routing
- In most of the applications, they require **instant response**
  - **Instant response** means within second

Nearest Neighboor Search: an overview (2)

- Up-to-now, the problem is not well solved
  1. Complexity increases exponentially with the number of dimension
  2. Known as "**curse of dimensionality**"
  3. No general-purpose exact solution in high dimensional Euclidean space
  4. Polynomial preprocessing and polylogarithmic search time
  5. The complexity upper bound is $O(D \cdot N)$
- Linear processing complexity does not meet up with the expectation

Nearest Neighboor Search: an overview (3)

- Both D and N could be very large
  - Photos in Flickr are in billions, $> 3,000$ images uploaded per minutes
  - 120,000,000 videos in YouTube, $> 200,000$ videos uploaded per day
  - Total duration is more than 600 years
- In all these contexts, it requires **instance response** to the user
- We would expect $O(D^{1/c} \cdot logN)$, where $c > 1$

Nearest Neighboor Search: an overview (4)

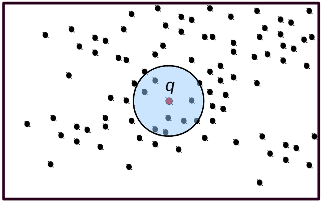
- Based on the size of D and N, the problem can be partitioned into following sub-problems

Low & Dense	**Low & Sparse**
High & Dense	**High & Sparse**

- Dense VS Sparse: there is no clear border between them
- Sparse: the number of non-zero dimensions $\leq$ 10% D
- High VS Low: there is no clear border either
- $\geq$ 10 already very high dimensional
- LD: X-Y, RGB and HSV; LS: Spatial-temporal data, e.g. GIS
- HD: SIFT, VLAD; HS; text document, Bag-of-Visual Word

Nearest Neighboor Search: the problem (1)

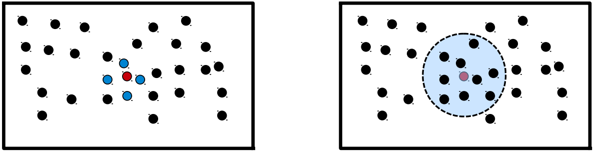
- Given a set of points S in a metric space M and a query point $q \in M$
- Task: try to find nearest neighbors from set S for q



- In most of the practices, the algorithm should be able to return k nearest neighbors (at least the top one)

Nearest Neighboor Search: two types of NNS

- KNN Search
  - NNS algorithm should be able to return k nearest neighbors given any query q (k is arbitrary)
- Range Search
  - NNS algorithm should be able to return nearest neighbors within a radius of the query



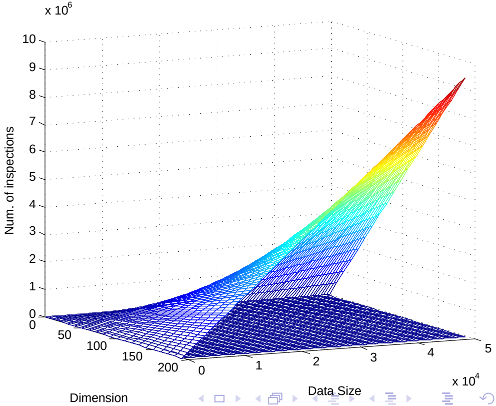
top 4 range search

- In many cases, these two requirements are not necessary and hard to meet

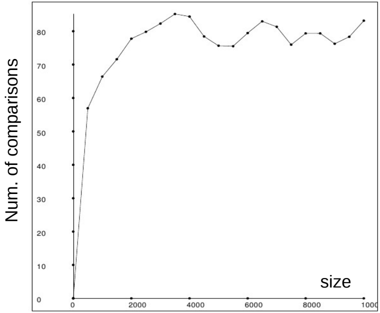
Challenge of Nearest Neighboor Search (1)

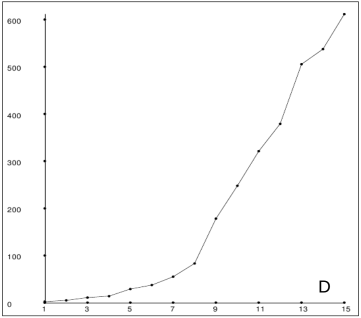
- The complexity (measured by the num. of comparisons) increases exponentially when dimension increases
- We look at the upper bound and lower bound of this problem
  - D: dimensionality; N: Size of data items

- Lower bound: $log(D \cdot N)$
- Upper bound: $D \cdot N$
- Really challenging
- It is clear that there is large space to improve



Challenge of Nearest Neighboor Search (2)





- Performance observations from KD-tree
- Left figure: Num. comparisons VS data size
- Right figure: Num. of comparisons VS num. of dimensions

Distance Measures: the metric spaces

- A metric space consists a pair of (Z, d), Z is a set
- d is a mapping function, which maps $Z \times Z$ (Cartesian product) to R
- $d(.,.)$ is called as a metric or distance function
- Following conditions hold for all x, y, z $\in$ Z
  1. $d(x, y) \geq 0$, known as non-negative
  2. $d(x, y) = 0$ iff x=y
  3. $d(x, y) = d(y, x)$, known as symmetric
  4. $d(x, z) \leq d(x, y) + d(y, z)$, known as triangle inequality

Distance Measures: the norms

- **Norm** is a function defined on a vector space, mapping $v \rightarrow R$, where $v \in R^n$
- Given scalar **a** and vector **u**, **v**, **norm** has following properties:
  1. $p(av) = |a|p(v)$, known as scale invariance
  2. $p(u + v) \leq p(v) + p(u)$, known as triangle inequality
  3. $p(v) = 0$ when v=0
- Given $p \geq 1$, l_p-norm is defined as

$$\|v\|_p = (\sum_{i=1}^{n} |v_i^p|)^{1/p}$$

- Notice that when $p = 2$, it becomes l_2-norm

Euclidean Distance and Euclidean Space

- Euclidean distance:

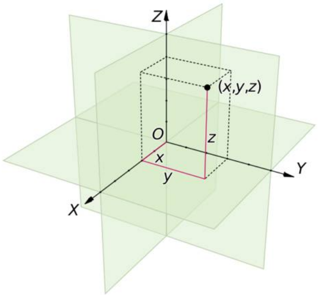
$$d(x, y) = (\sum_{i=1}^{n}(x_i - y_i)^2)^{1/2}$$

- Notice that when $p = 2$, it becomes l_2-norm

- It is scale & translation invariant

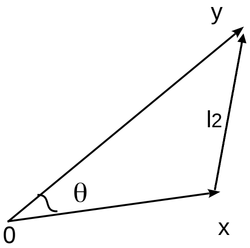
  1. d(x+v, y+v)=d(x,y)
  2. $d(cx, cy) = |c|d(x, y)$

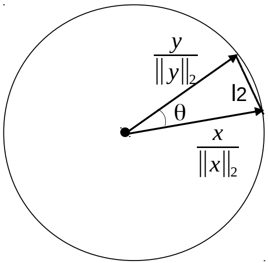
- Check yourself about l_1-norm



Cosine Distance

- *Cosine* distance: $d(x, y) = \frac{x^T y}{\|x\|_2 \|y\|_2}$

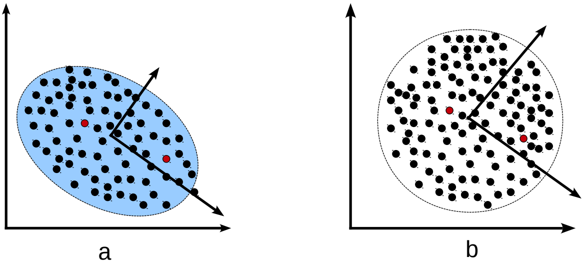




- If x and y are l_2 normalized in advance, *Cosine* distance is equivalent to Euclidean distance (Law of Cosine)

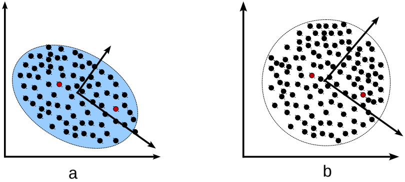
$$d(x, y) = x^T x + y^T y - 2 \cdot x^T y \cdot cos\theta \tag{1}$$

Mahalanobis distance (1)



- For two points in red, do they hold the same distances in Figure a and b?
- Due to the intrinsic data distribution, distances calculation can be biased towards certain data dimension(s)
- Normalize the data distribution can alleviate this issue

Mahalanobis distance (2)

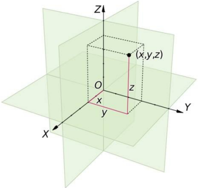


- Given a group of data, the covariance matrix can be estimated
- The eigenvectors coincide with the axis of ellipse, the eigenvalues are treated as normalizing factors
- Mahalanobis distance:

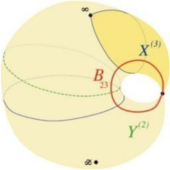
$$d(x, y) = \sqrt{(x-\mu)^T \Sigma^{-1} (y-\mu)}$$

- Mahalanobis measure is used not as frequently as l_2
- One can think of doing PCA before applying l_2 to achieve similar effect

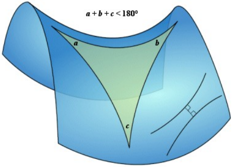
Other Distance Spaces



Euclidean space



Riemann space

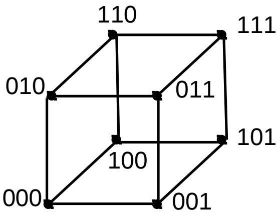


Lobachevsky space

- Euclidean distance cannot reveal the real topology of the spaces shown above
- Currently, there is no effective distance measure for these non Euclidean spaces
- It is hard to work out a universal distance measure
- It is a latent issue in NNS
- l_1 and l_2 are mainly considered in the literature

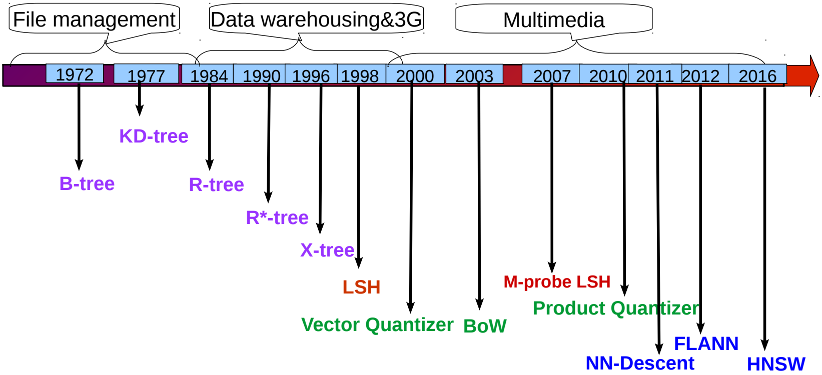
Hamming Distance

- Hamming distance: $d(x, y) = XOR(x, y)$



- Distance measure for binary numbers and strings
- It measures how many substitutions it takes from one string change to another
- The smaller the similar
- It is very efficient
- The **distance space** is much much smaller

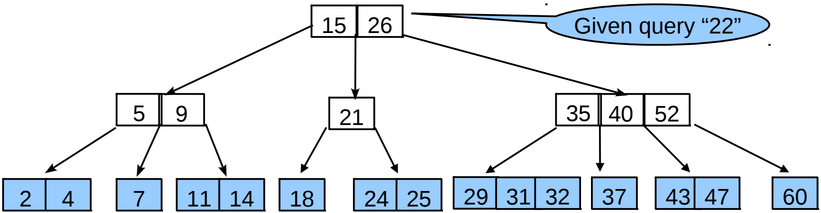
A Spectrum of NNS History



- In the whole 90s and early 00s, researchers were working on "trees"
- The introduction of SIFT and Web 2.0 changes the culture

B-Tree file: a review

- Leaf node keeps all the data items, non-leaf nodes are used for indexing
- For one dimensional data, B-Tree is best solution



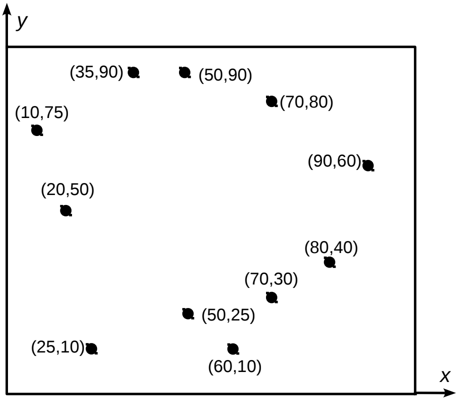
- Online query complexity is $log(N)$
- Does this simplicity still hold when it is extended to $D \geq 2$?

Outline

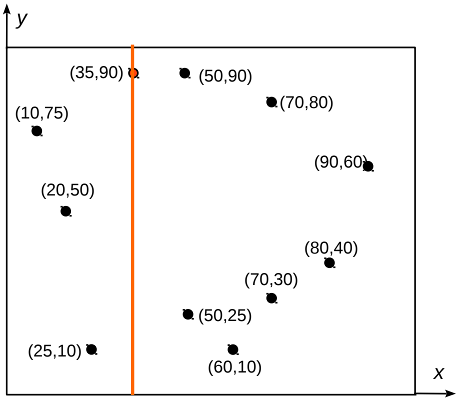
KD Tree: a space partition approach

- K-D Tree means Tree for K-dimensional data points
  - It is a binary tree
  - Designed to index data in multiple dimensions
  - The space complexity is O(N)
  - the time complexity for online search is O(logN) if it is balanced
  - It supports range search and top-k search
- K-D Tree construction procedure:
  1. Choose one of the coordinate as basis to split all rest points into two parts (left child and right child)
  2. Do Step 1 recursively on left and right child until there are no two points in the same node

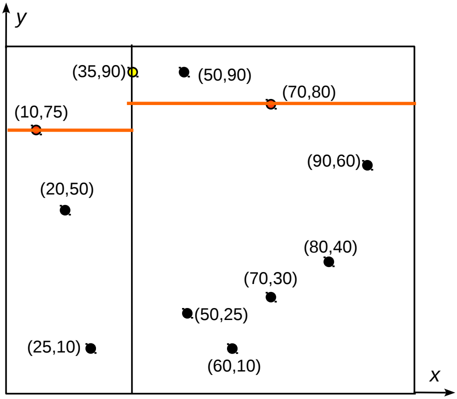
KD-Tree: construction (1)



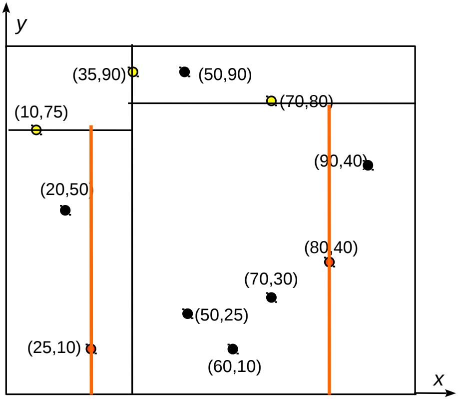
KD-Tree: construction (2)



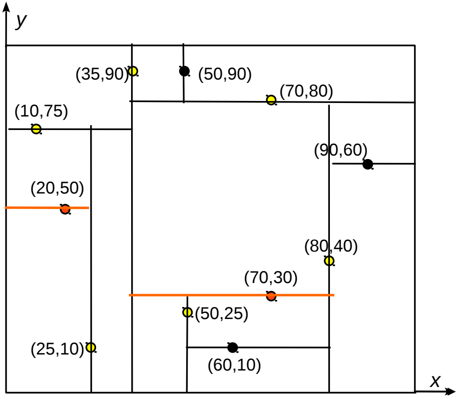
KD-Tree: construction (3)



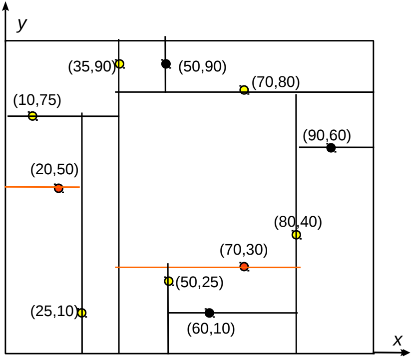
KD-Tree: construction (4)



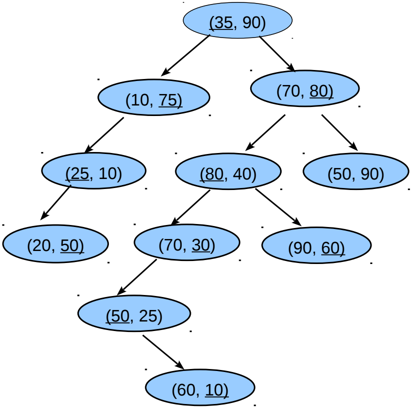
KD-Tree: construction (5)



KD-Tree: construction (6)

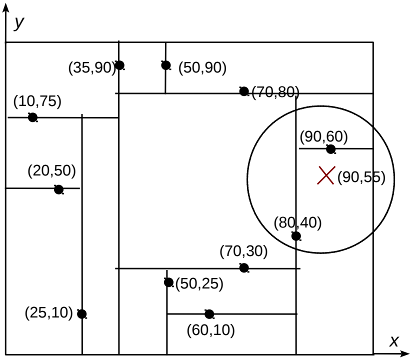


(a)

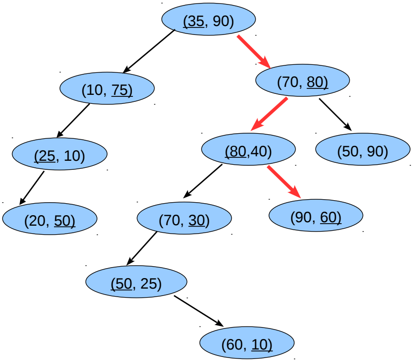


(b)

KD-Tree: query (1)

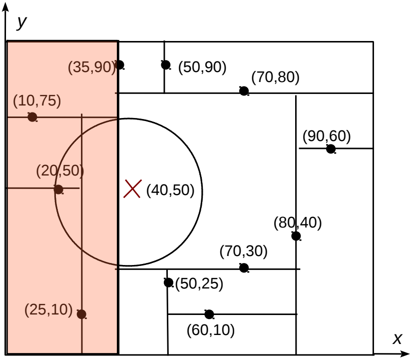


(a)

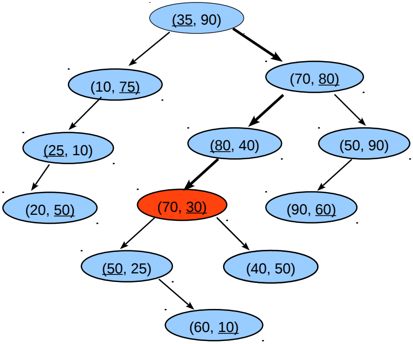


(b)

KD-Tree: query (2)

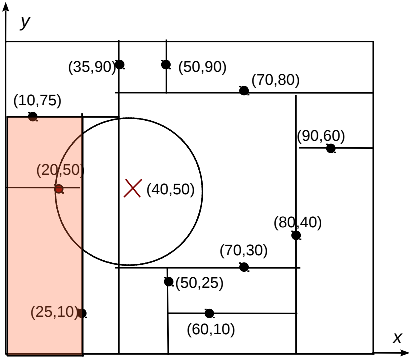


(a)

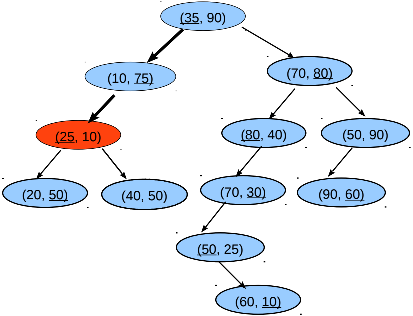


(b)

KD-Tree: query (3)



(a)



(b)

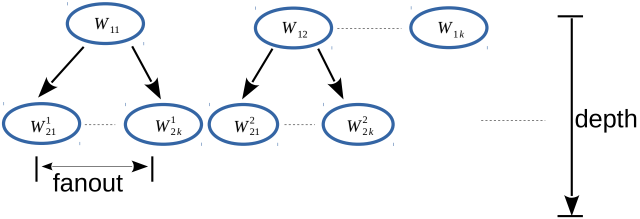
- The retrieval cannot be as efficient as it is supposed to
- It may nearly traverse the whole tree in the worst case
- Balanced KD tree may alleviate the issue
- The reference data is partitioned according to axis differences each time, while NN is measured by l_1 or l_2 norms NN is not necessarily located in the same branch
- Improvement over K-D tree: R-Trees try to group candidate points close to each other (not only in terms of one dimension)

Outline

Overview about FLANN

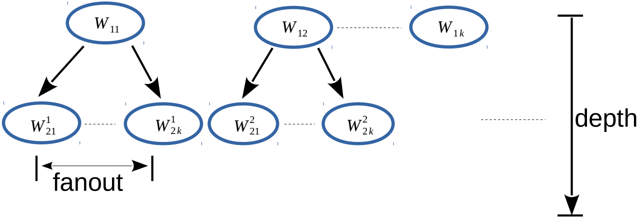
- It becomes popular since 2009
- Proposed by Prof. David G. Lowe and his student
- It achieves 20 - 100 speed-up on high dim. features, e.g. SIFT
- It only returns approximate NNs, say 40 - 60%

Idea of FLANN: hierarchical quantization



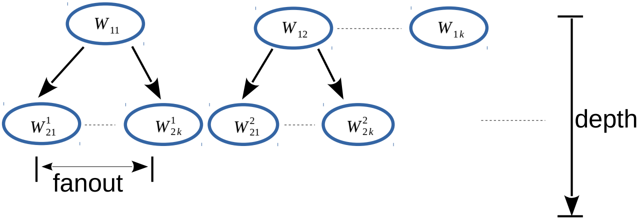
- The hierarchical vocabulary is built by *k*-means
- Two parameters are there, fanout and depth

FLANN: offline indexing



- Each sample is quantized to the closest word in each hierarchy
- Each sample is quantized along one path (from one root to one leaf)

FLANN: online search



1. Query is compared with words of each level
2. Top-k closest candidates are maintained
3. Expand the search to words of next level covered by these closest candidates

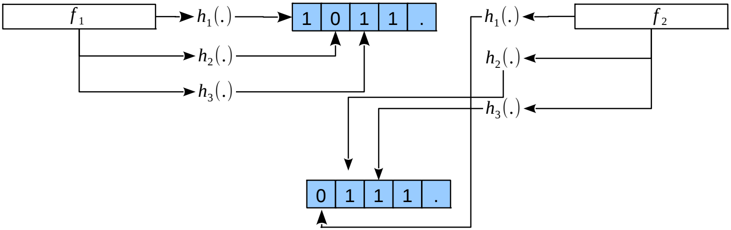
FLANN: comments and suggestions

- The paper has been well cited
- A lot of memories are required to maintain this indexing tree
- It is fast but not precise
- It is OK if you do not require precise NN search

Outline

Randomized NNS approach: the idea

- Idea: generate Hash codes for all data items
- Based on hash function F or hash functions F
- Similar points will have same (or similar) Hash codes
- Key issue: how to define the Hash function(s)



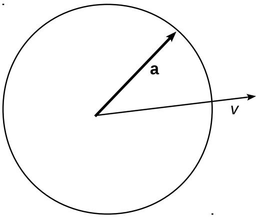
Randomized NNS approach: the procedure

- The steps of producing Hash functions
  1. Draw a random vector **a**
  2. Join **h** to H, which keeps the set of hash functions
  3. Repeat steps 1-2 for L times

- The steps of producing Hash codes
  1. foreach h $\in$ H

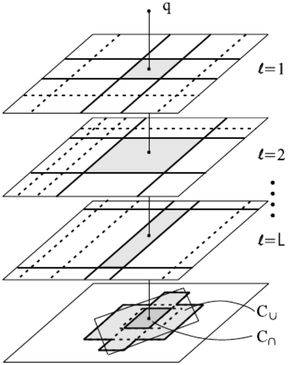
$$c = [\frac{f^T h + b}{W}] \quad 0 < b < W$$

  2. Concatenate all generated c into a binary code



Randomized NNS approach: the query process

- The steps of query
  1. Encode query similarly as before
  2. Compare hash code to all hash codes in reference set
  3. Keep the same one or similar ones as candidate
  4. Compute real distance between query and these candidates
  5. Output the top-k ranked candidates



Randomized NNS approach: the procedure

- Two types of errors: Mismatch and false match
  1. Alleviate mismatch by multiple-probe LSH
  2. Alleviate false match by using more Hash functions (it is a trade-off)
  3. It does not support exact range search and top k search
  4. If your problem doesn't require 100% NN, LSH is an option

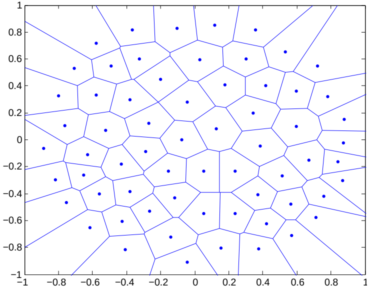
Outline

General criticism on KD-Tree, R-Tree and LSH

- For the approaches discussed so far
- The original data have to be loaded into memory
- This will be a huge burden when we have billions of data items
- We are going to discuss one approach which performs the query on compressed data

Overview about vector quantization

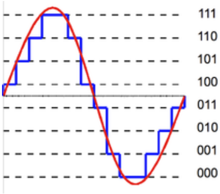
- Let's think about vector quantization first
- Given a 2D points, the vector quantization assigns this point to a Voronoi cell



- As a result, a 2-dimensional point is deconstructed as a Voronoi cell ID (1D)
- The same thing happens when $VQ(x) \rightarrow k$, $dim(x) \geq 2$

Overview about scalar quantization

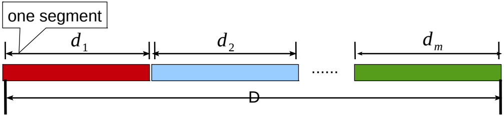
- Now look at another different thing
- The idea is to represent 1D continuous signal with few digital numbers
- Similar thing happens to R, G and B
- Notice that we use 0 - 255 to digitize R, G and B



- This is another case of quantization: scalar quantization

Overview about product quantization (1)

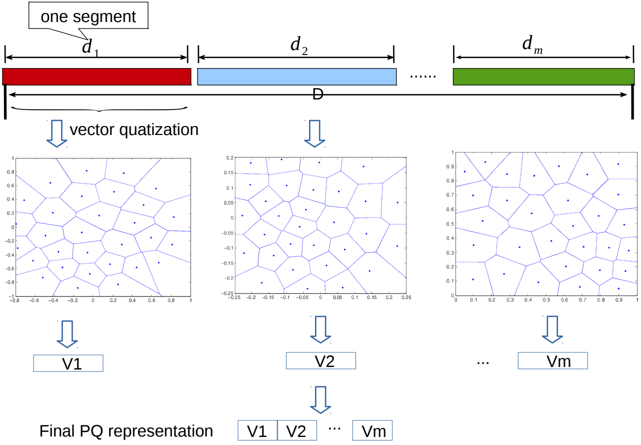
- Product quantizer quantizes segments of one vector
- A D-dimensional vector is partitioned into m segments



- This is something in between scalar quantization and vector quantization

Overview about product quantization (2)

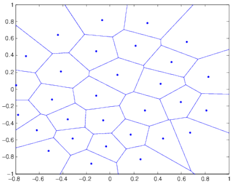
- Quantization is conducted on each segments



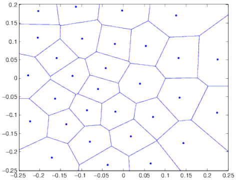
- This results in m integers to represent the original vector
- This is something in between scalar quantization and VQ

Overview about product quantization (3)

- Scalar quantization and VQ can be viewed as special cases of product quantization
- Comparing with original vector, the vector has been compressed
- To perform product quantization, we should build m vocabularies for m sub-spaces

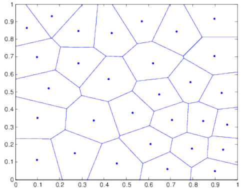


Vocab1



Vocab2

...

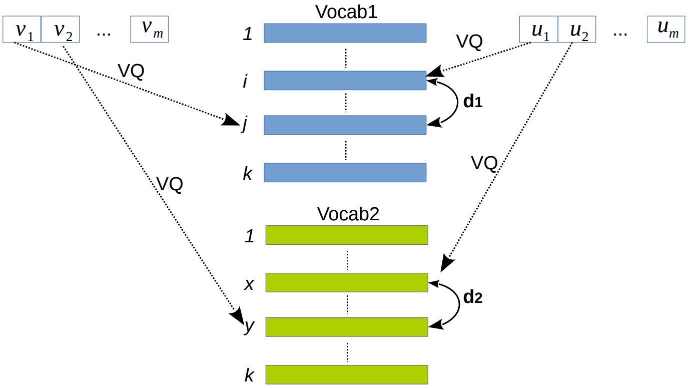


Vocabm

- How we calculate the distance between PQ vectors $[v_1, v_2, \cdots, v_m]$??

Product quantization: symmetric product quantizer (1)

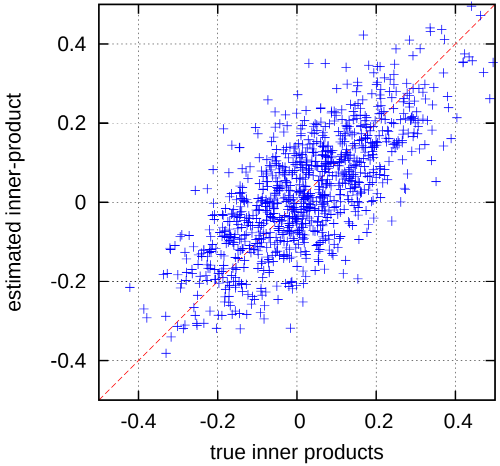
- Product quantizer vocabularies are known
- Distance between two product quantized vectors is $\sum d_i$



- We can build lookup table for each product quantizer vocabulary

Product quantization: symmetric product quantizer (2)

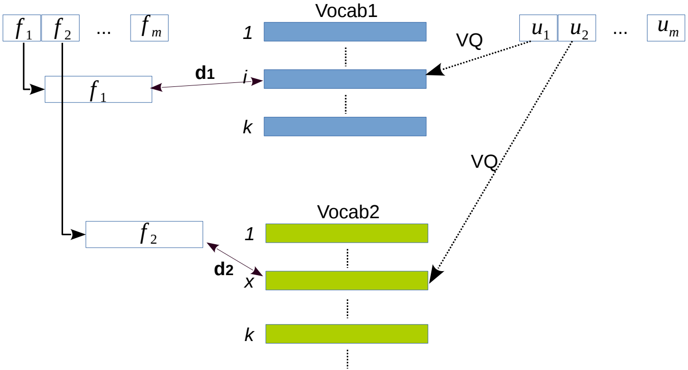
- Distance estimated product quantizer is not precise
- However it is efficient



- We can product quantize all data items in advance

Product quantization: asymmetric product quantizer

- A more precise way is to encode (product-quantizing) the reference side only
- However, it is slower
- $d(.,.) = \sum d_i$

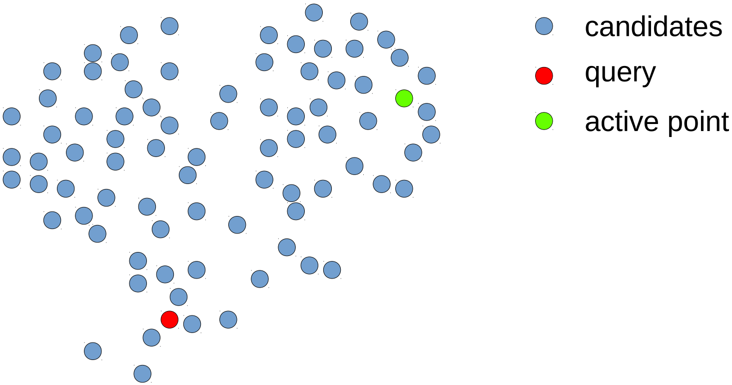


Outline

General criticism on KD-Tree, R-Tree, LSH and PQ

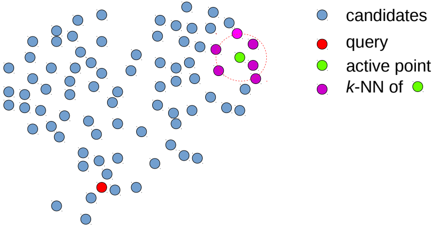
- KD-Tree, R-Tree and LSH are in general slow
- A lot of extra memories are required
- The precision is far below our expectation in the large-scale and high dim. scenario
- PQ is memory efficient, however only return approximate results
- Its precision is not much better than FLANN

NN-Descent: the idea

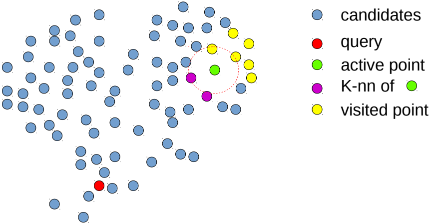


- Given query and the candidate set
- Sample a candidate point randomly
- Climb to the query as much as we can

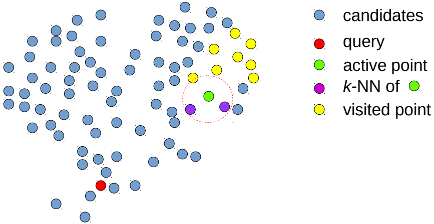
NN-Descent: the procedure (1)



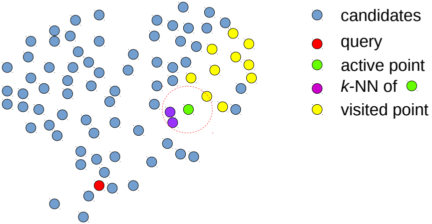
(a) Step-1



(b) Step-2

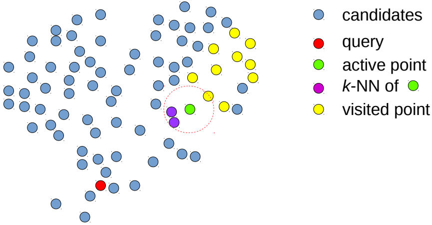


(c) Step-3

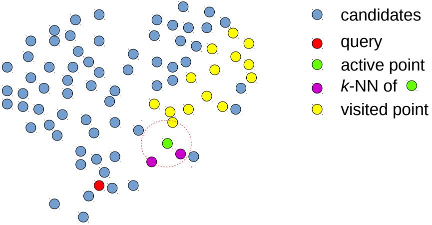


(d) Step-4

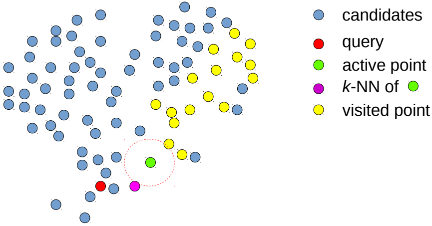
NN-Descent: the procedure (2)



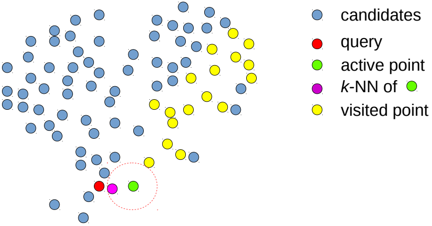
(d) Step-4



(e) Step-5

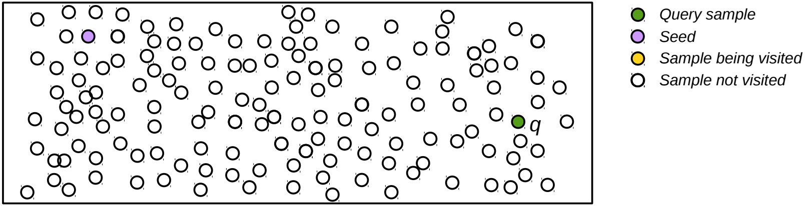


(f) Step-6



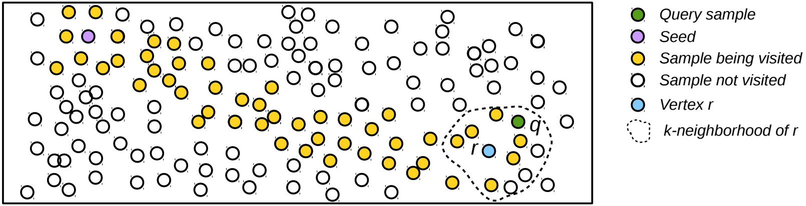
(g) Step-7

NN-Descent: a brief summary (1)



- The starting point (seed) is selected at random
- Scanning the neighborhood of visited point
- Routing towards the query point greedily

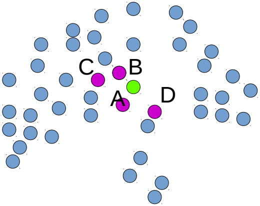
NN-Descent: a brief summary (2)



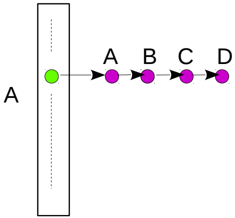
- The starting point (seed) is selected at random
- Scanning the neighborhood of visited point
- Routing towards the query point greedily

NN-Descent: a brief summary (3)

- The starting point (seed) is selected at random
- Scanning the neighborhood of visited point
- Routing towards the query point greedily
- What's more? We need a k-NN graph



(a) candidates



(b) 4-nn graph

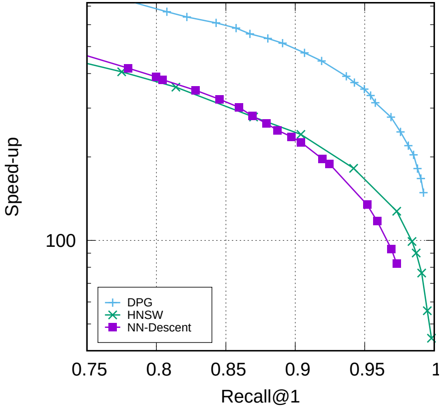
NN-Descent: how well it works? (1)

Table: Summary on Datasets used for Evaluation

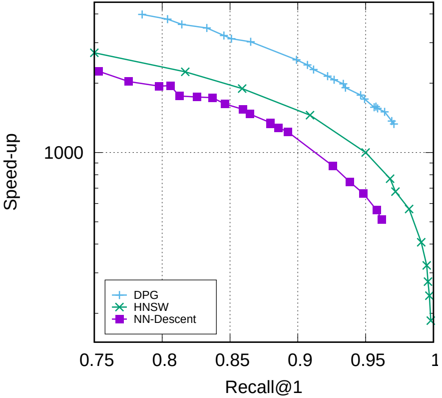
Name	n	d	# Qry	m(·)	Type
SIFT1M	1×10^6	128	1×10^4	l_2	SIFT
SIFT10M	1×10^7	128	1×10^4	l_2	SIFT
GIST1M	1×10^6	960	1×10^3	l_2	GIST
GloVe1M	1×10^6	100	1×10^3	*Cosine*	Text
NUSW	22,660	500	1×10^3	l_2	BoVW
NUSW	22,660	500	1×10^3	κ^2	BoVW
YFCC1M	1×10^6	128	1×10^4	l_2	Deep Feat.
Rand1M	1×10^6	100	1×10^3	l_2	synthetic

- They are all on million level

NN-Descent: how well it works? (2)



(a) SIFT1M



(b) SIFT10M

Figure: Performance of NN Descent Variants

NN-Descent: how well it works? (3)

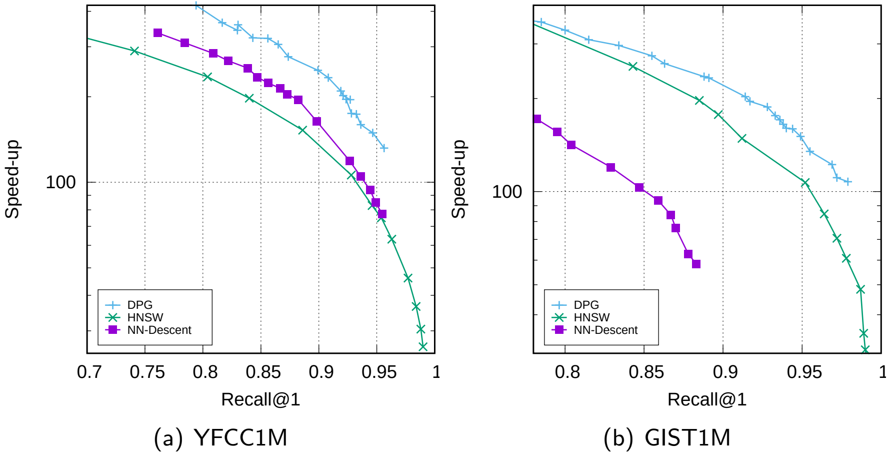
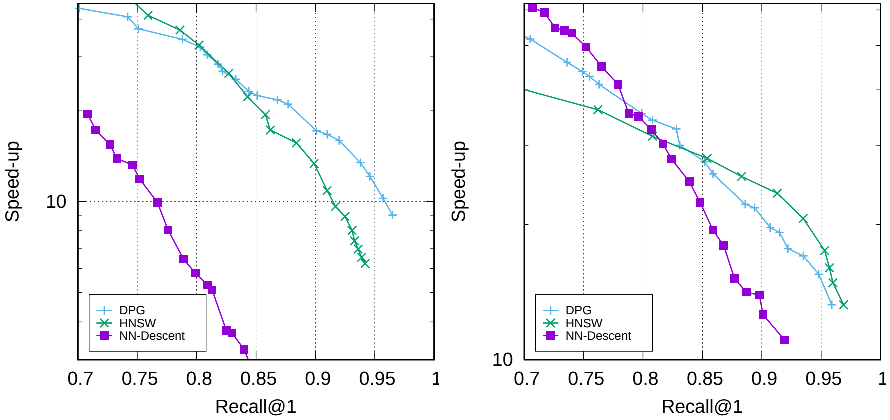


Figure: Performance of NN Descent Variants

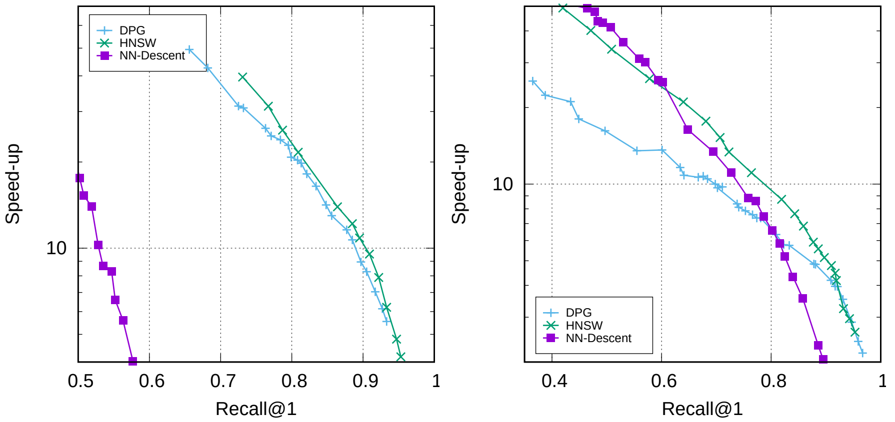
NN-Descent: how well it works? (4)



(a) NUSW-l_2 (b) NUSW-κ^2

Figure: Performance of NN Descent Variants

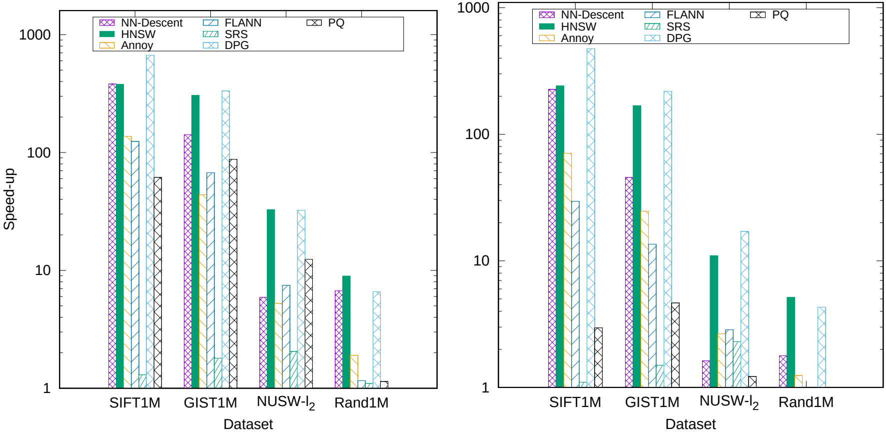
NN-Descent: how well it works? (5)



(a) GloVe1M (b) RAND1M

Figure: Performance of NN Descent Variants

NN-Descent: how well it works? (6)

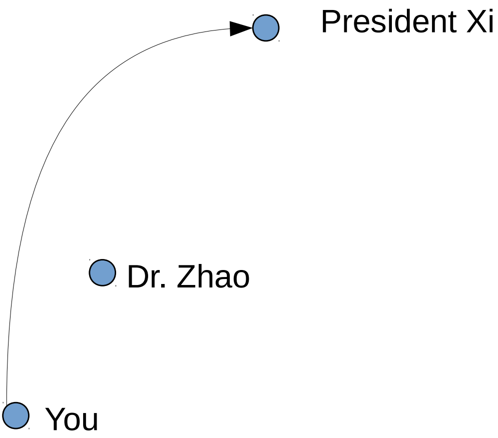


(a) SIFT1M (b) SIFT10M

Figure: Performance of NN Descent Variants

NN-Descent: why it works? (1)

- Think about the idea of "small world"



NN-Descent: why it works? (2)

- Think about the idea of “small world”

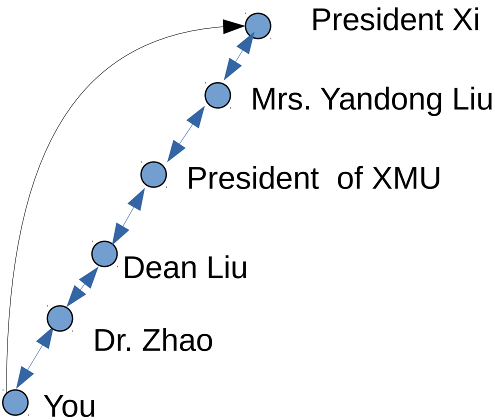
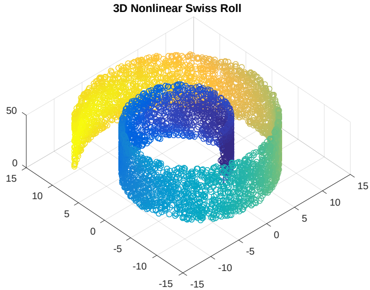


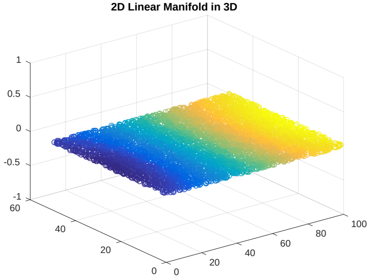
Figure: Illustration of smallworld.

NN-Descent: why it works? (3)

- Think about the idea of "small world"



(a) Swiss roll



(b) Swiss roll unfolded

Figure: The wide existence of subspace in realworld.

NN-Descent: why it works? (4)

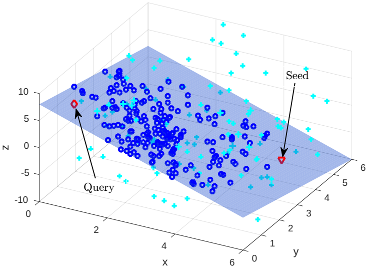


Figure: 2D sub-space embedded in 3D

- Climbing in the sub-space is much easier than exploring the whole

Intrinsic Dimension of Datasets Considered

Table: Dimension and Intrinsic Dimension of 8 Datasets

Name	n	d	Intrinsic Dim.	m($\cdot$)	Type
SIFT1M	1×10^6	128	18.7	l_2	SIFT
SIFT10M	1×10^7	128	18.7	l_2	SIFT
GIST1M	1×10^6	960	38.1	l_2	GIST
GloVe1M	1×10^6	100	39.5	*Cosine*	Text
NUSW	$22,660$	500	57.1	l_2	BoVW
NUSW	$22,660$	500	N.A.	κ^2	BoVW
YFCC1M	1×10^6	128	25.3	l_2	Deep Feat.
Rand1M	1×10^6	100	48.9	l_2	synthetic

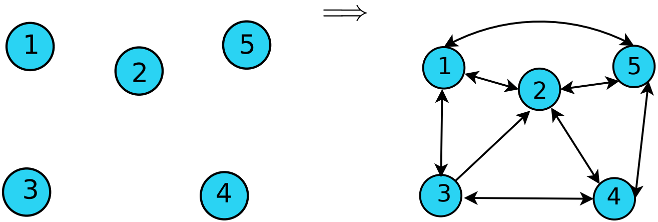
Outline

- NN-Descent Search is built upon a k-NN graph
- How to build the k-NN graph is a problem
- The k-NN graph should be in high quality
- The construction should be efficient
- The time complexity for exhaustive construction is $O(d \cdot n^2)$

NN-Descent for Graph Construction: the idea

- Based on the principle: neighbor's neighbor is likely the neighbor
- Samples in the neighhorhood are iteratively compared

NN-Descent: the procedure (1)



(a) Samples (b) k-NN graph

Figure: Approx. k-NN graph construction for a fixed dataset.

- Given a set of samples $x_{1...n} \in R^d$
- We want to build a k-NN graph based on metric $m(\cdot,\cdot)$
- The time complexity is $O(d \cdot n^2)$

NN-Descent: the procedure (2)

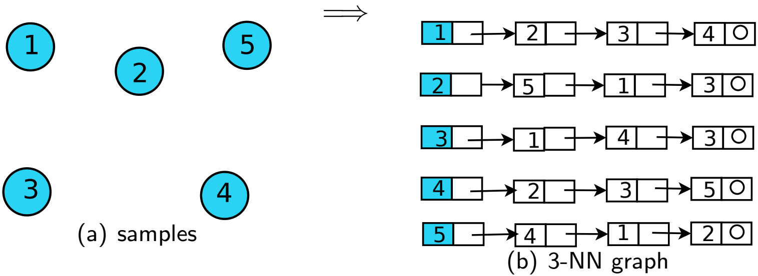


Figure: Step 1. Initialize a random 3-NN graph.

1. Initialize a 3-NN graph.

NN-Descent: the procedure (3)

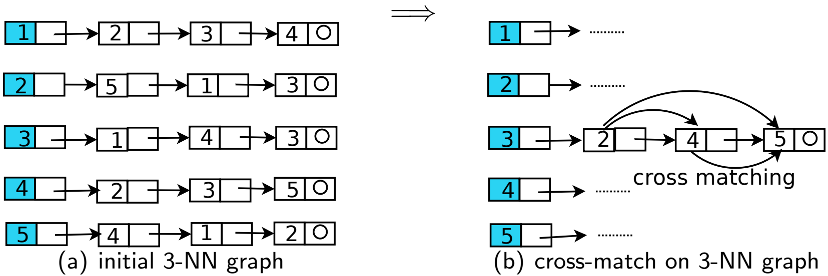
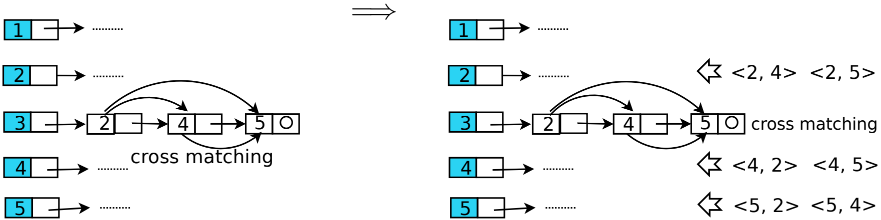


Figure: Step 2. cross-match on each 3-NN list.

2. Perform cross-matching on each 3-NN list.

NN-Descent: the procedure (4)



(a) cross-match

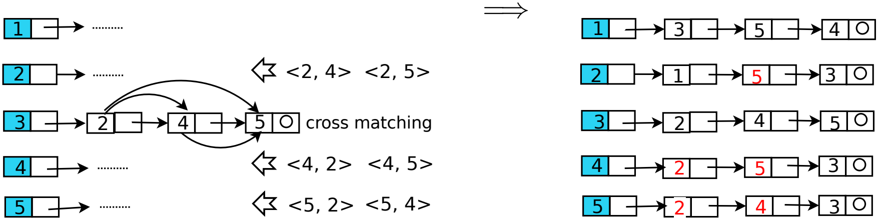
(b) cross-match on 3-NN graph

Figure: Step 2.1. join pairs into NN list.

2. Perform cross-matching on each 3-NN list.
    1. Join pairs into NN list.

Cross matching only happens between new and old samples, and within new samples

NN-Descent: the procedure (5)



(a) cross-match (b) Resulting 3-NN graph

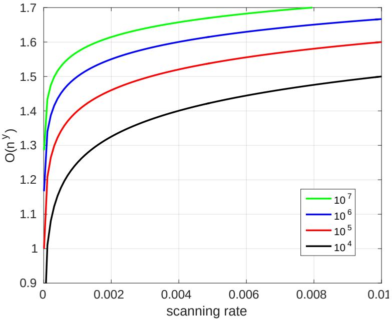
Figure: Step 2.1. join pairs into NN list.

2. Perform cross-matching on each 3-NN list.
   1. Join pairs into NN list.
3. Repeat above steps until it converges

NN-Descent: comments (1)

Table: Scanning rate

$n = 10^5, d =$	l_1	l_2
2	0.0040	0.0034
5	0.0057	0.0047
10	0.0088	0.0075
20	0.0213	0.0209
50	0.1037	0.1014
100	0.1390	0.1370



1. NN-Descent is a batchful version hill-climbing
2. It is generic and efficient particularly on low dimensional data
3. It suffers from "curse of dimensionality" as well

A Summary

<table>
<tr><td>Low & Dense
KD-tree</td><td>Low & Sparse
KD-tree, Inverted file</td></tr>
<tr><td>High & Low ID & Dense PQ, NN-Desc.</td><td rowspan="2">High & Sparse
Inverted file</td></tr>
<tr><td>High & High ID & Dense PQ, NN-Desc.</td></tr>
</table>

Available toolkits in the web

- ANN: approximate nearest neighbor search library
  - Based on KD-Tree
  - By Arya et. al from University of Maryland
- E2LSH: http://www.mit.edu/~andoni/LSH/
  - Based on LSH
  - By Alexandr Andoni and Piotr Indyk from MIT
- FLANN: http://www.cs.ubc.ca/research/flann/
  - Based on hierarchical k-means
  - By Marius Muja and David G. Lowe from University of British Columnbia
- KGraph: https://github.com/erikbern/ann-benchmarks
  - Based on NN Descent Alglrithm
  - By Wei Dong from Princeton University

References

1. R-Trees: A Dynamic Index Structure for Spatial Searching, A. Guttman, SIGMOD'84
2. Product Quantization for Nearest Neighbor Search, H. Jegou, M. Douze and C. Schmid, TPAMI'12
3. Similarity Search in High Dimensions via Hashing, A. Gionis, P. Indyk, R. Motwani, VLDB'99
4. Scalable Nearest Neighbor Algorithms for High Dimensional Data, M. Muja and D. G. Lowe, TPAMI'14
    - URL: http://www.cs.ubc.ca/research/flann/
    - Comments: built by hierarchical k-means
5. Efficient k-Nearest Neighbor Graph Construction for Generic Similarity Measures, W. Dong, et. al, WWW'11
6. Fast k-Nearest Neighbor Graph Construction: a generic online approach, W.-L. Zhao, https://arxiv.org/abs/1804.03032

Q & A

Thanks for your attention!