

Approximate nearest neighbor search: binary codes and vector quantization

Yannis Avrithis

University of Athens

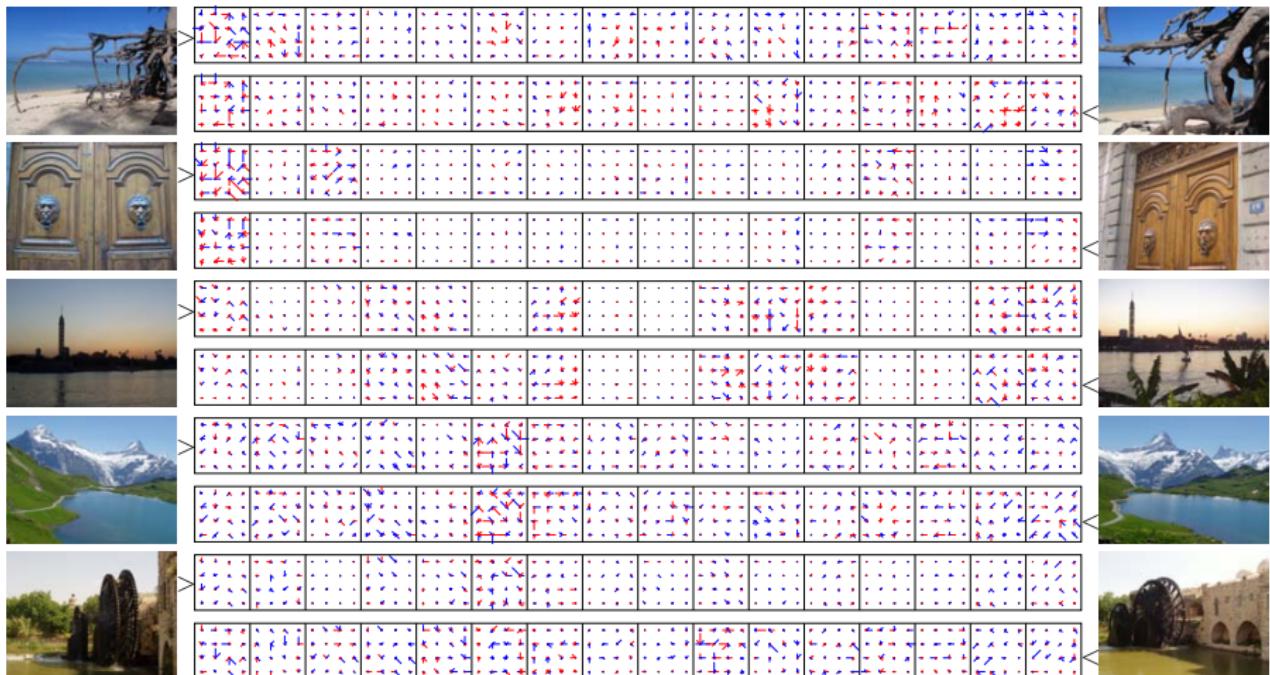
Athens, March 2015

Problem

- Given query point q , find its nearest neighbor with respect to Euclidean distance within data set \mathcal{X} in a d -dimensional space
- Focus on large scale: encode (compress) vectors, speed up distance computations
- Fit underlying distribution with little space & time overhead

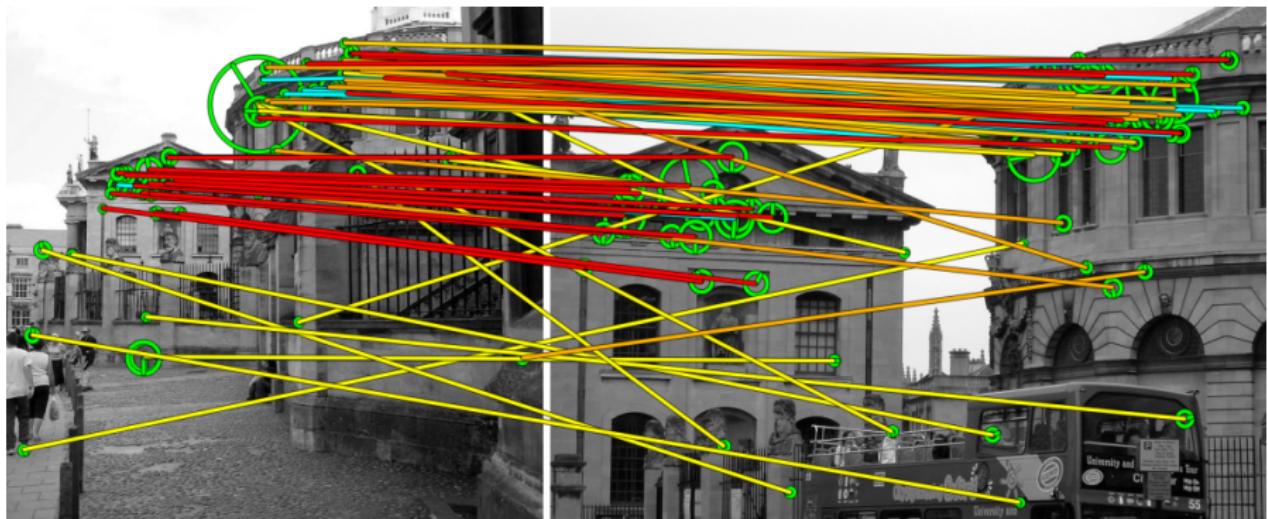
Applications in vision

Retrieval (image as point) [Jégou et al. '10][Perronnin et al. '10]



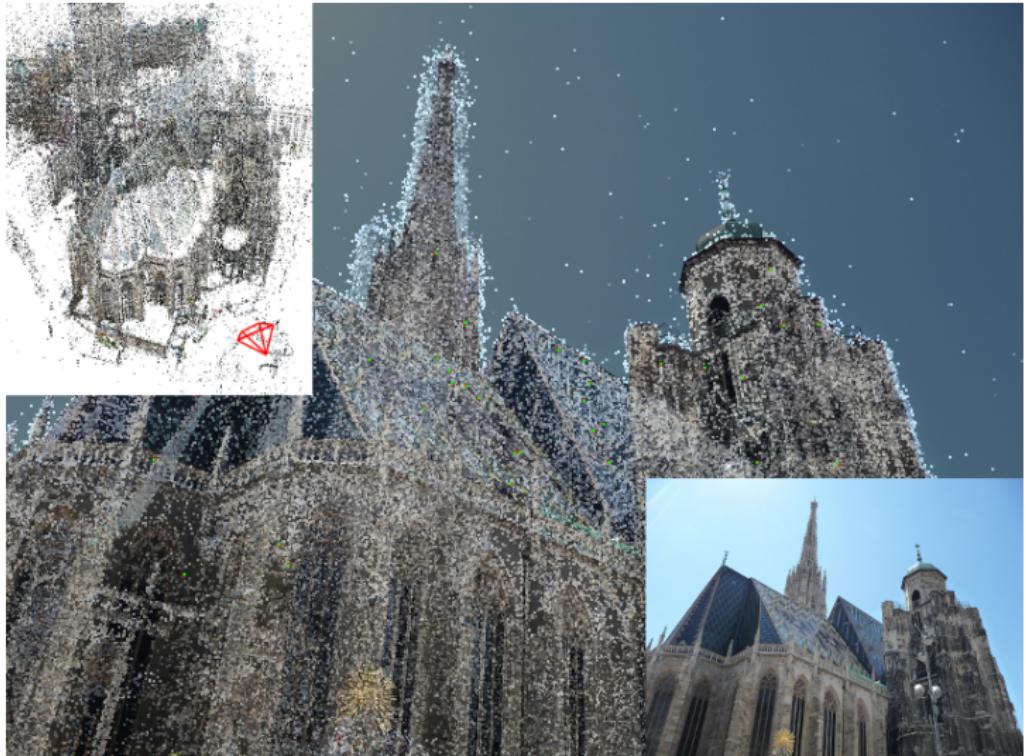
Applications in vision

Retrieval (patch as point) [Tolias et al. '13][Qin et al. '13]



Applications in vision

Localization, pose estimation [Sattler et al. '12][Li et al. '12]



Applications in vision

Classification [Boiman et al. '08][McCann & Lowe '12]

*query
image
 Q*



$$KL(p_Q \mid p_C) = 8.35$$



$$KL(p_Q \mid p_1) = 17.54$$



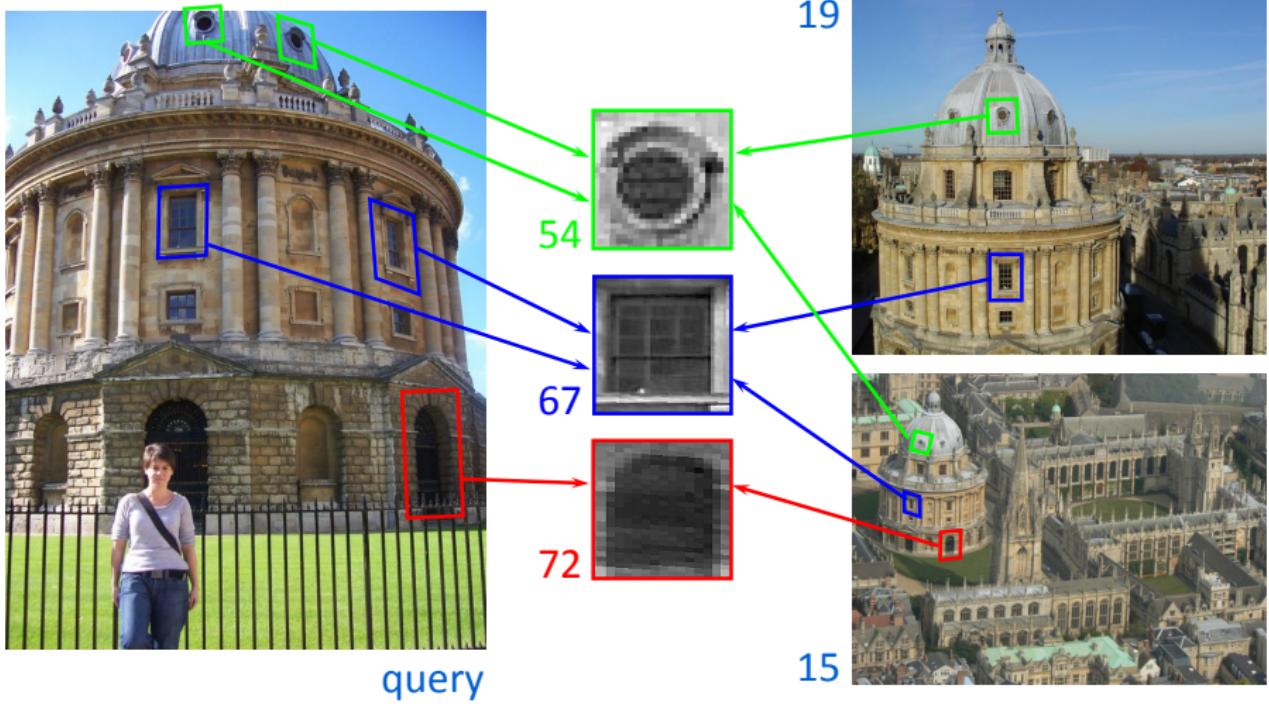
$$KL(p_Q \mid p_2) = 18.20$$



$$KL(p_Q \mid p_3) = 14.56$$

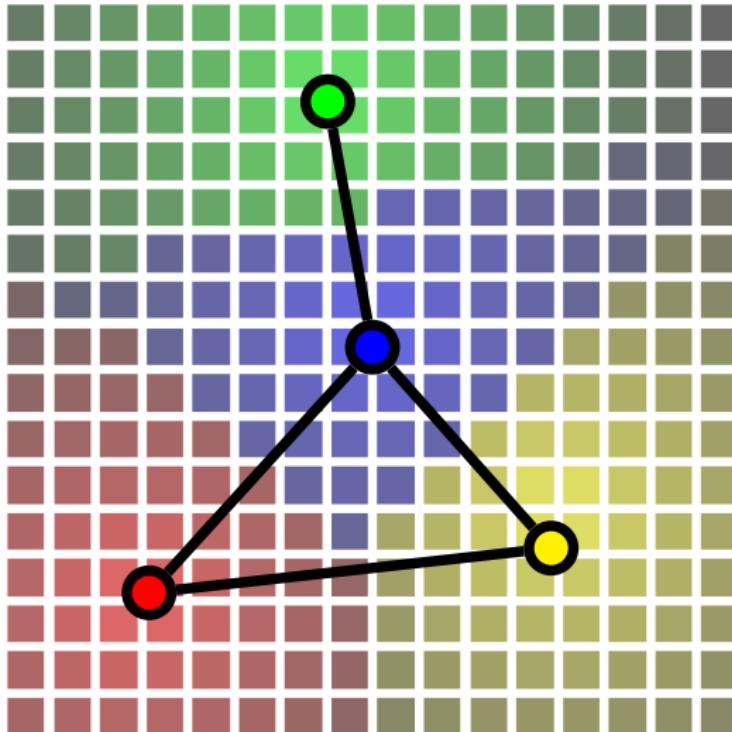
Applications in vision

Quantization [Sivic et al. '03][Philbin et al. '07]



Applications in vision

Clustering [Philbin et al. '07][Avrithis '13]



Overview (1)

Tree-based search

- k -d trees [Bentley '75]
- randomized k -d trees [Silpa-Anan & Hartley '02]
- hierarchical k -means tree [Fukunaga & Narendra '75]
- FLANN [Muja & Lowe '09]

Binary codes

- locality sensitive hashing [Charikar '02]
- spectral hashing [Weiss *et al.* '08]
- iterative quantization [Gong and Lazebnik '11]

Overview (1)

Tree-based search

- k -d trees [Bentley '75]
- randomized k -d trees [Silpa-Anan & Hartley '02]
- hierarchical k -means tree [Fukunaga & Narendra '75]
- FLANN [Muja & Lowe '09]

Binary codes

- locality sensitive hashing [Charikar '02]
- spectral hashing [Weiss *et al.* '08]
- iterative quantization [Gong and Lazebnik '11]

Overview (2)

Quantization

- vector quantization (VQ)
- product quantization (PQ) [Jégou *et al.* '11]
- optimized product quantization (OPQ) [Ge *et al.* '13]
Cartesian k -means [Norouzi & Fleet '13]
- locally optimized product quantization (LOPQ) [Kalantidis and Avrithis '14]

Non-exhaustive search

- non-exhaustive PQ [Jégou *et al.* '11]
- inverted multi-index [Babenko & Lempitsky '12]
- multi-LOPQ [Kalantidis and Avrithis '14]

Overview (2)

Quantization

- vector quantization (VQ)
- product quantization (PQ) [Jégou *et al.* '11]
- optimized product quantization (OPQ) [Ge *et al.* '13]
Cartesian k -means [Norouzi & Fleet '13]
- locally optimized product quantization (LOPQ) [Kalantidis and Avrithis '14]

Non-exhaustive search

- non-exhaustive PQ [Jégou *et al.* '11]
- inverted multi-index [Babenko & Lempitsky '12]
- multi-LOPQ [Kalantidis and Avrithis '14]

Overview (3)

Clustering

- hierarchical k -means [Nister & Stewenius '06]
- approximate k -means [Philbin *et al.* '07]
- approximate Gaussian mixtures [Kalantidis & Avrithis '12]
- dimensionality-recursive vector quantization [Avrithis '13]
- ranked retrieval [Broder *et al.* '14]

I. Tree-based search

k-d tree

[Bentley '75]

Construction

- choose the dimension of greatest variance
- split at medoid to make tree balanced
- recurse until both sides of splitting plane are empty

Search (exact)

- at each node, choose child according to splitting dimension and value
- starting at root, descend recursively
- backtrack

k-d tree

[Bentley '75]

Construction

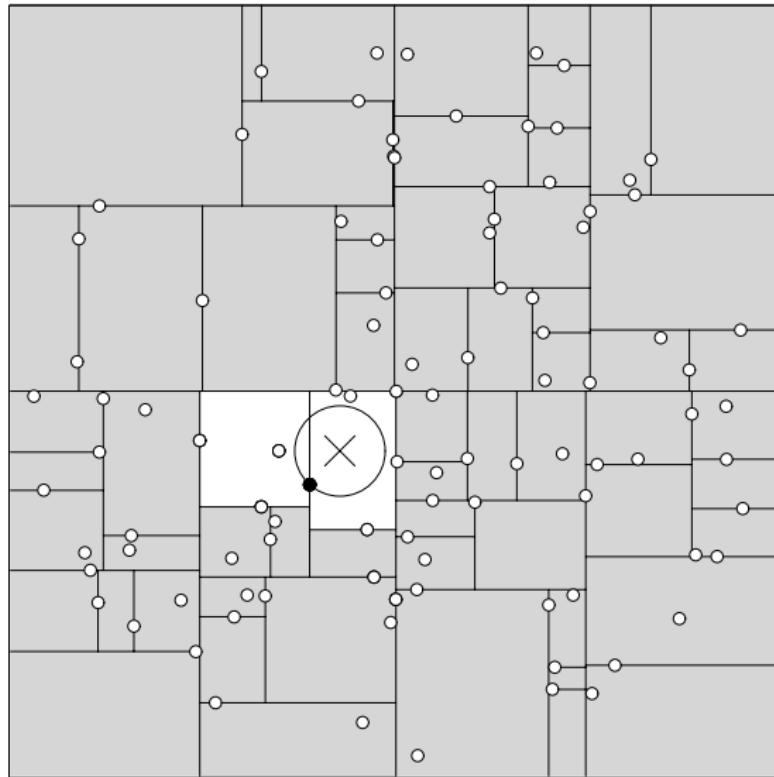
- choose the dimension of greatest variance
- split at medoid to make tree balanced
- recurse until both sides of splitting plane are empty

Search (exact)

- at each node, choose child according to splitting dimension and value
- starting at root, descend recursively
- backtrack

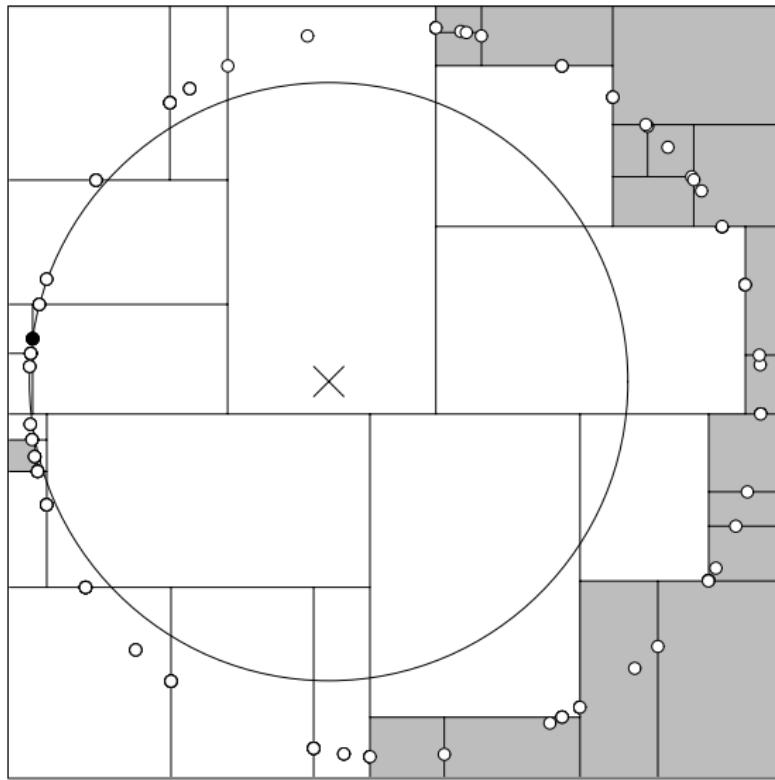
k-d tree

[Bentley '75]



k-d tree

[Bentley '75]



Randomized k -d trees

[Silpa-Anan & Hartley '75]

Construction

- construct m different trees
- randomly rotate data points
- choose randomly among the dimensions of greatest variance
- split at a random point near the medoid

Search (approximate)

- descend each tree once independently
- insert nodes in a shared priority queue
- keep descending until l leaves are visited

Randomized k -d trees

[Silpa-Anan & Hartley '75]

Construction

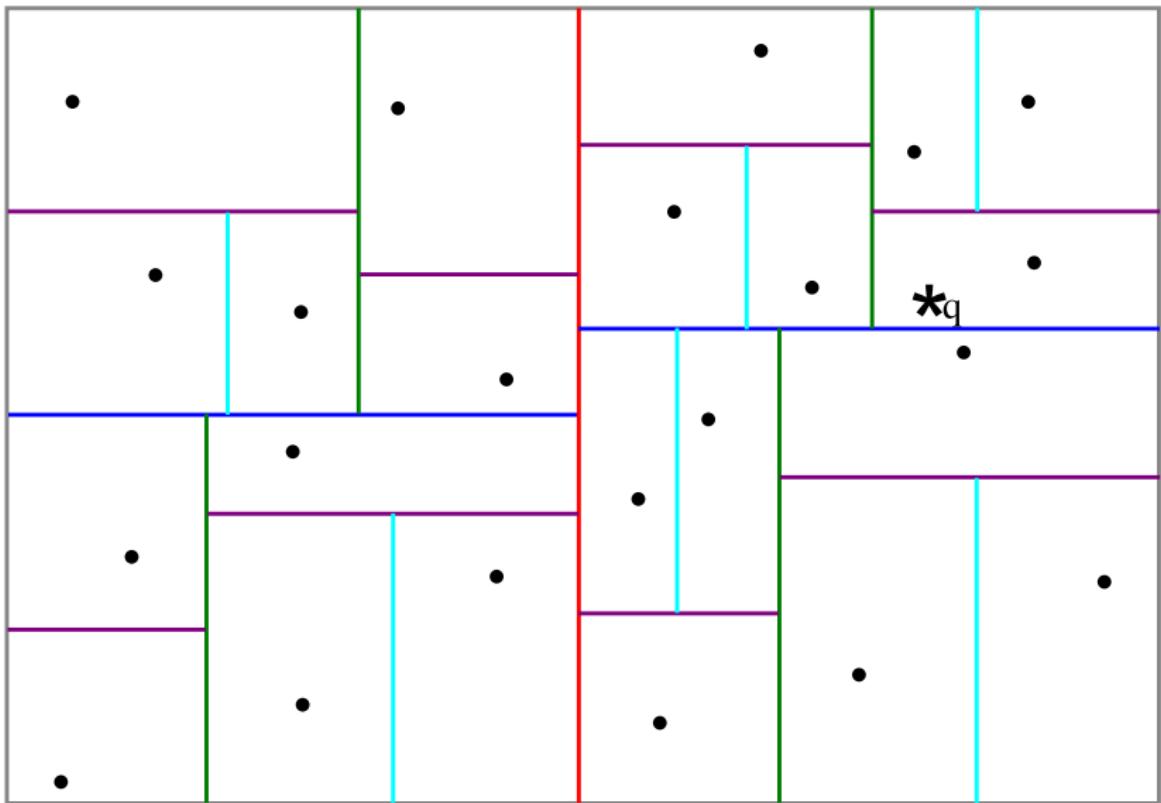
- construct m different trees
- randomly rotate data points
- choose randomly among the dimensions of greatest variance
- split at a random point near the medoid

Search (approximate)

- descend each tree once independently
- insert nodes in a shared priority queue
- keep descending until l leaves are visited

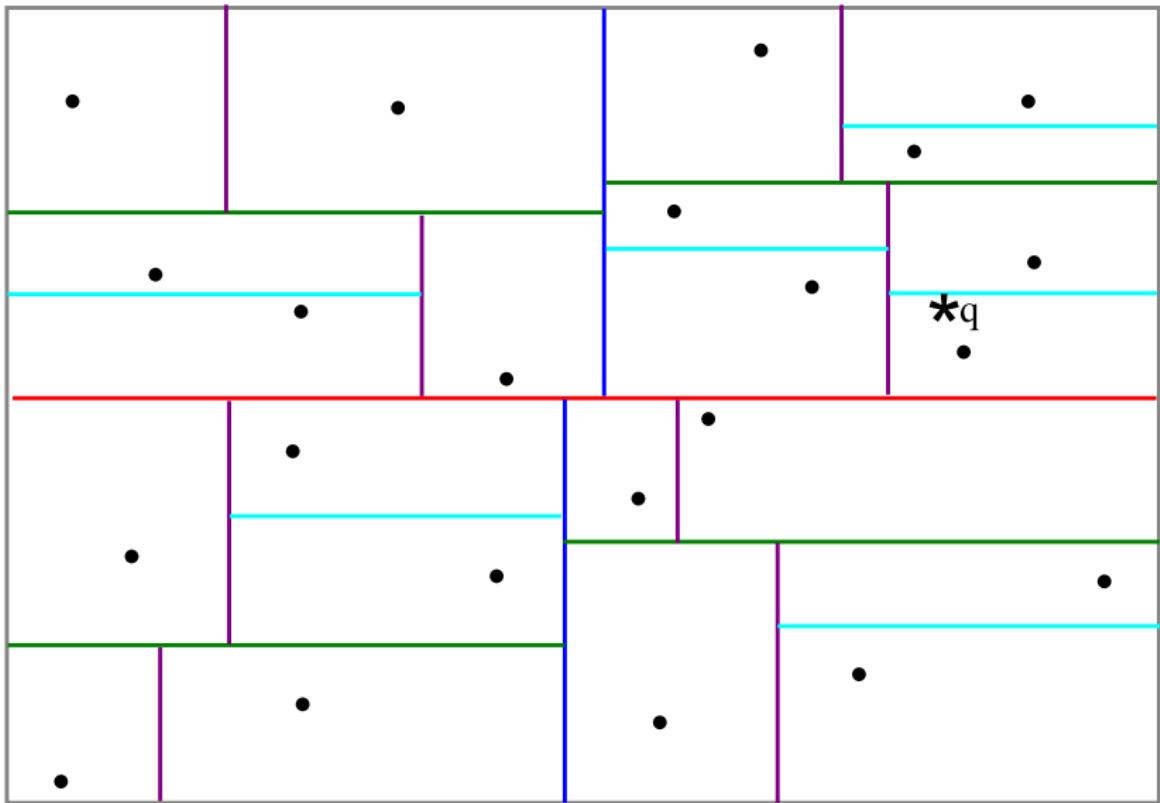
Randomized k -d trees

[Silpa-Anan & Hartley '75]



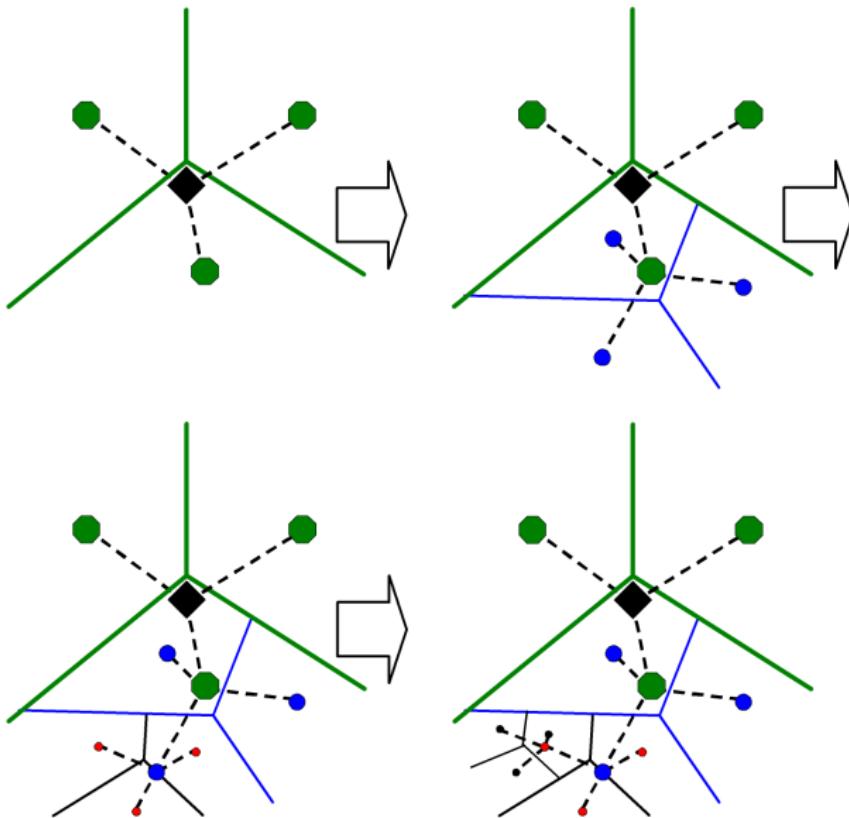
Randomized k -d trees

[Silpa-Anan & Hartley '75]



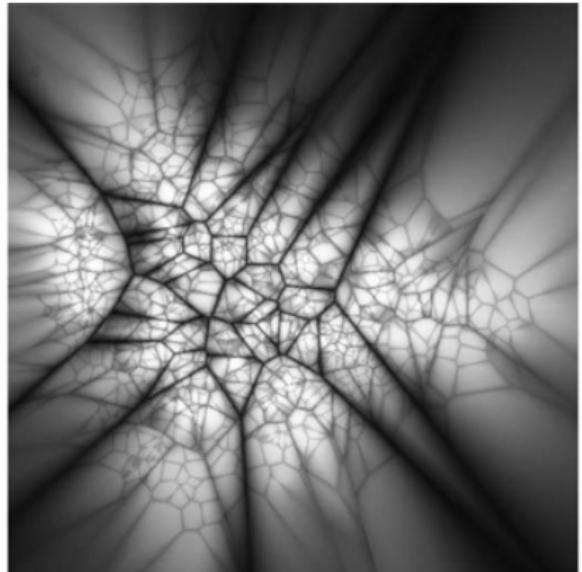
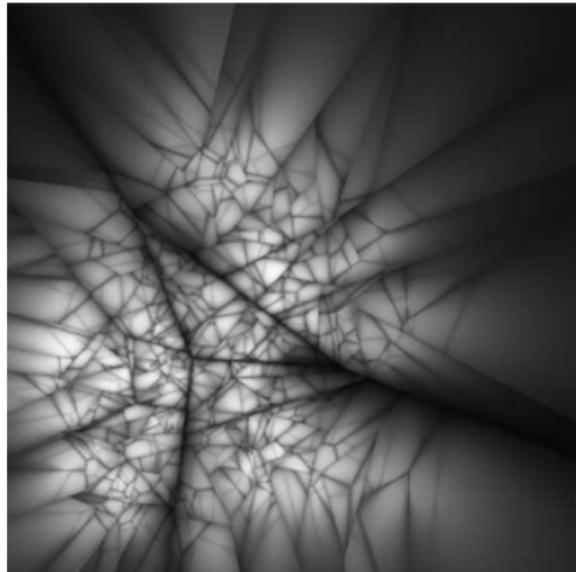
Hierarchical k -means tree

[Fukunaga & Narendra '75][Nister & Stewenius '06]



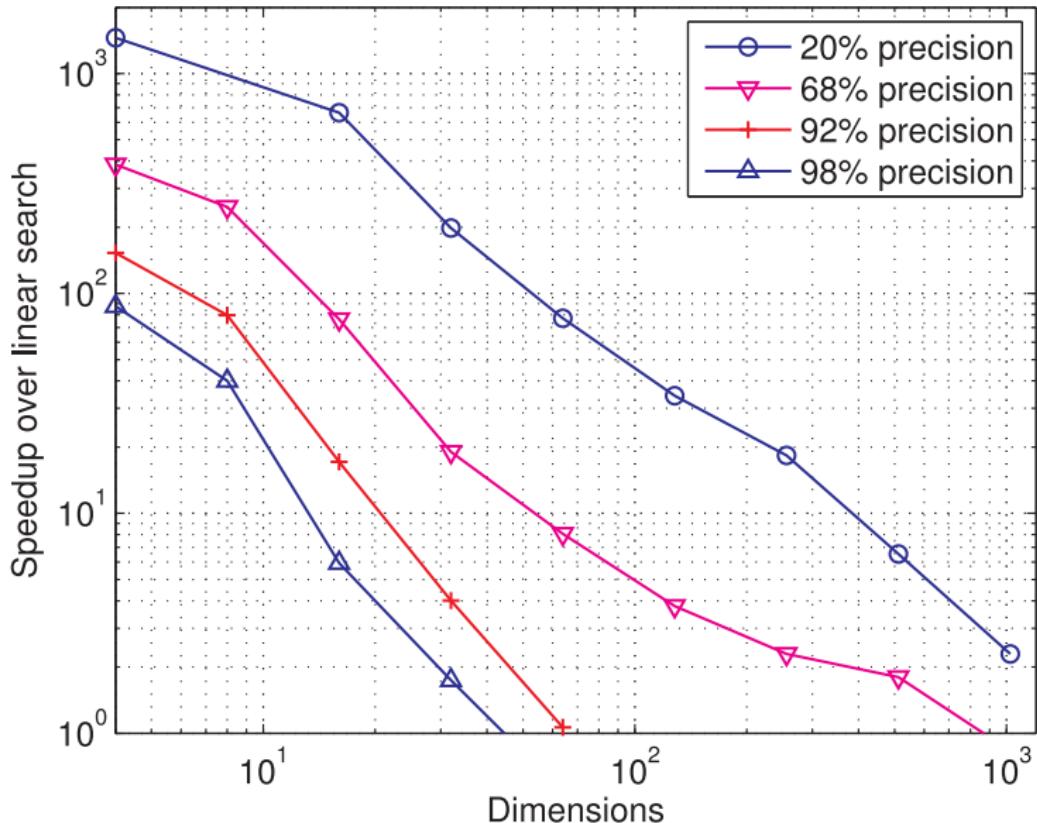
Hierarchical k -means tree

[Fukunaga & Narendra '75][Nister & Stewenius '06]



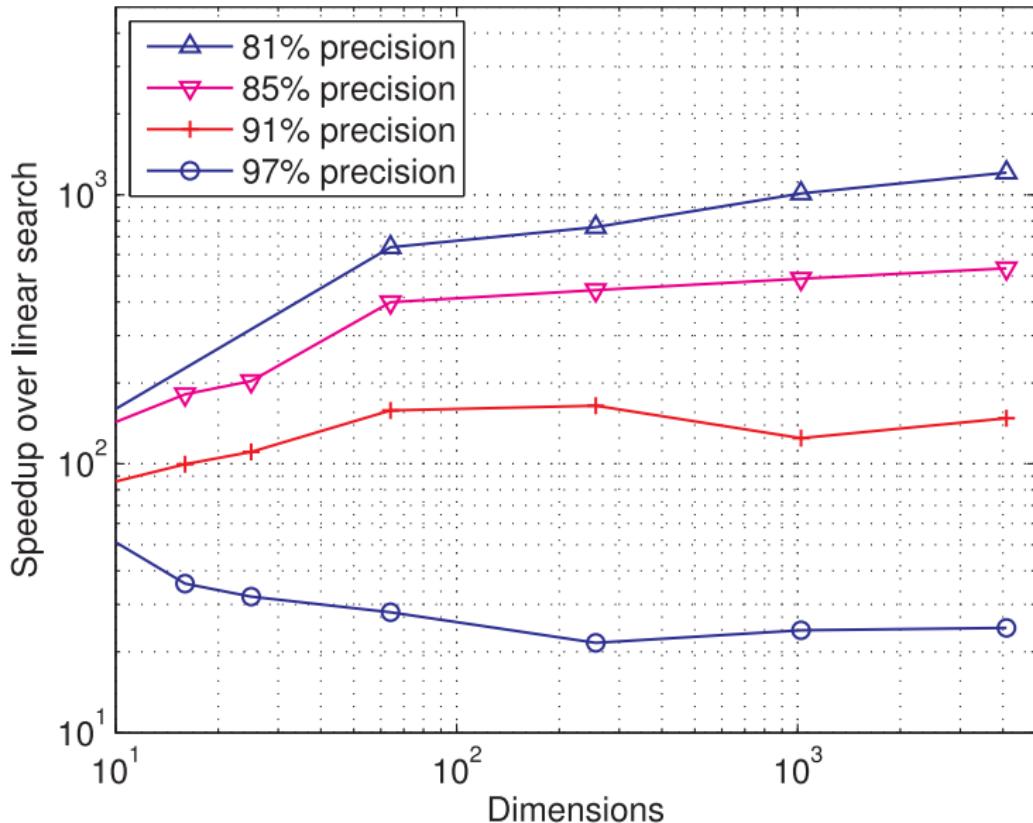
FLANN (uniform data)

[Muja & Lowe '09]



FLANN (real data)

[Muja & Lowe '09]



Tree-based methods

All methods so far

- assume all data points are represented **exactly** in memory
- compute **exact distances** to a subset of data points
- design a data structure for efficient search

We rather focus on methods that

- only **approximate** data points
- use space partition not only to limit search but to **approximate distances**
- design an efficient encoding

Tree-based methods

All methods so far

- assume all data points are represented **exactly** in memory
- compute **exact distances** to a subset of data points
- design a data structure for efficient search

We rather focus on methods that

- only **approximate** data points
- use space partition not only to limit search but to **approximate distances**
- design an efficient encoding

II. Binary codes

Locality sensitive hashing

random projections [Charikar '02]

- Choose a random vector \mathbf{a} from the d -dimensional Gaussian distribution $\mathcal{N}(0, 1)$.
- Define *hash function* $h_{\mathbf{a}} : \mathbb{R}^d \rightarrow \{-1, 1\}$ with

$$h_{\mathbf{a}}(\mathbf{x}) = \operatorname{sgn}(\mathbf{a} \cdot \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{a} \cdot \mathbf{x} \geq 0 \\ -1, & \text{if } \mathbf{a} \cdot \mathbf{x} < 0. \end{cases}$$

- Then, given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbb{P}[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi}$$

where $\theta(\mathbf{x}, \mathbf{y})$ is the angle between \mathbf{x}, \mathbf{y} .

Locality sensitive hashing

random projections [Charikar '02]

- Choose a random vector \mathbf{a} from the d -dimensional Gaussian distribution $\mathcal{N}(0, 1)$.
- Define *hash function* $h_{\mathbf{a}} : \mathbb{R}^d \rightarrow \{-1, 1\}$ with

$$h_{\mathbf{a}}(\mathbf{x}) = \operatorname{sgn}(\mathbf{a} \cdot \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{a} \cdot \mathbf{x} \geq 0 \\ -1, & \text{if } \mathbf{a} \cdot \mathbf{x} < 0. \end{cases}$$

- Then, given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbb{P}[h_{\mathbf{a}}(\mathbf{x}) = h_{\mathbf{a}}(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi}$$

where $\theta(\mathbf{x}, \mathbf{y})$ is the angle between \mathbf{x}, \mathbf{y} .

Binary codes and Hamming distance

- Given a set of n data points $\mathbf{x}_i \in \mathbb{R}^d$, represented by matrix $X \in \mathbb{R}^{d \times n}$.
- Define k hash functions $h_j : \mathbb{R}^d \rightarrow \{-1, 1\}$, and let $h(\mathbf{x}) = (h_1(\mathbf{x}), \dots, h_k(\mathbf{x}))$.
- Encode each data point \mathbf{x} by **binary code** $\mathbf{y} = h(\mathbf{x})$, and represent all encoded points by matrix $Y \in \{-1, 1\}^{k \times n}$.
 - For instance, $Y = \text{sgn}(A^\top X)$ for random projections, where $A \in \mathbb{R}^{d \times k}$ represents the k random vectors.
- Now, given a query \mathbf{q} , encode it as $h(\mathbf{q})$ and search in Y by **Hamming distance**.

Binary codes and Hamming distance

- Given a set of n data points $\mathbf{x}_i \in \mathbb{R}^d$, represented by matrix $X \in \mathbb{R}^{d \times n}$.
- Define k hash functions $h_j : \mathbb{R}^d \rightarrow \{-1, 1\}$, and let $h(\mathbf{x}) = (h_1(\mathbf{x}), \dots, h_k(\mathbf{x}))$.
- Encode each data point \mathbf{x} by **binary code** $\mathbf{y} = h(\mathbf{x})$, and represent all encoded points by matrix $Y \in \{-1, 1\}^{k \times n}$.
 - For instance, $Y = \text{sgn}(A^\top X)$ for random projections, where $A \in \mathbb{R}^{d \times k}$ represents the k random vectors.
- Now, given a query \mathbf{q} , encode it as $h(\mathbf{q})$ and search in Y by **Hamming distance**.

Spectral hashing

[Weiss et al. '08]

- Define **similarity matrix** S with $S_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/t^2)$.
- Require binary codes to be **similarity preserving**, **balanced**, and **uncorrelated**:

$$\begin{aligned} & \text{minimize} && \sum_{ij} S_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ & \text{subject to} && \mathbf{y}_i \in \{-1, 1\}^k \\ & && \sum_i \mathbf{y}_i = 0 \\ & && \frac{1}{n} \sum_i \mathbf{y}_i \mathbf{y}_i^\top = I. \end{aligned}$$

Spectral hashing

[Weiss et al. '08]

- Define **similarity matrix** S with $S_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/t^2)$.
- Require binary codes to be **similarity preserving, balanced, and uncorrelated**:

$$\begin{aligned} & \text{minimize} && \sum_{ij} S_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \\ & \text{subject to} && \mathbf{y}_i \in \{-1, 1\}^k \\ & && \sum_i \mathbf{y}_i = 0 \\ & && \frac{1}{n} \sum_i \mathbf{y}_i \mathbf{y}_i^\top = I. \end{aligned}$$

Spectral hashing

Relaxation

- Define Laplacian matrix $L = D - S$ with $D = \text{diag}(S\mathbf{1})$.
- Problem is relaxed as

$$\begin{aligned} & \text{minimize} && \text{tr}(YLY^\top) \\ & \text{subject to} && Y\mathbf{1} = 0 \\ & && YY^\top = I, \end{aligned}$$

and solutions are the k eigenvectors of L with minimal eigenvalue, excluding eigenvector $\mathbf{1}$ with eigenvalue 0.

- See also Laplacian eigenmaps [Belkin & Niyogi '01].

Spectral hashing

Relaxation

- Define Laplacian matrix $L = D - S$ with $D = \text{diag}(S\mathbf{1})$.
- Problem is relaxed as

$$\begin{aligned} & \text{minimize} && \text{tr}(YLY^\top) \\ & \text{subject to} && Y\mathbf{1} = 0 \\ & && YY^\top = I, \end{aligned}$$

and solutions are the k eigenvectors of L with minimal eigenvalue, excluding eigenvector $\mathbf{1}$ with eigenvalue 0.

- See also Laplacian eigenmaps [Belkin & Niyogi '01].

Spectral hashing

Out of sample extension

- Replace data points by probability distribution p ; and Laplacian matrix by Laplacian operator L_p acting on functions.
- Then, solutions are the k eigenfunctions f of L_p (such that $L_p f = \lambda f$) with minimal eigenvalue, excluding eigenfunction $f(\mathbf{x}) = 1$ with eigenvalue 0.
- If p is uniform, then eigenfunctions have outer product form, and for 1-dimensional distribution on $[a, b]$,

$$\begin{aligned}\phi_j(x) &= \sin\left(\frac{\pi}{2} + \frac{j\pi}{b-a}x\right) \\ \lambda_j &= 1 - e^{-\frac{t^2}{2}\left(\frac{j\pi}{b-a}\right)^2}\end{aligned}$$

Spectral hashing

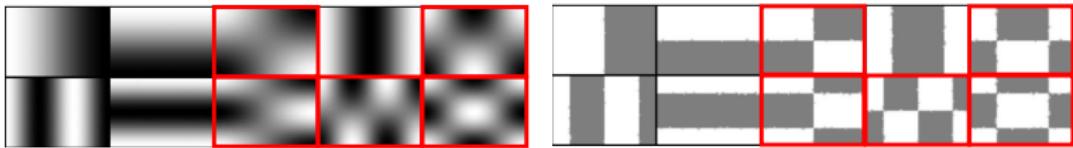
Out of sample extension

- Replace data points by probability distribution p ; and Laplacian matrix by Laplacian operator L_p acting on functions.
- Then, solutions are the k eigenfunctions f of L_p (such that $L_p f = \lambda f$) with minimal eigenvalue, excluding eigenfunction $f(\mathbf{x}) = 1$ with eigenvalue 0.
- If p is uniform, then eigenfunctions have outer product form, and for 1-dimensional distribution on $[a, b]$,

$$\begin{aligned}\phi_j(x) &= \sin\left(\frac{\pi}{2} + \frac{j\pi}{b-a}x\right) \\ \lambda_j &= 1 - e^{-\frac{t^2}{2}\left(\frac{j\pi}{b-a}\right)^2}\end{aligned}$$

Spectral hashing

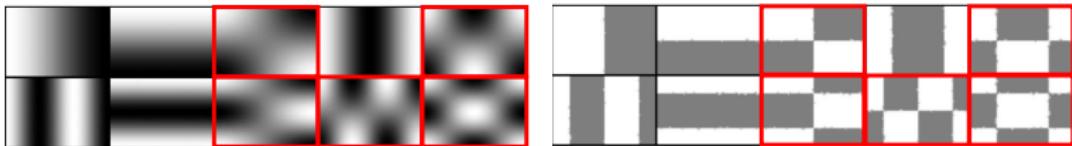
Example



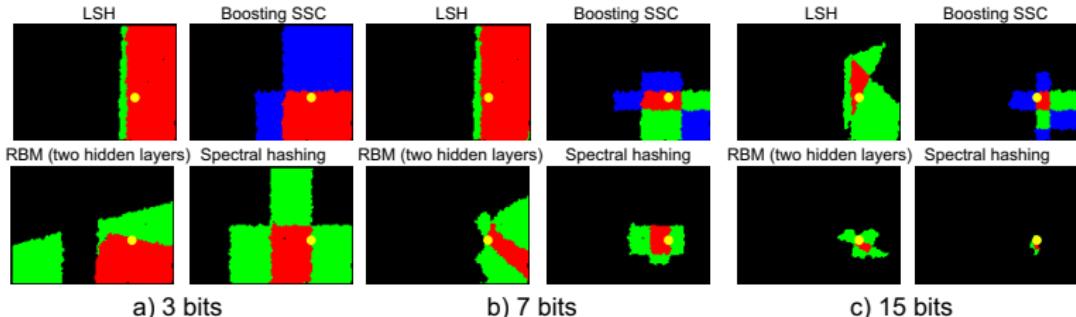
- Red: outer-product eigenfunctions: excluded
- Better to cut long dimension first
- Lower spatial frequencies are better than higher ones

Spectral hashing

Example



- Red: outer-product eigenfunctions: excluded
- Better to cut long dimension first
- Lower spatial frequencies are better than higher ones



- Red: radius = 0; green: radius = 1; blue: radius = 2

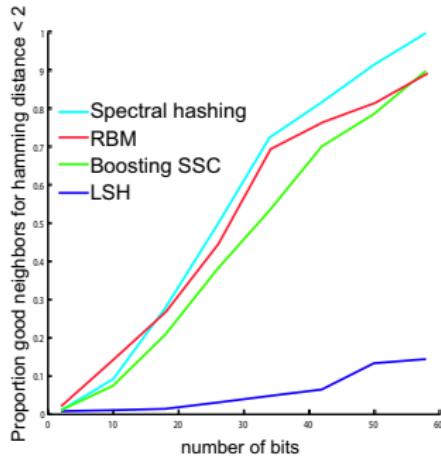
Spectral hashing

Algorithm

1. Center and rotate data points by PCA.
2. Evaluate k smallest eigenvalues for each PCA direction.
3. Sort the kd eigenvalues, exclude outer-product ones, and select the k smallest.
4. Set hash function $h_j(\mathbf{x}) = \text{sgn}(\phi_j(\mathbf{x}))$ for each of the corresponding k eigenfunctions ϕ_j .

Spectral hashing

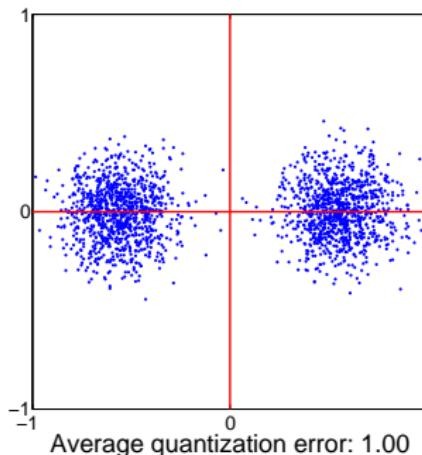
Result on LabelMe



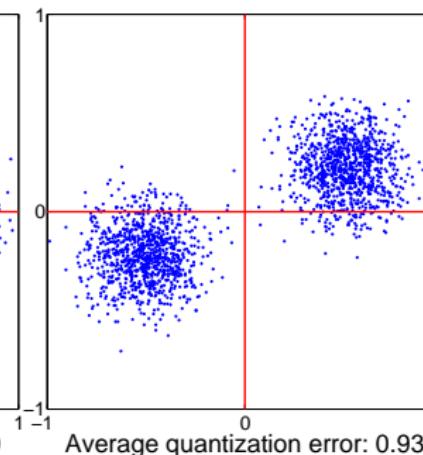
Iterative quantization

[Gong and Lazebnik '11]

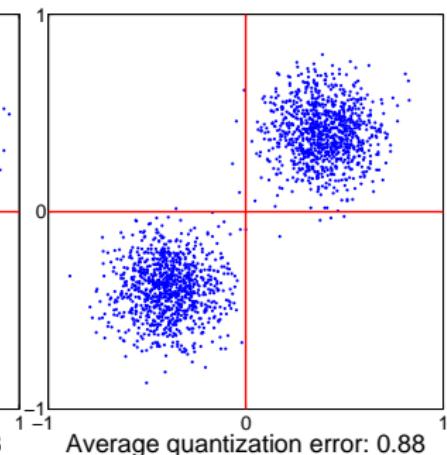
Quantize each data point to the closest vertex of the binary cube,
 $(\pm 1, \pm 1)$.



(a) PCA aligned.



(b) Random Rotation.



(c) Optimized Rotation.

Iterative quantization

Formulation

- Assume data points to be zero centered, $X\mathbf{1} = 0$.
- Assume hash functions $y^j = h_j(\mathbf{x}) = \text{sgn}(\mathbf{a}_j \cdot \mathbf{x})$, or $Y = \text{sgn}(A^\top X)$.
- Drop similarity preservation
- Balance $h_j(\mathbf{x}) \cdot \mathbf{1} = 0$ is equivalent to variance of $h_j(\mathbf{x})$ being maximized:

$$\begin{aligned} & \text{maximize} && \sum_j \text{var}(\text{sgn}(\mathbf{a}_j^\top X)) \\ & \text{subject to} && \frac{1}{n}YY^\top = I. \end{aligned}$$

Iterative quantization

Relaxation

- Drop sgn.
- Relax correlation constraint by just requiring hyperplanes to be orthogonal:

$$\begin{aligned} & \text{maximize} && \text{tr}(A^\top X X^\top A) \\ & \text{subject to} && A^\top A = I, \end{aligned}$$

and a solution consists of the k eigenvectors of data covariance matrix XX^\top with maximal eigenvalue.

- See also semi-supervised hashing [Wang et al. '10].

Iterative quantization

Relaxation

- Drop sgn.
- Relax correlation constraint by just requiring hyperplanes to be orthogonal:

$$\begin{aligned} & \text{maximize} && \text{tr}(A^\top X X^\top A) \\ & \text{subject to} && A^\top A = I, \end{aligned}$$

and a solution consists of the k eigenvectors of data covariance matrix XX^\top with maximal eigenvalue.

- See also semi-supervised hashing [Wang et al. '10].

Iterative quantization

Refinement

- But, if A is an optimal solution, then so is AR^\top for orthogonal $R \in \mathbb{R}^{k \times k}$.
- So, if $Z = A^\top X$ is the projected data, define loss

$$E(Y, R) = \|Y - RZ\|_F^2$$

and repeat

- Fix R , update $Y \leftarrow \text{sgn}(RZ)$
- Fix Y , update $R \leftarrow UV^\top$ where $YZ^\top = USV^\top$ (align by SVD)
- See also **multiclass spectral clustering** [Yu & Shi '03].

Iterative quantization

Refinement

- But, if A is an optimal solution, then so is AR^\top for orthogonal $R \in \mathbb{R}^{k \times k}$.
- So, if $Z = A^\top X$ is the projected data, define loss

$$E(Y, R) = \|Y - RZ\|_F^2$$

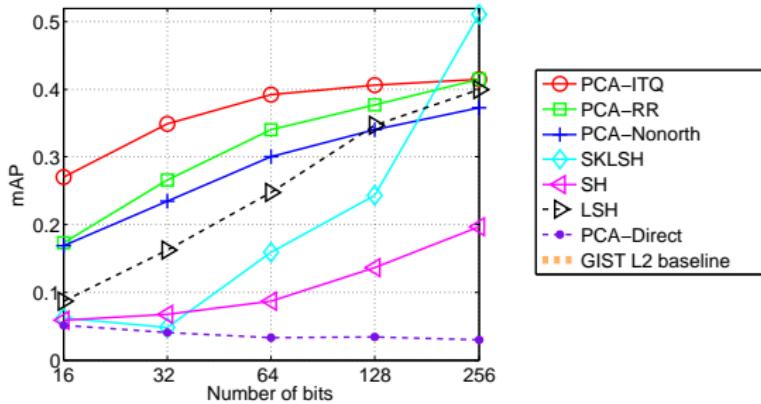
and repeat

- Fix R , update $Y \leftarrow \text{sgn}(RZ)$
- Fix Y , update $R \leftarrow UV^\top$ where $YZ^\top = USV^\top$ (align by SVD)
- See also **multiclass spectral clustering** [Yu & Shi '03].

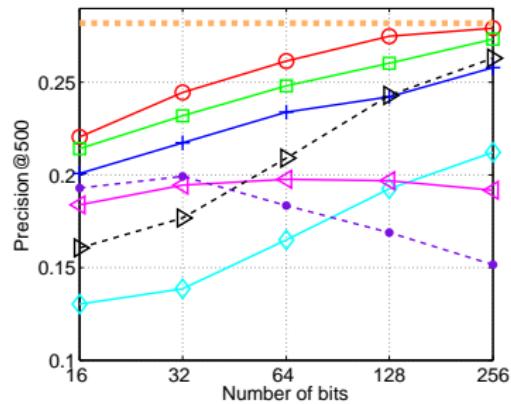
Iterative quantization

Result on CIFAR

(a) Euclidean ground truth



(b) Class label ground truth



III. Quantization

Locality sensitive hashing

scalar quantization [Datar et al. '04]

- Choose a random vector \mathbf{a} from the d -dimensional Gaussian distribution $f = \mathcal{N}(0, 1)$ and a real b uniformly in $[0, r]$.
- Define *hash function* $h_{\mathbf{a}, b} : \mathbb{R}^d \rightarrow \mathbb{Z}$ with

$$h_{\mathbf{a}, b}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{x} + b}{r} \right\rfloor.$$

- Then, given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbb{P}[h_{\mathbf{a}, b}(\mathbf{x}) = h_{\mathbf{a}, b}(\mathbf{y})] = \int_0^r \frac{1}{c} f\left(\frac{1}{c}\right) \left(1 - \frac{t}{c}\right) dt$$

is decreasing with $c = \|\mathbf{x} - \mathbf{y}\|$.

Locality sensitive hashing

scalar quantization [Datar et al. '04]

- Choose a random vector \mathbf{a} from the d -dimensional Gaussian distribution $f = \mathcal{N}(0, 1)$ and a real b uniformly in $[0, r]$.
- Define *hash function* $h_{\mathbf{a}, b} : \mathbb{R}^d \rightarrow \mathbb{Z}$ with

$$h_{\mathbf{a}, b}(\mathbf{x}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{x} + b}{r} \right\rfloor.$$

- Then, given $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

$$\mathbb{P}[h_{\mathbf{a}, b}(\mathbf{x}) = h_{\mathbf{a}, b}(\mathbf{y})] = \int_0^r \frac{1}{c} f\left(\frac{1}{c}\right) \left(1 - \frac{t}{c}\right) dt$$

is decreasing with $c = \|\mathbf{x} - \mathbf{y}\|$.

Vector quantization

[Gray '84]

Construction

- given dataset $\mathcal{X} \subset \mathbb{R}^d$
- construct finite codebook $\mathcal{C} \subset \mathbb{R}^d$
- map (quantize) each point $\mathbf{x} \in \mathcal{X}$ to $q(\mathbf{x}) = \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2$
- discard dataset; represent each point by $\log k$ bits, where $k = |\mathcal{C}|$

Search (approximate, exhaustive)

- given query \mathbf{y}
- for each $\mathbf{c} \in \mathcal{C}$, compute and store distance $\|\mathbf{y} - \mathbf{c}\|^2$
- for each $\mathbf{x} \in \mathcal{X}$, approximate distance $\|\mathbf{y} - \mathbf{x}\|^2$ by $\|\mathbf{y} - q(\mathbf{x})\|^2$, which is looked up

Vector quantization

[Gray '84]

Construction

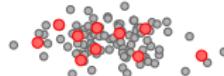
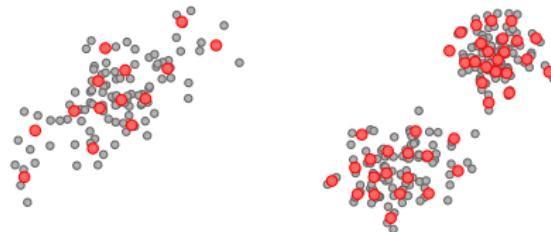
- given dataset $\mathcal{X} \subset \mathbb{R}^d$
- construct finite codebook $\mathcal{C} \subset \mathbb{R}^d$
- map (quantize) each point $\mathbf{x} \in \mathcal{X}$ to $q(\mathbf{x}) = \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2$
- discard dataset; represent each point by $\log k$ bits, where $k = |\mathcal{C}|$

Search (approximate, exhaustive)

- given query \mathbf{y}
- for each $\mathbf{c} \in \mathcal{C}$, compute and store distance $\|\mathbf{y} - \mathbf{c}\|^2$
- for each $\mathbf{x} \in \mathcal{X}$, approximate distance $\|\mathbf{y} - \mathbf{x}\|^2$ by $\|\mathbf{y} - q(\mathbf{x})\|^2$, which is looked up

Vector quantization

[Gray '84]



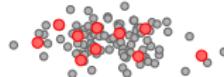
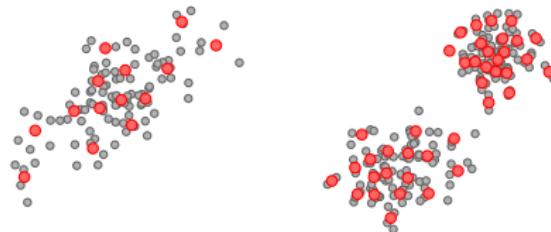
$$\text{minimize } E(\mathcal{C}) = \sum_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2 = \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - q(\mathbf{x})\|^2$$

distortion dataset codebook quantizer

The equation shows the minimization of distortion $E(\mathcal{C})$. Red arrows point from the labels "distortion", "dataset", and "codebook" to the corresponding terms in the equation. A pink arrow points from the label "quantizer" to the term $q(\mathbf{x})$.

Vector quantization

[Gray '84]

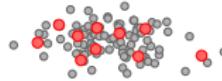
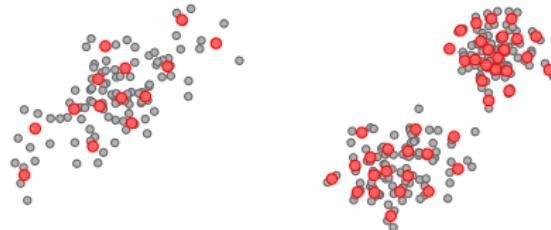


$$\text{minimize } E(\mathcal{C}) = \sum_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2 = \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - q(\mathbf{x})\|^2$$

distortion dataset codebook quantizer

Vector quantization

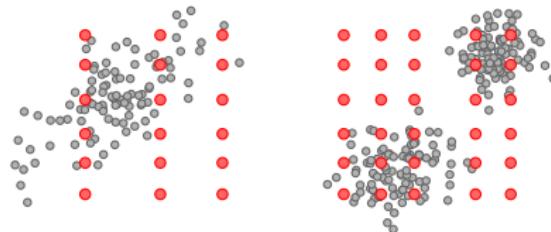
[Gray '84]



- For small distortion \rightarrow large $k = |\mathcal{C}|$:
 - hard to train
 - too large to store
 - too slow to search

Product quantization

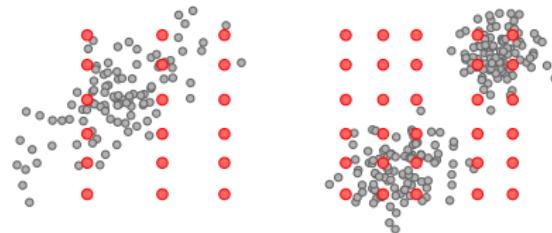
[Jégou et al. '11]



$$\begin{aligned} & \text{minimize} && \sum_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2 \\ & \text{subject to} && \mathcal{C} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m \end{aligned}$$

Product quantization

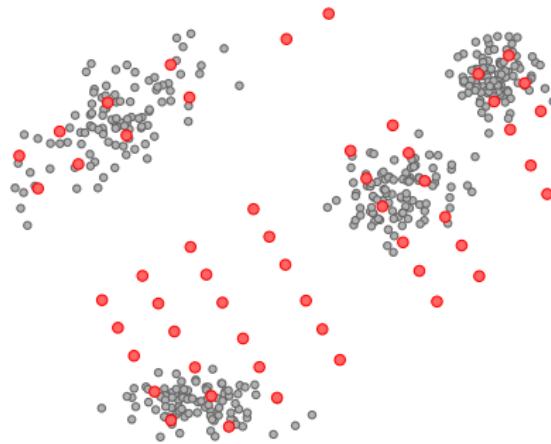
[Jégou et al. '11]



- train: $q = (q^1, \dots, q^m)$ where q^1, \dots, q^m obtained by VQ
- store: $|\mathcal{C}| = k^m$ with $|\mathcal{C}^1| = \dots = |\mathcal{C}^m| = k$
- search: $\|\mathbf{y} - q(\mathbf{x})\|^2 = \sum_{j=1}^m \|\mathbf{y}^j - q^j(\mathbf{x}^j)\|^2$ where $q^j(\mathbf{x}^j) \in \mathcal{C}^j$

Optimized product quantization

[Ge et al. '13]



$$\begin{aligned} & \text{minimize}_{\mathbf{x} \in \mathcal{X}} \sum_{\hat{\mathbf{c}} \in \hat{\mathcal{C}}} \|\mathbf{x} - R^\top \hat{\mathbf{c}}\|^2 \\ & \text{subject to} \quad \hat{\mathcal{C}} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m \\ & \quad R^\top R = I \end{aligned}$$

Optimized product quantization

Non-parametric solution

rotate: $\hat{X} \leftarrow RX$

update: $q \leftarrow \text{PQ}(\hat{X})$ [one step]

assign: $Y \leftarrow q(\hat{X})$

align: $R \leftarrow UV^\top$ where $YX^\top = USV^\top$

- From PQ only one step of centroid update is needed, because update of R does not alter assignment.
- Alignment minimizes $\|Y - RX\|_F^2$, as in ITQ.

Optimized product quantization

Non-parametric solution

rotate: $\hat{X} \leftarrow RX$

update: $q \leftarrow PQ(\hat{X})$ [one step]

assign: $Y \leftarrow q(\hat{X})$

align: $R \leftarrow UV^\top$ where $YX^\top = USV^\top$

- From PQ only one step of centroid update is needed, because update of R does not alter assignment.
- Alignment minimizes $\|Y - RX\|_F^2$, as in ITQ.

Optimized product quantization

Parametric solution for $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$

- From rate-distortion theory, distortion satisfies

$$E \geq k^{-2/d} d |\Sigma|^{1/d}$$

and practical distortion achieved by k -means is typically within $\sim 5\%$ of the bound. So after rotation $\hat{\Sigma} = R\Sigma R^T$,

$$E_{\text{PQ}} \geq k^{-2m/d} \frac{d}{m} \sum_{i=1}^m |\hat{\Sigma}_{ii}|^{m/d}$$

- But, by *arithmetic-geometric means* and *Fisher's inequalities*,

$$\frac{1}{m} \sum_{i=1}^m |\hat{\Sigma}_{ii}|^{m/d} \geq \prod_{i=1}^m |\hat{\Sigma}_{ii}|^{1/d} \geq |\hat{\Sigma}|^{1/d} = |\Sigma|^{1/d}$$

with equality implying **balanced variance** and **independence**.

Optimized product quantization

Parametric solution for $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$

- From rate-distortion theory, distortion satisfies

$$E \geq k^{-2/d} d |\Sigma|^{1/d}$$

and practical distortion achieved by k -means is typically within $\sim 5\%$ of the bound. So after rotation $\hat{\Sigma} = R\Sigma R^\top$,

$$E_{\text{PQ}} \geq k^{-2m/d} \frac{d}{m} \sum_{i=1}^m |\hat{\Sigma}_{ii}|^{m/d}$$

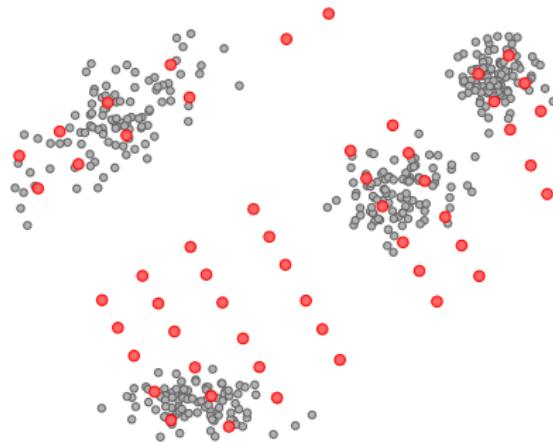
- But, by *arithmetic-geometric means* and *Fisher's inequalities*,

$$\frac{1}{m} \sum_{i=1}^m |\hat{\Sigma}_{ii}|^{m/d} \geq \prod_{i=1}^m |\hat{\Sigma}_{ii}|^{1/d} \geq |\hat{\Sigma}|^{1/d} = |\Sigma|^{1/d}$$

with equality implying **balanced variance** and **independence**.

Optimized product quantization

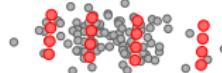
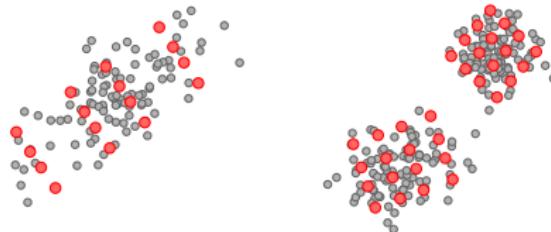
Parametric solution for $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$



- **independence:** PCA-align by diagonalizing Σ as $U\Lambda U^\top$
- **balanced variance:** permute Λ by π such that $\prod_i \lambda_i$ is constant in each subspace; $R \leftarrow UP_\pi^\top$
- find $\hat{\mathcal{C}}$ by PQ on rotated data $\hat{X} = RX$

Locally optimized product quantization

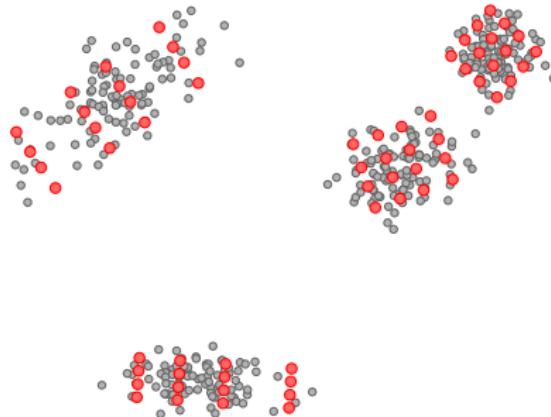
[Kalantidis & Avrithis '14]



- compute residuals $r(\mathbf{x}) = \mathbf{x} - Q(\mathbf{x})$ on coarse quantizer Q
- collect residuals $\mathcal{Z}_i = \{r(\mathbf{x}) : Q(\mathbf{x}) = \mathbf{c}_i\}$ per cell
- train $(R_i, q_i) \leftarrow \text{OPQ}(\mathcal{Z}_i)$ per cell

Locally optimized product quantization

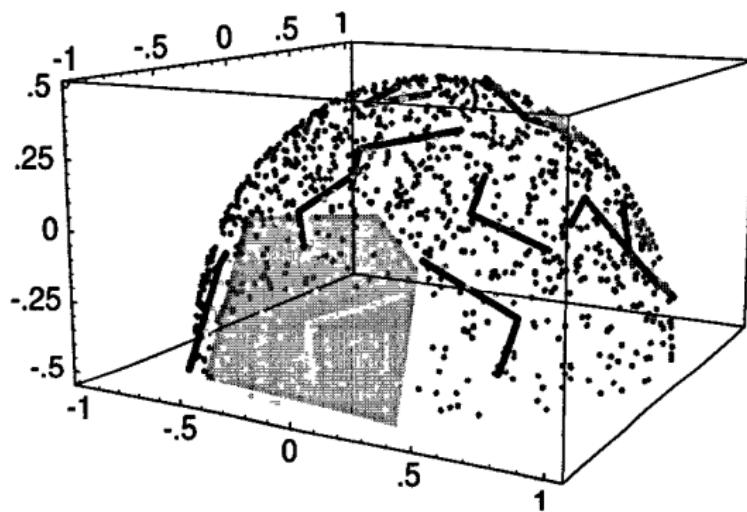
[Kalantidis & Avrithis '14]



- residual distributions closer to Gaussian assumption
- better captures the support of data distribution, like local PCA
 - multimodal (e.g. mixture) distributions
 - distributions on nonlinear manifolds

Local principal component analysis

[Kambhatla & Leen '97]



But, we are not doing dimensionality reduction!

IV. Non-exhaustive search

Inverted index

IVFADC [Jégou et al. '11]

Construction

- train a coarse quantizer Q of K centroids or **cells**
- quantize each point $\mathbf{x} \in \mathcal{X}$ to $Q(\mathbf{x})$ and compute its **residual vector**
 $r(\mathbf{x}) = \mathbf{x} - Q(\mathbf{x})$
- quantize residuals by a product quantizer q
- for each cell, maintain an **inverted list** of data points and PQ-encoded residuals

Search

- quantize query \mathbf{y} to w nearest cells
- exhaustively search by PQ only within the w inverted lists

Inverted index

IVFADC [Jégou et al. '11]

Construction

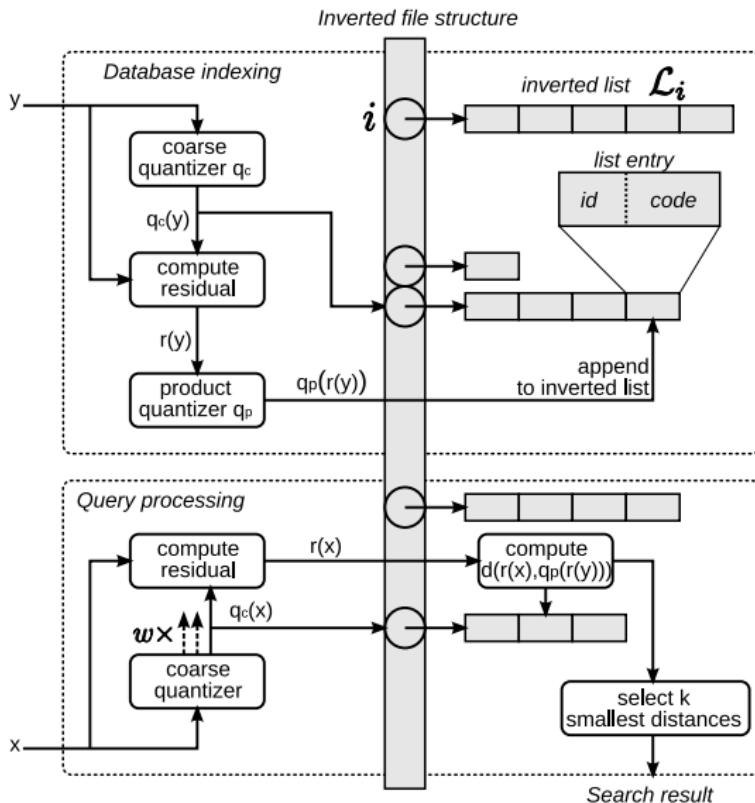
- train a coarse quantizer Q of K centroids or **cells**
- quantize each point $\mathbf{x} \in \mathcal{X}$ to $Q(\mathbf{x})$ and compute its **residual vector**
 $r(\mathbf{x}) = \mathbf{x} - Q(\mathbf{x})$
- quantize residuals by a product quantizer q
- for each cell, maintain an **inverted list** of data points and PQ-encoded residuals

Search

- quantize query \mathbf{y} to w nearest cells
- exhaustively search by PQ only within the w inverted lists

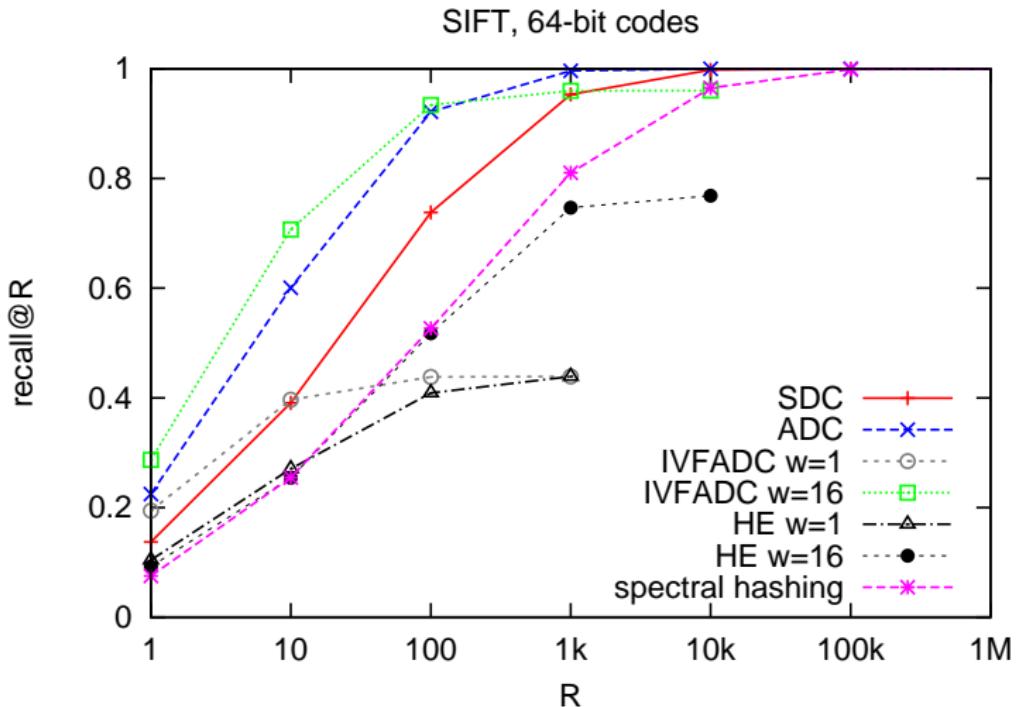
Inverted index

IVFADC [Jégou et al. '11]



Product quantization

Result on SIFT1M



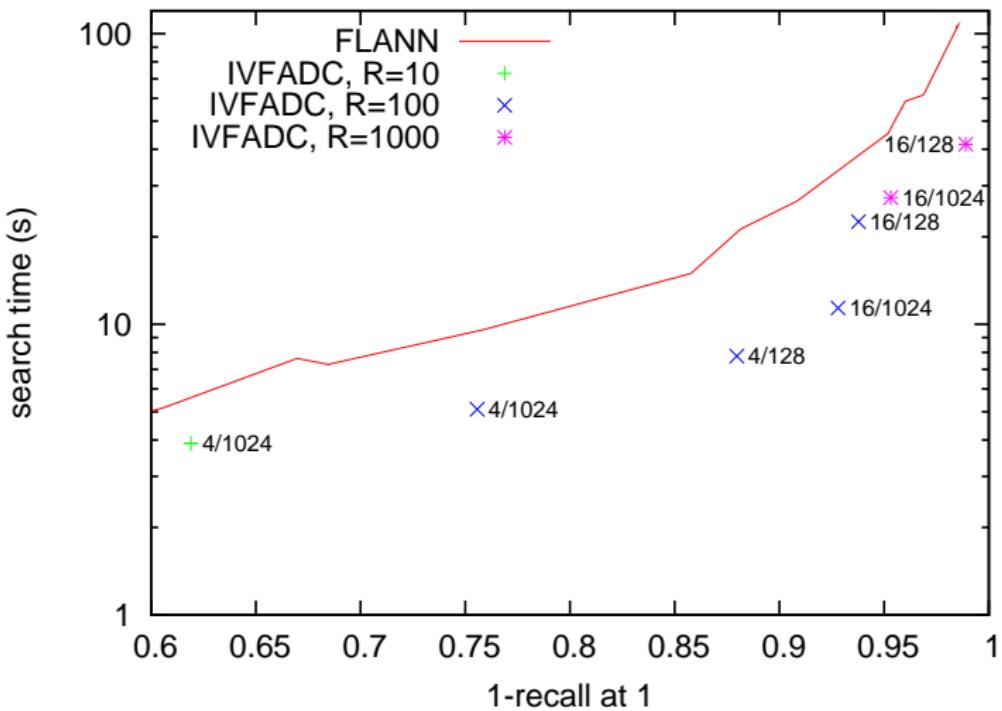
Product quantization

vs. FLANN on SIFT1M

- keep uncompressed vectors \mathcal{X} in memory
- find the R top-ranking points by IVFADC
- re-rank according to corresponding uncompressed vectors

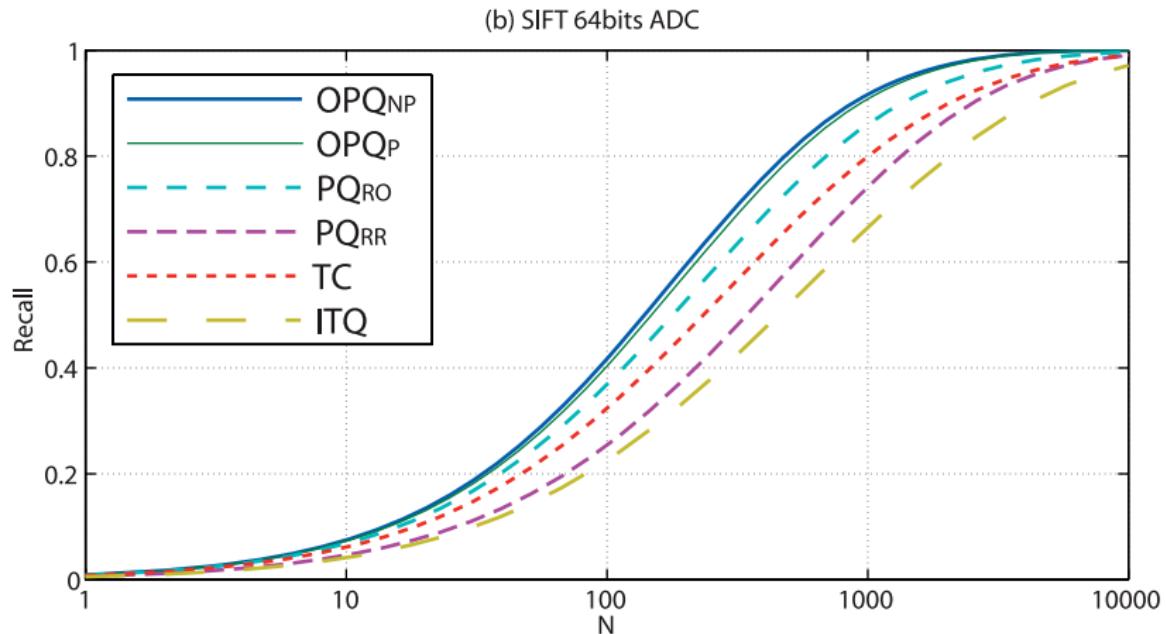
Product quantization

vs. FLANN on SIFT1M



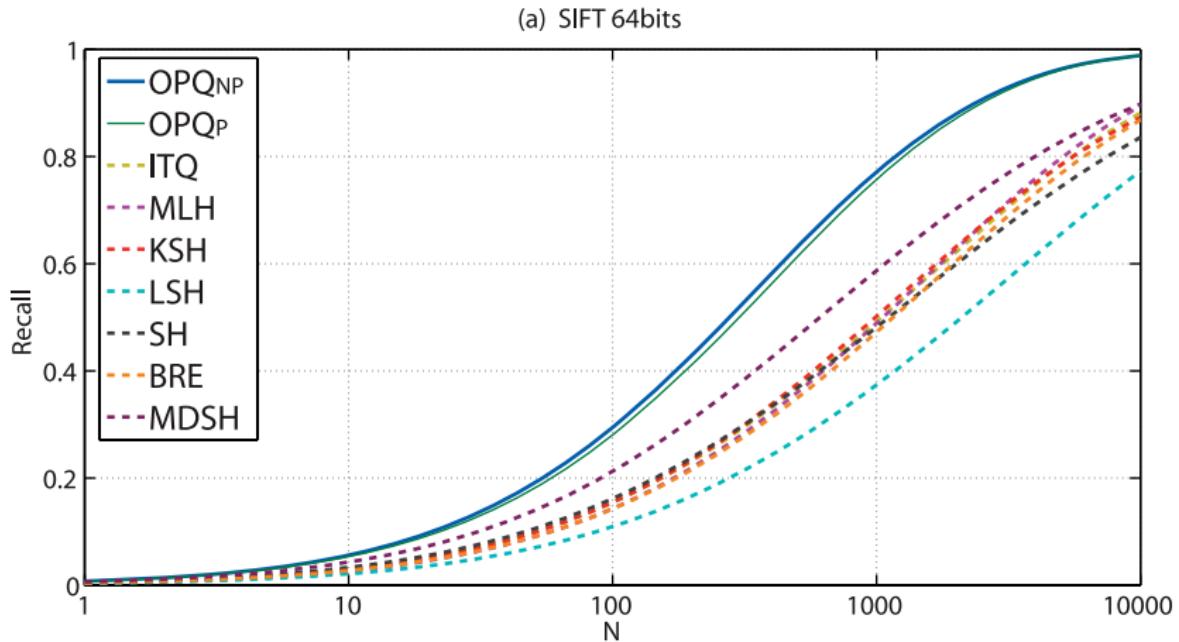
Optimized product quantization

Result on SIFT1M



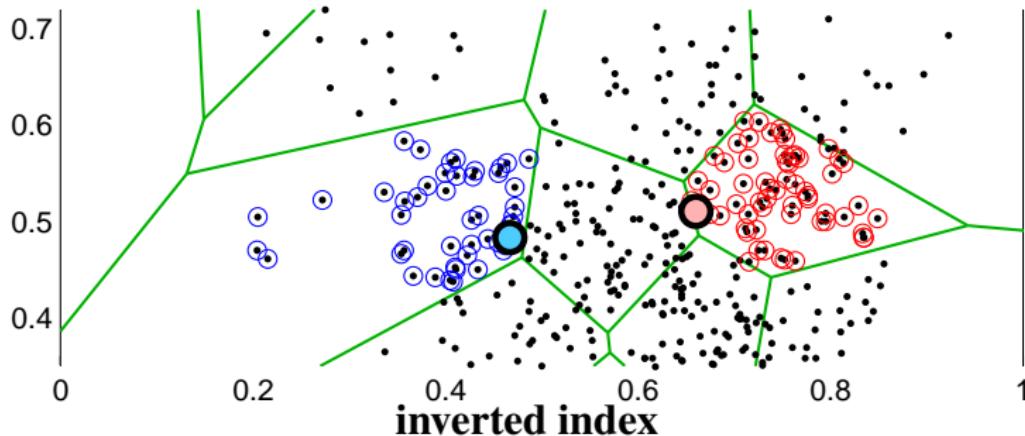
Optimized product quantization

vs. binary codes on SIFT1M



Inverted multi-index

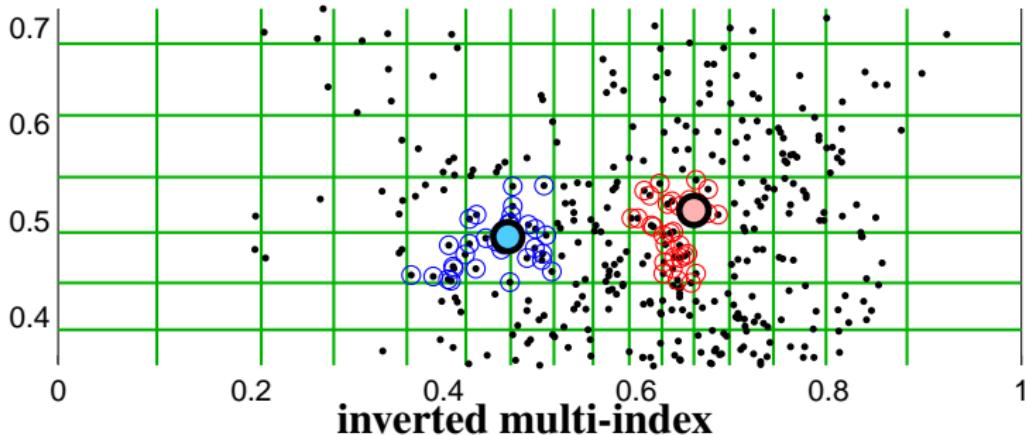
[Babenko & Lempitsky '12]



- train codebook \mathcal{C} from dataset $\{\mathbf{x}_n\}$
- this codebook provides a **coarse** partition of the space

Inverted multi-index

[Babenko & Lempitsky '12]

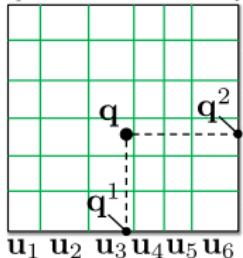


- decompose vectors as $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2)$
- train codebooks $\mathcal{C}^1, \mathcal{C}^2$ from datasets $\{\mathbf{x}_n^1\}, \{\mathbf{x}_n^2\}$
- induced codebook $\mathcal{C}^1 \times \mathcal{C}^2$ gives a **finer** partition
- given query \mathbf{y} , visit cells $(\mathbf{c}^1, \mathbf{c}^2) \in \mathcal{C}^1 \times \mathcal{C}^2$ in ascending order of distance to \mathbf{y}

Inverted multi-index

Multi-sequence algorithm

space subdivision via PQ



71
72
73
74
75
76

product
antization

q^1 vs. \mathcal{U}

i	$\mathbf{u}_{\alpha(i)}$	r
1	\mathbf{u}_3	0.5
2	\mathbf{u}_4	0.7
3	\mathbf{u}_5	4
4	\mathbf{u}_2	6
5	\mathbf{u}_1	8
6	\mathbf{u}_6	9

q^2 vs. ν

j	$\mathbf{v}_{\beta(j)}$	s
1	\mathbf{v}_4	0.1
2	\mathbf{v}_3	2
3	\mathbf{v}_5	3
4	\mathbf{v}_2	6
5	\mathbf{v}_6	7
6	\mathbf{v}_1	11

multi-sequence algorithm

$[u_{\alpha(i)} v_{\beta(j)}]$	(i, j)	$r(i) + s(j)$
$u_3 v_4$	(1,1)	0.6 (0.5+0.1)
$u_4 v_4$	(2,1)	0.8 (0.7+0.1)
$u_3 v_3$	(1,2)	2.5 (0.5+2)
$u_4 v_3$	(2,2)	2.7 (0.7+2)
$u_3 v_5$	(1,3)	3.5 (0.5+3)
$u_4 v_5$	(2,3)	3.7 (0.7+3)
$u_5 v_4$	(3,1)	4.1 (4+0.1)
$u_5 v_3$	(3,2)	6 (4+2)
$u_3 v_2$	(1,4)	6.5 (0.5+6)
	...	

	1	2	3	4	5	6
1	0.6	0.8	4.1	6.1	8.1	9.1
2	2.5	2.7	6	8	10	11
3	3.5	3.7	7	9	11	12
4	6.5	6.7	10	12	14	15
5	7.5	7.7	11	13	15	16
6	11.5	11.7	15	17	19	20

1	2	3	4	5	6
0.6	0.8	4.1	6.1	8.1	9.1
2.5	2.7	6	8	10	11
3.5	3.7	7	9	11	12
6.5	6.7	10	12	14	15
7.5	7.7	11	13	15	16
11.5	11.7	15	17	19	20

1	2	3	4	5	6
0.6	0.8	4.1	6.1	8.1	9.1
2.5	2.7	6	8	10	11
3.5	3.7	7	9	11	12
6.5	6.7	10	12	14	15
7.5	7.7	11	13	15	16
11.5	11.7	15	17	19	20

1	2	3	4	5	6
0.6	0.8	4.1	6.1	8.1	9.1
2.5	2.7	6	8	10	11
3.5	3.7	7	9	11	12
6.5	6.7	10	12	14	15
7.5	7.7	11	13	15	16
11.5	11.7	15	17	19	20

1	2	3	4	5
0.6	0.8	4.1	6.1	8.1
2.5	2.7	6	8	10
3.5	3.7	7	9	11
6.5	6.7	10	12	14
7.5	7.7	11	13	15
11.5	11.7	15	17	19

	1	2	3	4	5	6	
1	0.6	0.8	4.1	6.1	8.1	9.1	V ₄
1	2.5	2.7	6	8	10	11	V ₃
2	3.5	3.7	7	9	11	12	V ₂
5	6.5	6.7	10	12	14	15	V ₂
6	7.5	7.7	11	13	15	16	V ₀
0	11.5	11.7	15	17	19	20	V ₁

OUTPUT:

$$(1, 1) \rightarrow \mathbf{W}_{3,4}$$

$(2, 1) \rightarrow \mathbf{W}_{4,4}$

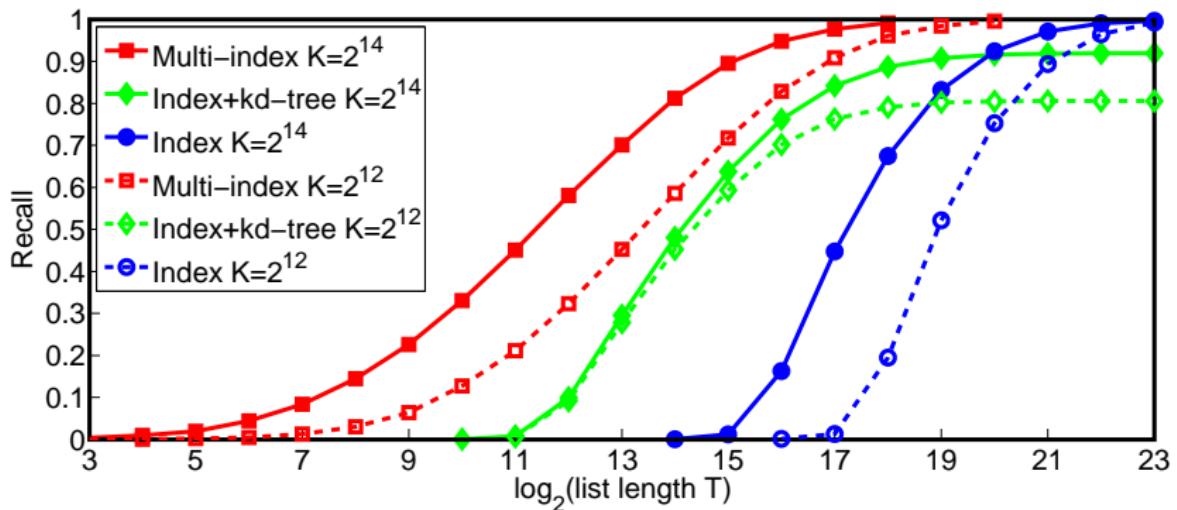
$$(1, 2) \rightarrow W_{3,3}$$

$$(2,2) \rightarrow \mathbf{W}_4,$$

$$(1,3) \rightarrow W_{3,5}$$

Inverted multi-index

Result on SIFT1B: are NN in candidate lists?



Locally optimized product quantization

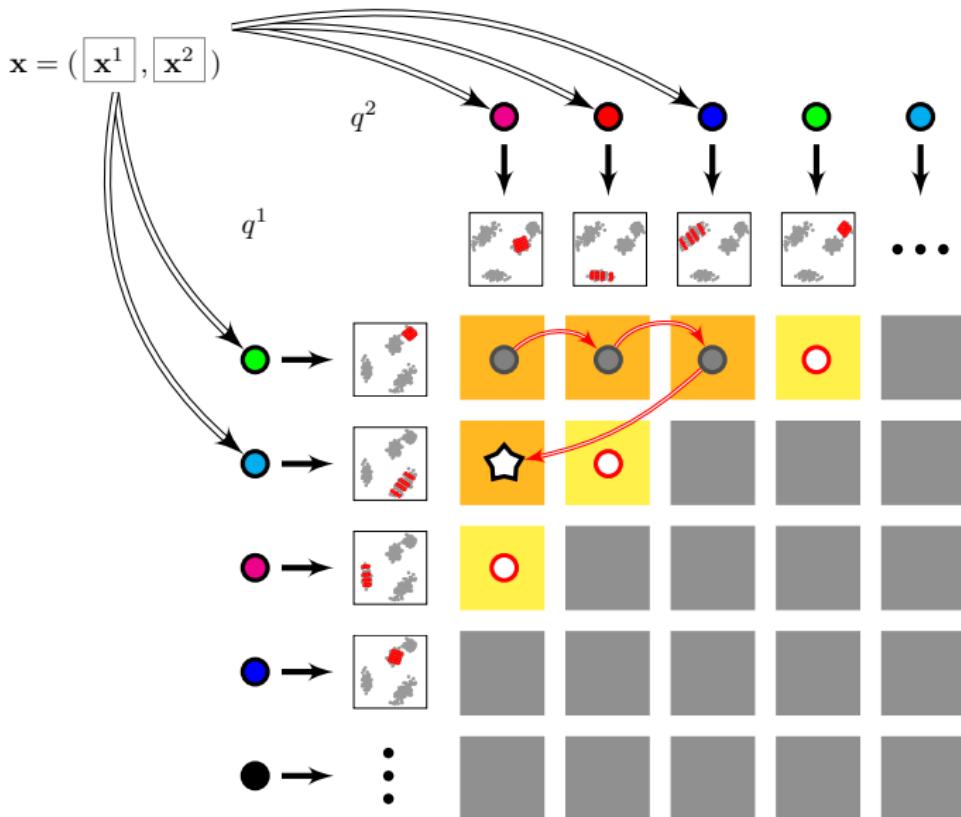
Result on SIFT1B, 64-bit codes

Method	$R = 1$	$R = 10$	$R = 100$
Ck-means [Norouzi & Fleet '13]	–	–	0.649
IVFADC [Jégou <i>et al.</i> '11]	0.106	0.379	0.748
IVFADC [Jégou <i>et al.</i> '11]	0.088	0.372	0.733
OPQ [Ge <i>et al.</i> '13]	0.114	0.399	0.777
Multi-D-ADC [Babenko & Lempitsky '12]	0.165	0.517	0.860
LOR+PQ [Kalantidis & Avrithis '14]	0.183	0.565	0.889
LOPQ [Kalantidis & Avrithis '14]	0.199	0.586	0.909

Most benefit comes from locally optimized rotation!

Multi-LOPQ

[Kalantidis & Avrithis '14]



Multi-LOPQ

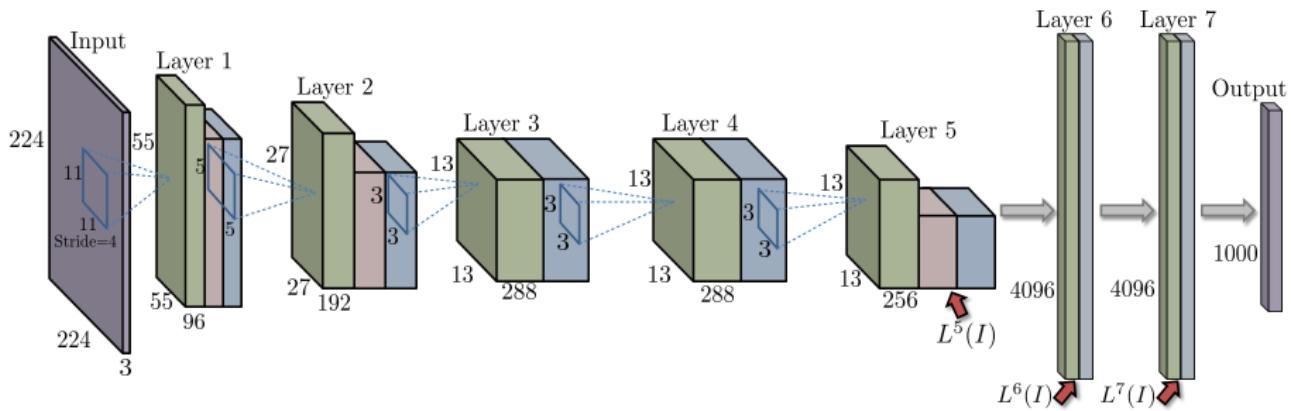
Result on SIFT1B, 128-bit codes

T	Method	$R = 1$	10	100
20K	IVFADC+R [Jégou <i>et al.</i> '11]	0.262	0.701	0.962
	LOPQ+R [Kalantidis & Avrithis '14]	0.350	0.820	0.978
10K	Multi-D-ADC [Babenko & Lempitsky '12]	0.304	0.665	0.740
	OMulti-D-OADC [Ge <i>et al.</i> '13]	0.345	0.725	0.794
	Multi-LOPQ [Kalantidis & Avrithis '14]	0.430	0.761	0.782
30K	Multi-D-ADC [Babenko & Lempitsky '12]	0.328	0.757	0.885
	OMulti-D-OADC [Ge <i>et al.</i> '13]	0.366	0.807	0.913
	Multi-LOPQ [Kalantidis & Avrithis '14]	0.463	0.865	0.905
100K	Multi-D-ADC [Babenko & Lempitsky '12]	0.334	0.793	0.959
	OMulti-D-OADC [Ge <i>et al.</i> '13]	0.373	0.841	0.973
	Multi-LOPQ [Kalantidis & Avrithis '14]	0.476	0.919	0.973

Application: image search

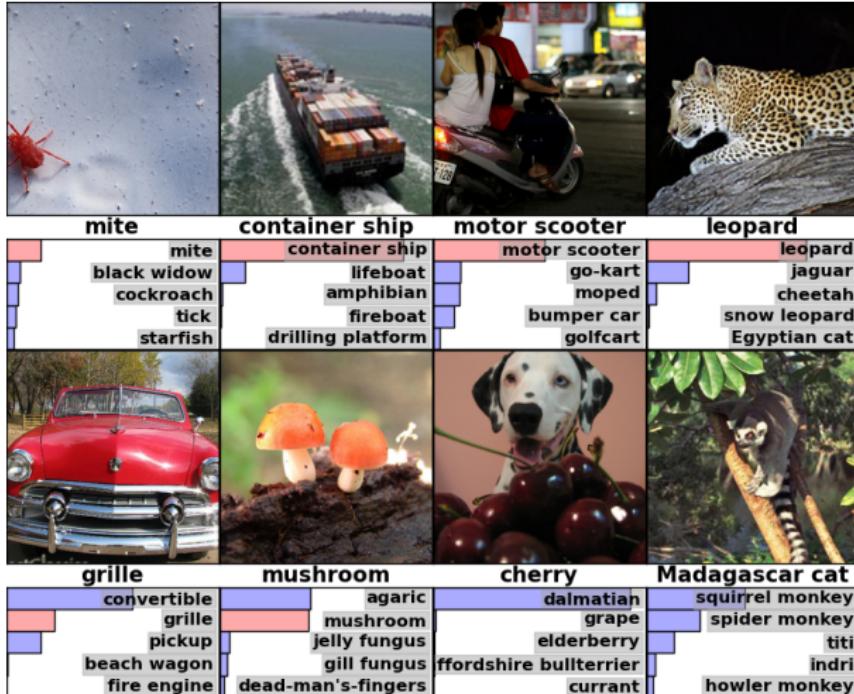
Deep learned image features

[Krizhevsky et al. '12]



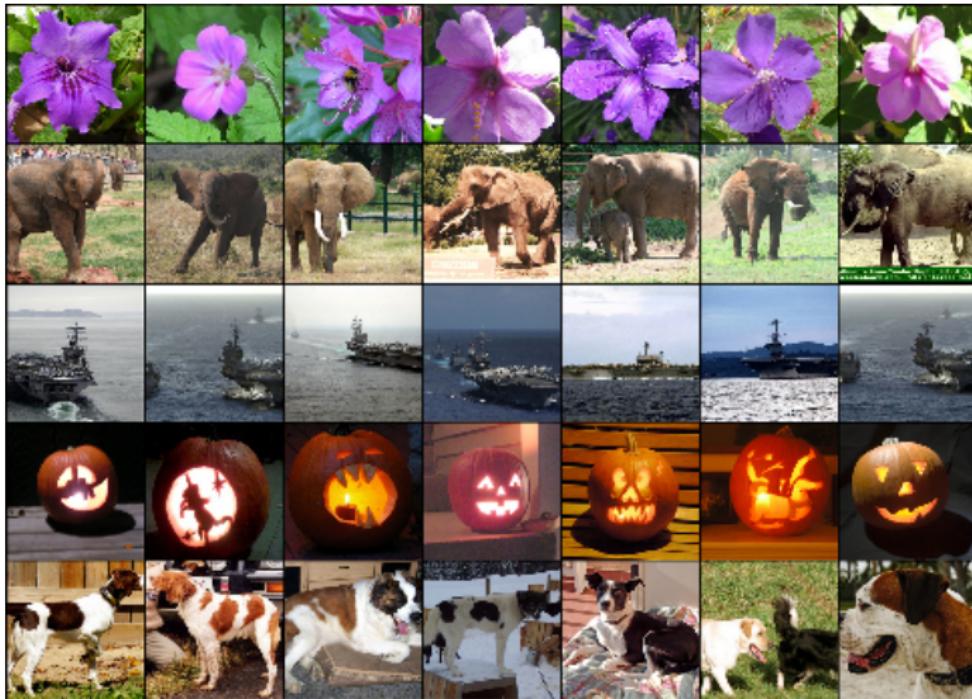
Deep learned image features

Classification



Deep learned image features

Search



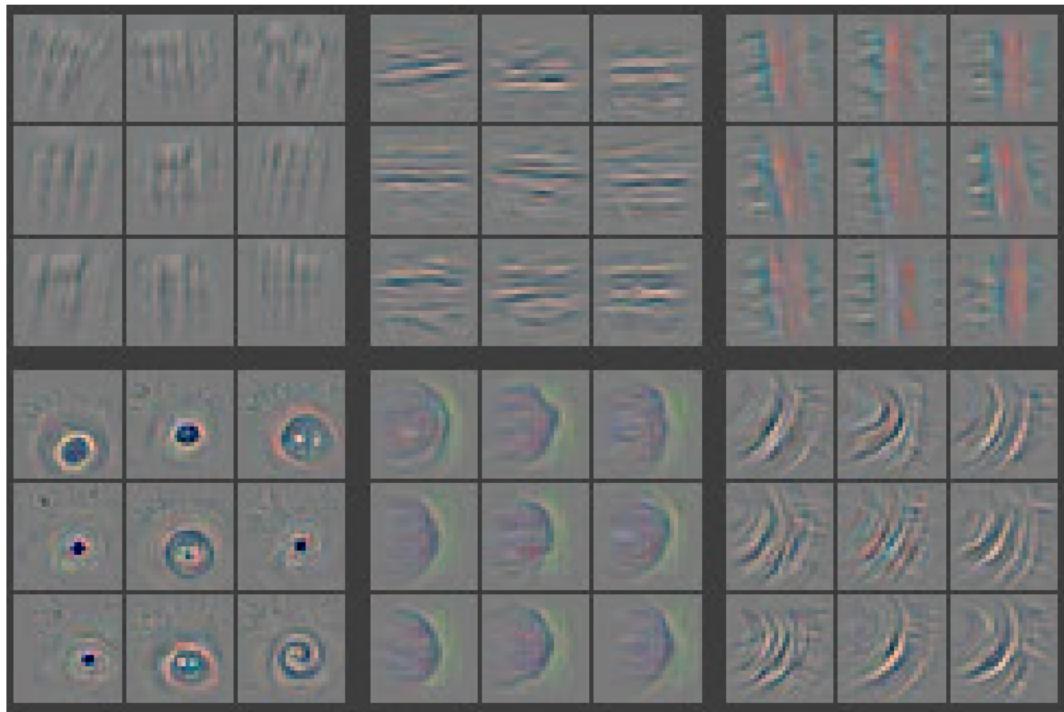
Deep learned image features

Layer 1 features



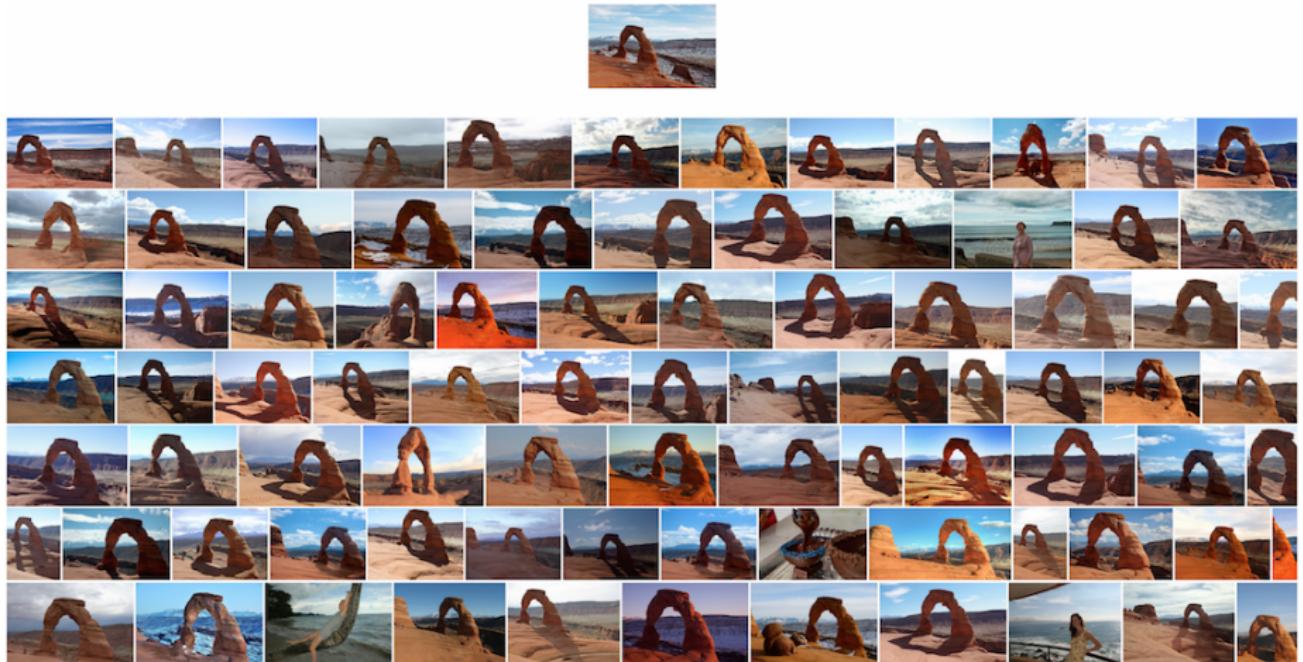
Deep learned image features

Layer 2 features



Multi-LOPQ

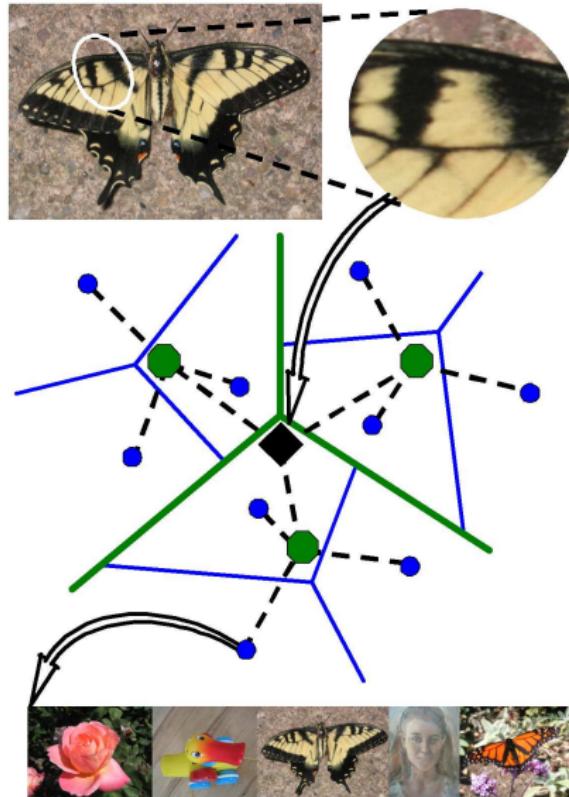
Image query on Flickr 100M (deep learned features, 4k → 128 dimensions)



V. Clustering

Hierarchical k -means

[Nister & Stewenius '06]



Approximate k -means

[Philbin et al. '07]

- centroids updated as in k -means
- points assigned to centroids by approximate search
- search by randomized k -d trees, even before the latter was published or FLANN was available
- index rebuilt in every k -means iteration

Approximate k -means

vs. Hierarchical k -means

Method	Dataset	mAP	
		Bag-of-words	Spatial
(a) HKM-1	5K	0.439	0.469
(b) HKM-2	5K	0.418	
(c) HKM-3	5K	0.372	
(d) HKM-4	5K	0.353	
(e) AKM	5K	0.618	0.647
(f) AKM	5K+100K	0.490	0.541
(g) AKM	5K+100K+1M	0.393	0.465

Robust approximate k -means

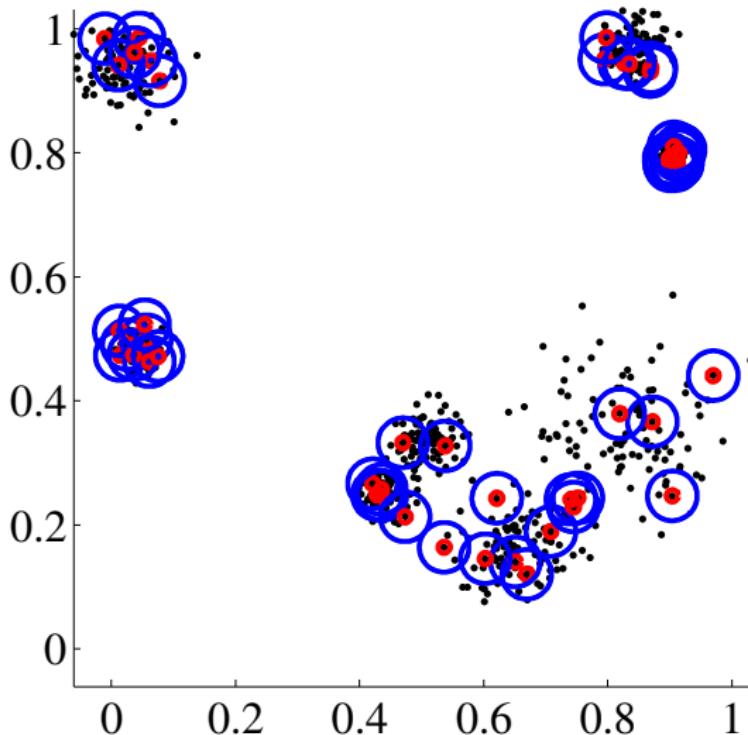
[Li et al. '10]

- the nearest neighbor in one iteration is re-used in the next
- less effort spent for new neighbor search
- faster convergence at same quality

Approximate Gaussian mixtures

[Kalantidis & Avrithis '12]

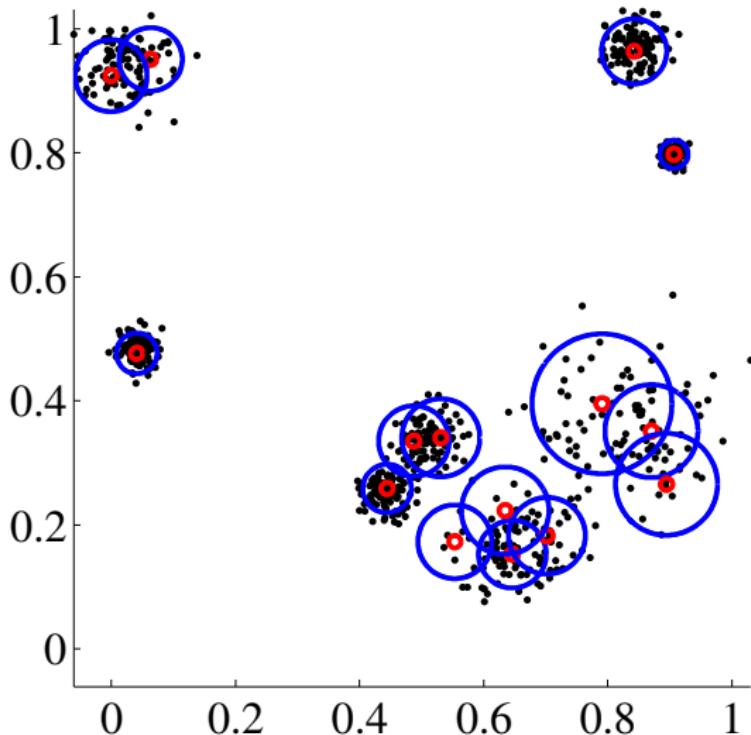
iteration=0, clusters=50



Approximate Gaussian mixtures

[Kalantidis & Avrithis '12]

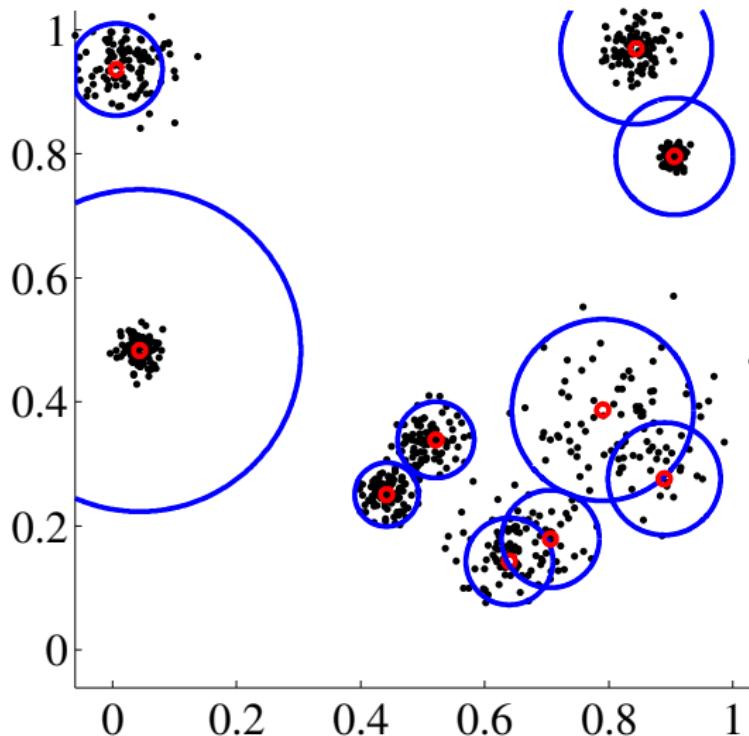
iteration=1, clusters=15



Approximate Gaussian mixtures

[Kalantidis & Avrithis '12]

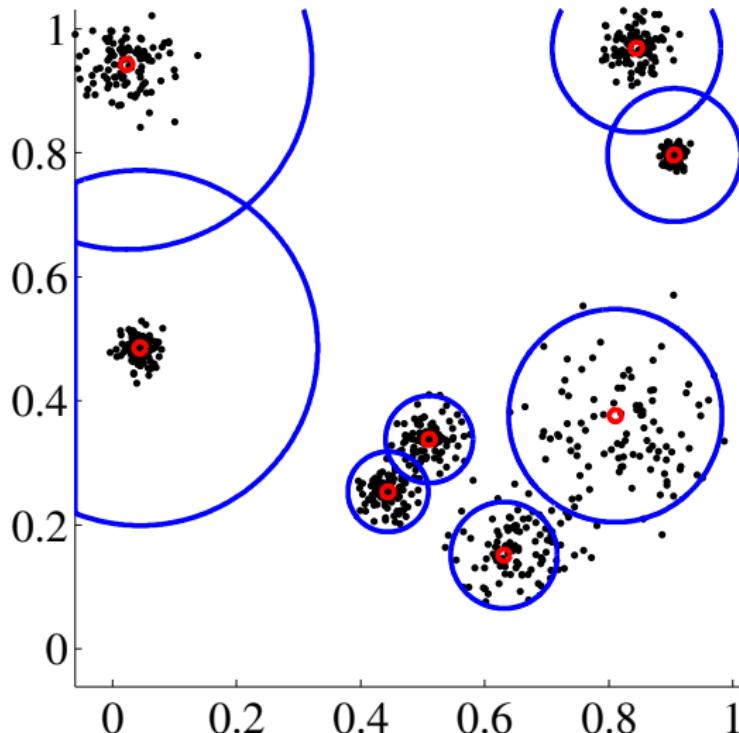
iteration=2, clusters=10



Approximate Gaussian mixtures

[Kalantidis & Avrithis '12]

iteration=3, clusters=8



Approximate Gaussian mixtures

Image search—mAP on Oxford 5k

Method	RAKM					AKM	AGM
k	350k	500k	550k	600k	700k	550k	857k
5k	0.471	0.479	0.486	0.485	0.476	0.485	0.492
5k + 20k	0.439	0.440	0.448	0.441	0.437	0.447	0.459
5k + 1M	—	—	0.250	—	—	—	0.280

ANN search - clustering connection

- *hierarchical k-means*: use k -means tree for ANN search
- *approximate k-means*: use ANN search to accelerate assignment step
- *product quantization*: use k -means on subspaces to accelerate ANN search
- *inverted multi-index*: exhaustively search on subspaces before searching on entire space

What is the actual connection? Can we use recursion to solve both problems at the same time?

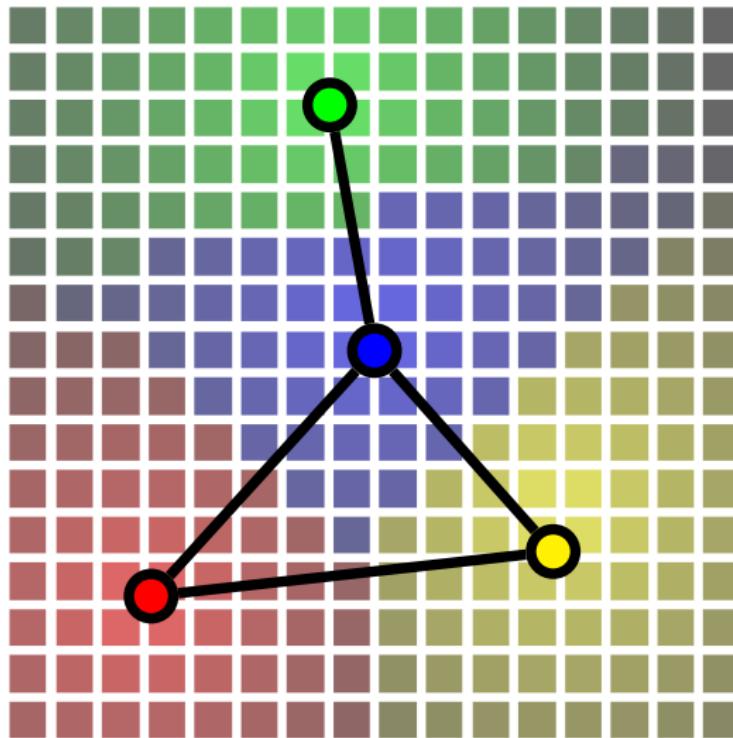
ANN search - clustering connection

- *hierarchical k-means*: use k -means tree for ANN search
- *approximate k-means*: use ANN search to accelerate assignment step
- *product quantization*: use k -means on subspaces to accelerate ANN search
- *inverted multi-index*: exhaustively search on subspaces before searching on entire space

What is the actual connection? Can we use recursion to solve both problems at the same time?

Dimensionality-recursive vector quantization

[Avrithis '13]



Dimensionality-recursive vector quantization

[Avrithis '13]

Problem

- given n points in d dimensions, quantize to k centroids under minimal distortion, with $n > 10^6, d > 10^2, k > 10^3$.

Bottleneck: k -means assignment

- exhaustive search: $O(nk)$ time
- approximate search: e.g. , $O(n \log k)$.

Lookup?

- n queries over the *same* centroids
- why not lookup on precomputed distance maps?
- $O(n)$ time, but $O(2^d)$ space: fine e.g. for $d = 2$.

Curse of dimensionality

- what if $d > 10$? is then lookup possible?
- $O(k^2 \log k)$ pre-processing, $O(n)$ time to assign, at $O(k^2)$ space.

Dimensionality-recursive vector quantization

[Avrithis '13]

Problem

- given n points in d dimensions, quantize to k centroids under minimal distortion, with $n > 10^6, d > 10^2, k > 10^3$.

Bottleneck: k -means assignment

- exhaustive search: $O(nk)$ time
- approximate search: e.g. , $O(n \log k)$.

Lookup?

- n queries over the *same* centroids
- why not lookup on precomputed distance maps?
- $O(n)$ time, but $O(2^d)$ space: fine e.g. for $d = 2$.

Curse of dimensionality

- what if $d > 10$? is then lookup possible?
- $O(k^2 \log k)$ pre-processing, $O(n)$ time to assign, at $O(k^2)$ space.

Dimensionality-recursive vector quantization

[Avrithis '13]

Problem

- given n points in d dimensions, quantize to k centroids under minimal distortion, with $n > 10^6, d > 10^2, k > 10^3$.

Bottleneck: k -means assignment

- exhaustive search: $O(nk)$ time
- approximate search: e.g. , $O(n \log k)$.

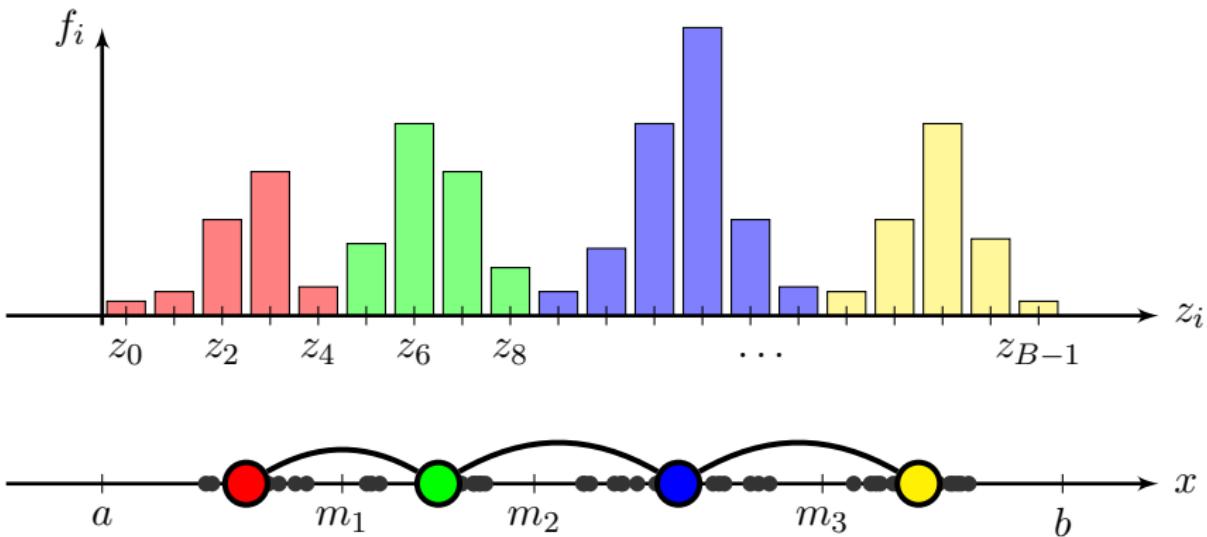
Lookup?

- n queries over the *same* centroids
- why not lookup on precomputed distance maps?
- $O(n)$ time, but $O(2^d)$ space: fine e.g. for $d = 2$.

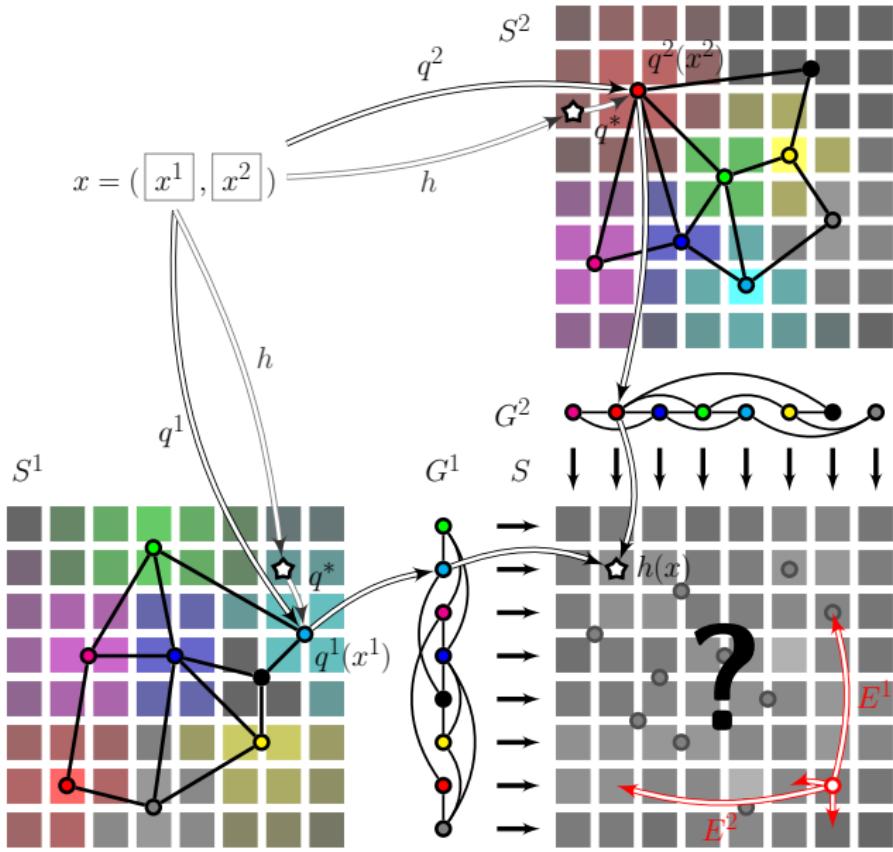
Curse of dimensionality

- what if $d > 10$? is then lookup possible?
- $O(k^2 \log k)$ pre-processing, $O(n)$ time to assign, at $O(k^2)$ space.

DRVQ base case: $d = 1$



DRVQ recursion: $d \rightarrow 2d$



DRVQ: vector quantization

k	16k	8k	4k	2k	1k	512
Approximate (μs)	0.95	0.83	0.80	0.73	0.80	0.90
Exact (ms)	1.19	0.79	0.51	0.26	0.21	0.11

averaged over the $n = 75k$ SIFT descriptors of the 55 cropped query images of *Oxford 5k*

DRVQ: clustering

k	$\log k_p$ ($d = 2^p$)						time (m)
	1	2	4	8	16	32	
16k	6	7	8	9	11	14	129.96
8k	6	7	8	9	11	13	119.43
4k	6	7	8	9	10	12	20.07
2k	5	6	7	8	9	11	2.792
1k	5	6	7	8	9	10	2.608
512	4	5	6	7	8	9	0.866
4k	Approximate k -means					504.2	

4 codebooks at $d = 32$ dimensions each on $n = 12.5M$ 128-dimensional SIFT descriptors of *Oxford 5k*

Approximate k -means

[Philbin et al. '07]

- centroids updated as in k -means
- points assigned to centroid by approximate search
- index rebuilt in every k -means iteration

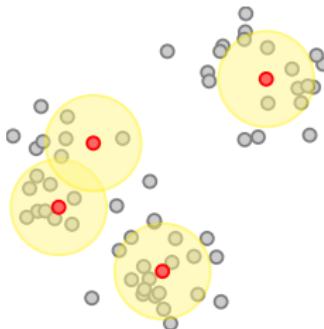
Ranked retrieval

[Broder et al. '14]

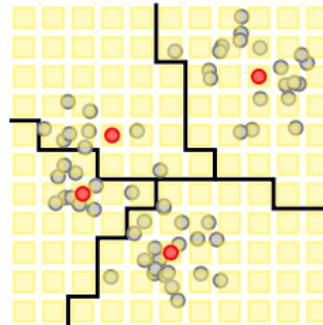
- centroids updated as in k -means
- points assigned by inverse search from **centroids** to points
- points may remain unassigned
- index built only once

Inverted-quantized k -means

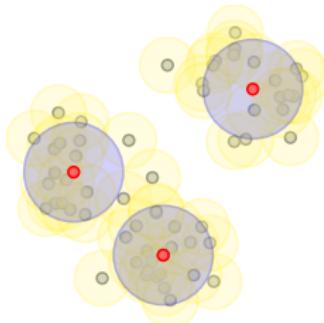
[unpublished '15]



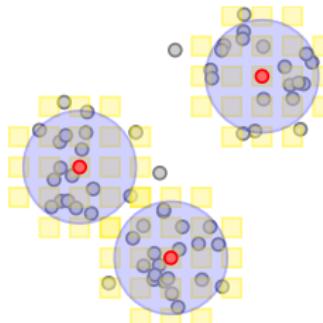
ranked retrieval



DRVQ



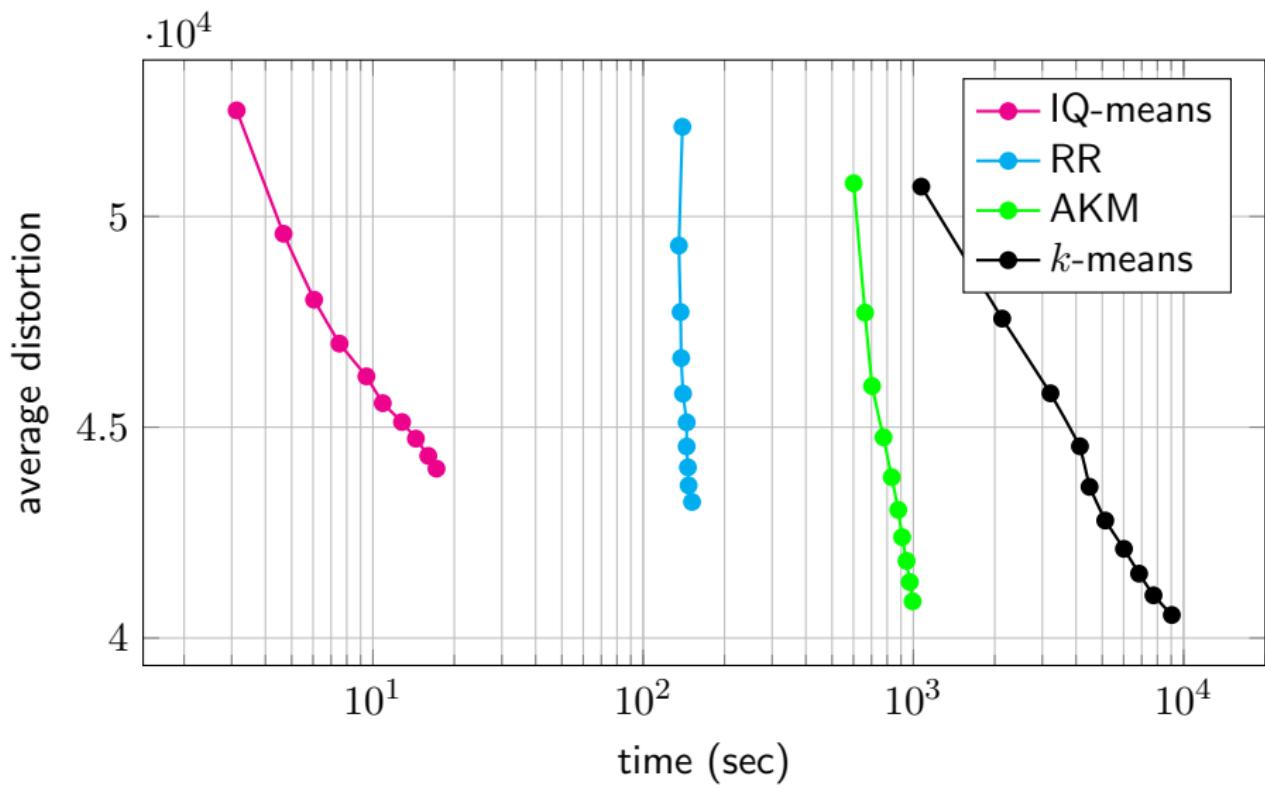
AGM



IQ-means

Inverted-quantized k -means

[unpublished '15]



<http://image.ntua.gr/iva/research/>



Thank you!