Example-Based Computer Vision
6.891

Greg Shakhnarovich

April 27, 2004

Today

- Example-based paradigm in vision
  - NN classification and regression
- algorithms for similarity search
  - kd-trees and Best Bin First seacrh
  - Locality-Sensitive Hashing
- Locally-Weighted Regression

Model-based vision

- Image $\mathbf{I}$ is produced by a parametric process $\mathbf{I} = F(\theta)$.
  - θ: object class, pose, illumination, activity, etc.
- Given $\mathbf{I}$, recover the relevant subset of θ.
- Typical paradigm: "Generate and test"

Model-based vision

- Advantages
  - Often an interpretable model
  - Relatively compact representation
- Disadvantages
  - May be computationally costly
  - Susceptible to local minima
  - It is very difficult to come up with a manageable model for a complex phenomenon

Example-based vision

- Large set of labeled examples $\langle \mathbf{x}_1, \theta_1 \rangle, \ldots, \langle \mathbf{x}_N, \theta_N \rangle$
- Distance measure $d(\mathbf{x}_a, \mathbf{x}_b)$

Example-based vision

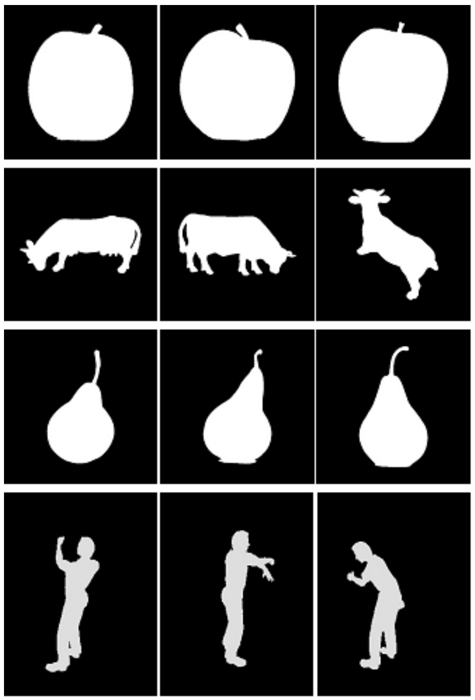
- Large set of labeled examples $\langle \mathbf{x}_1, \theta_1 \rangle, \ldots, \langle \mathbf{x}_N, \theta_N \rangle$
- Distance measure $d(\mathbf{x}_a, \mathbf{x}_b)$
- Approach: infer the unknown θ from the example(s) closest (most *similar*) to the query $\mathbf{x}_0$ w.r.t. d.

Example-based vision

- Large set of labeled examples $\langle \mathbf{x}_1, \theta_1 \rangle, \ldots, \langle \mathbf{x}_N, \theta_N \rangle$
- Distance measure $d(\mathbf{x}_a, \mathbf{x}_b)$
- Approach: infer the unknown θ from the example(s) closest (most *similar*) to the query $\mathbf{x}_0$ w.r.t. d.
- Intuition: similar images are likely to belong to the same class, or have the same parameters (class, pose etc.)
- Notion of "similarity" is problem-dependent, and critical to the success of an approach.

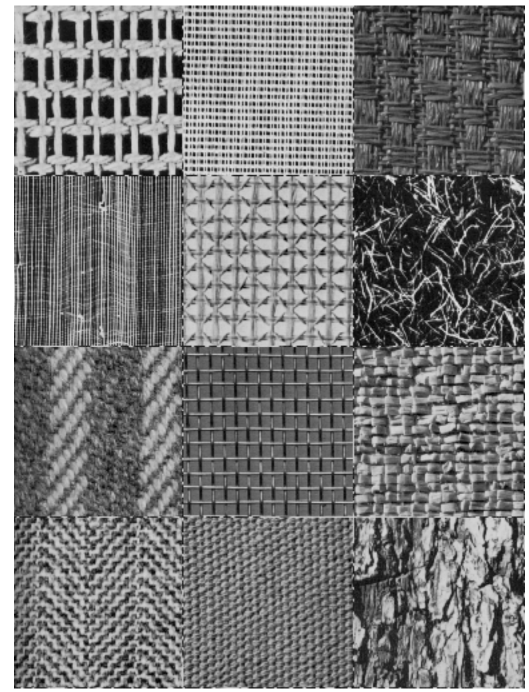
Examples: recognition

Shape



Multi-cue

Texture



Examples: estimation

Orientation

[Nyogi & Freeman '96]

Articulated pose

[Shakhnarovich *et al* '03]

Probabilistic approach to recognition

- C classes, with *prior* probabilities $P_1, \ldots, P_C$.
- Data from class c have distribution $p(\mathbf{x}|c) \equiv p_c(\mathbf{x})$.
- Risk of a classifier $h(\mathbf{x}) \to \{1, \ldots, C\}$:

$$R = E_{p(\mathbf{x},c)} \left[L\left(h(\mathbf{x}), c \right) \right],$$

where L is the *loss function*, e.g. 0/1 loss

$$L(c_1, c_2) = \begin{cases} 0, & \text{if} \quad c_1 = c_2, \\ 1 & \text{otherwise} \end{cases}$$

Recognition: optimal classifier

- Suppose we *know* p_c, P_c for all classes.
- Under 0/1 loss, the risk is minimized if

$$f^*(\mathbf{x}) = \operatorname*{argmax}_c p_c(\mathbf{x}) P_c.$$

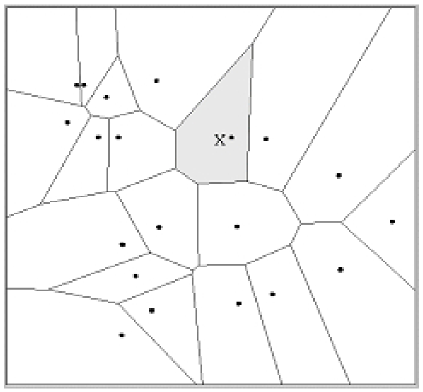
- This minimal risk R^* is called *Bayes risk* (of the problem).
- See [Duda, Hart & Stork (2nd edition)]

Nearest neighbors classification

- NN classifier: find $i = \operatorname{argmin}_j d(\mathbf{x}_0, \mathbf{x}_j)$.
- Theory [Cover & Hart '67]:
  - Binary classification: $\lim_{N\to\infty} R_N \leq 2R^*(1 - R^*)$.
  - C-class problem: $\lim_{N\to\infty} R_N = R^*\left(2 - \frac{C}{C-1}R^*\right)$
  - Arbitrarily slow rate of convergence to the asymptotic rate [Cover '68]
- Practice: often on par with the best classifier even with data sets of modest size.

Nearest Neighbor boundary

- NN decision boundaries set a *Voronoi tasselation*:

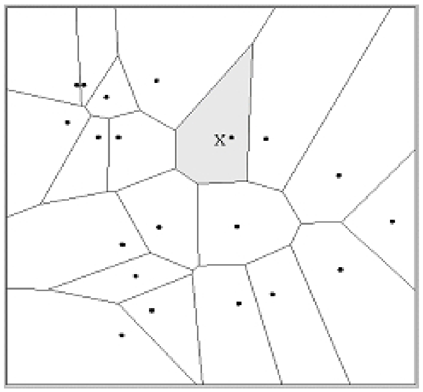


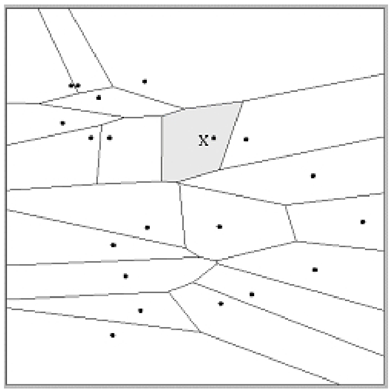
$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

- The distance defines the NN and thus the decision boundaries.

Nearest Neighbor boundary

- NN decision boundaries set a *Voronoi tasselation*:





$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \qquad \sqrt{(x_1 - y_1)^2 + (3x_2 - 3y_2)^2}$$

- The distance defines the NN and thus the decision boundaries.

Example: object categorization

- ETH data set: 8 categories $\times$ 10 objects $\times$ 41 viewpoints

[Leibe & Schiele '03]

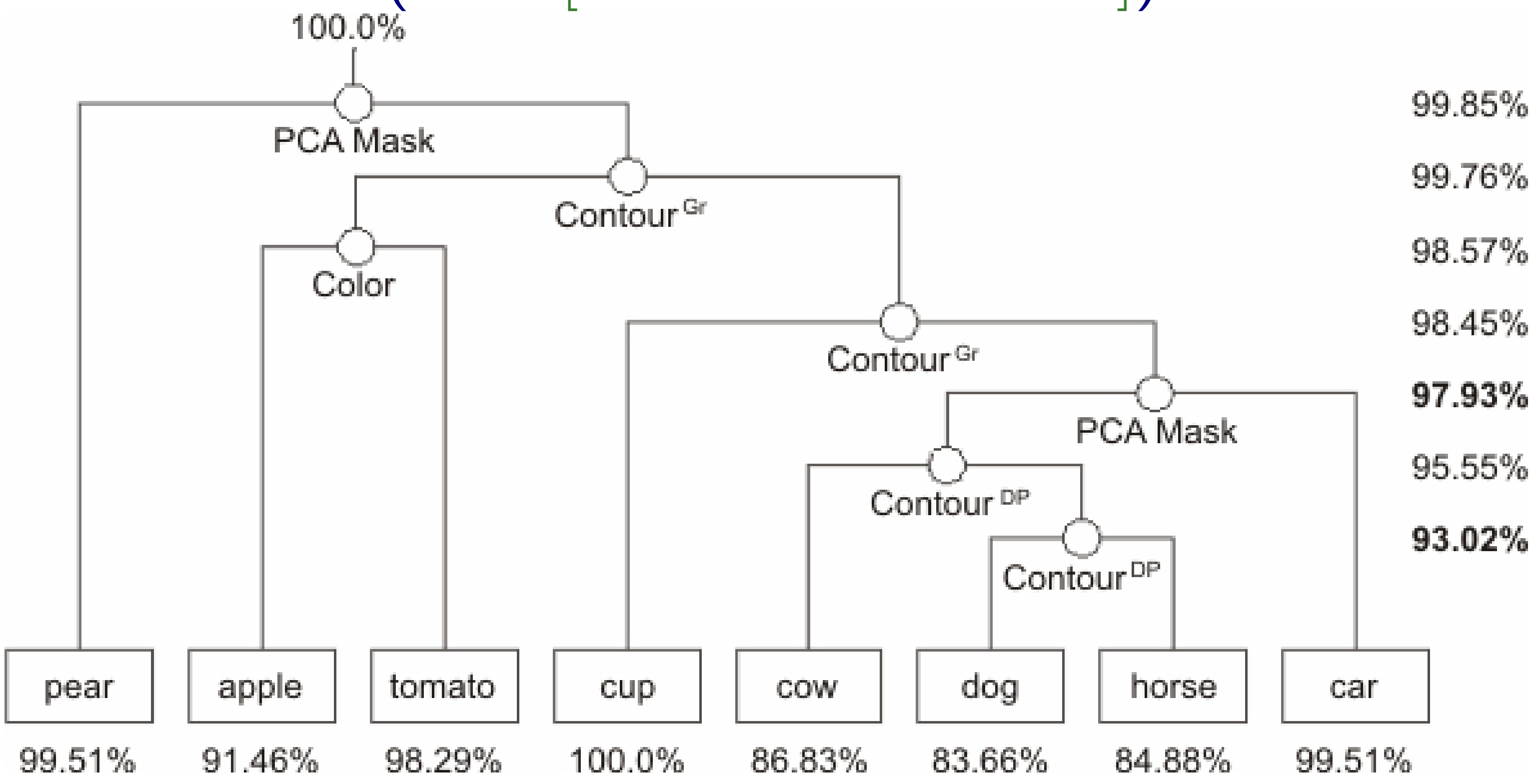
Example: object categorization

- The data set: 8 categories $\times$ 10 objects $\times$ 41 viewpoints

- Cues:
  - Color histograms
  - Texture (histograms of derivatives at multiple scales)
  - Shape - global descriptor (binary mask)
  - Shape - local descriptor (shape context)

Example: object categorization

- Best NN in a single modality: shape (contour) 86.4%
- Decision tree (from [Leibe & Schiele '03]):

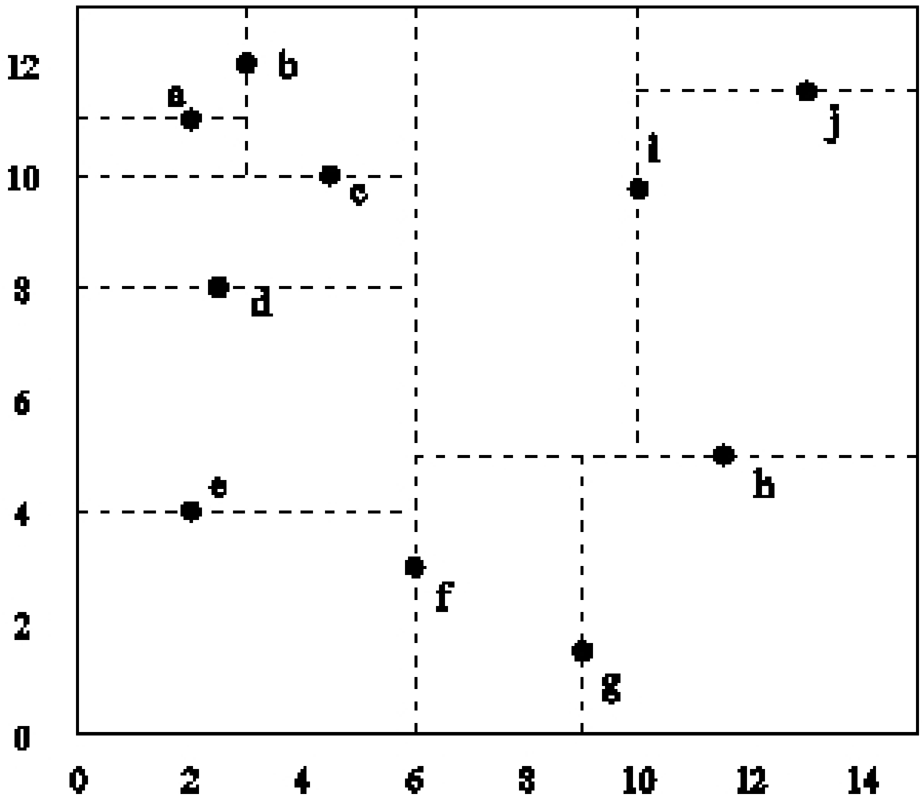


NN: search

- Brute force search for NN: $O(dN)$ (need to compare to all examples).
  - Becomes impractical for large high-dimensional data sets.
- In 1D, 2D tricks exist that allow very efficient search.
- In higher dimension, clever indexing of the data can help...

kd-trees: preprocessing

Recursively partition by the median along max spread.

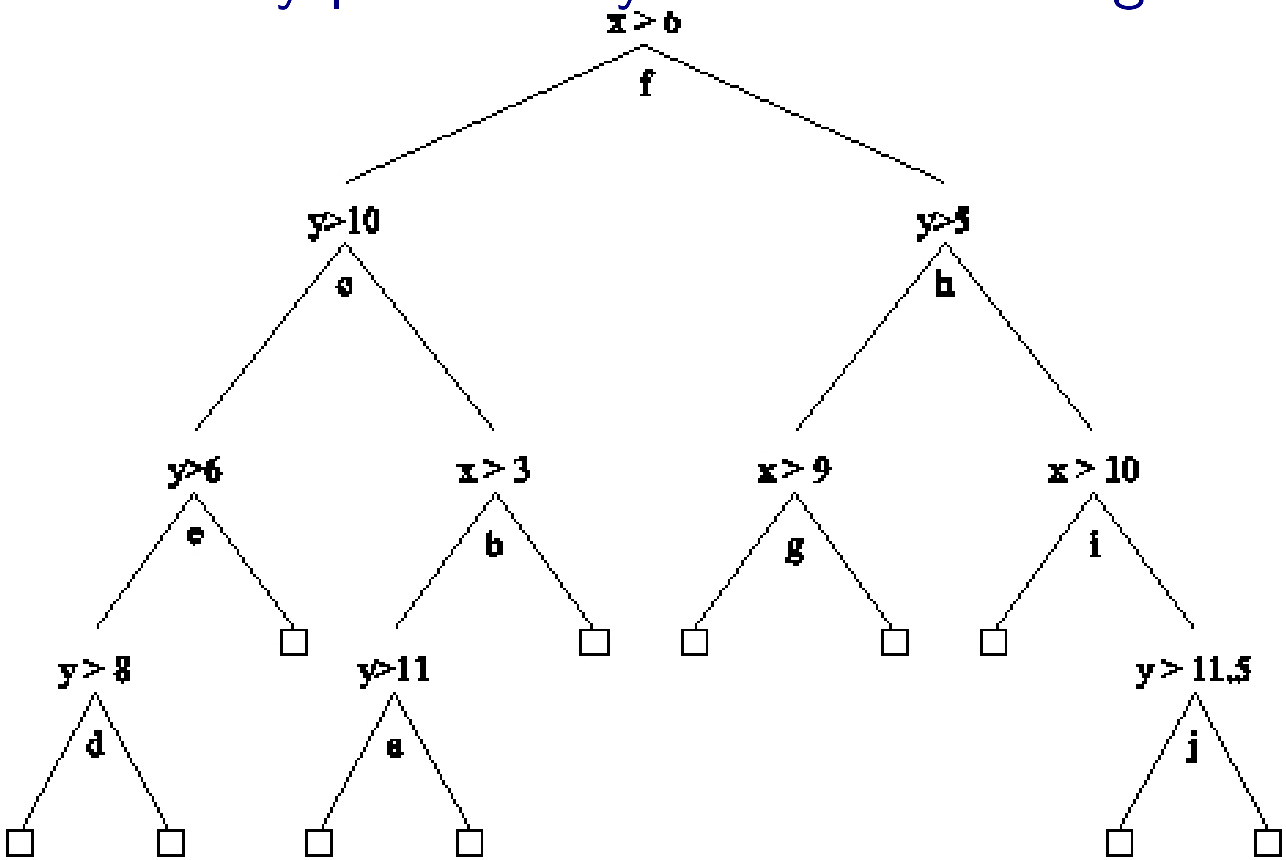


kd-trees: preprocessing

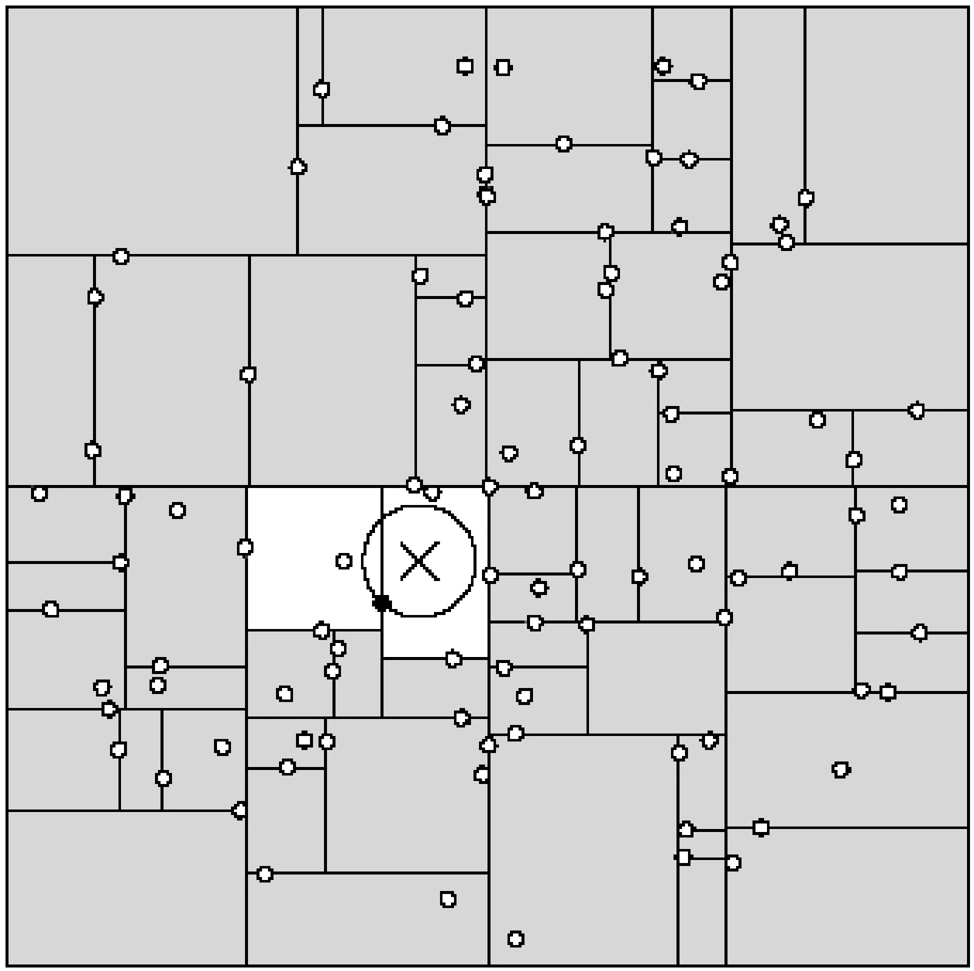
Recursively partition by the median along max spread

kd-trees: preprocessing

Recursively partition by the median along max spread:



kd-trees: search



kd-trees: search

- Find the leaf containing the query point
  - This may rule out much of the data set, since a NN must not be farther than this leaf!

kd-trees: search

- Find the leaf containing the query point
  - This may rule out much of the data set, since a NN must not be farther than this leaf!
- Back up, and explore the nodes which *may* contain the query.

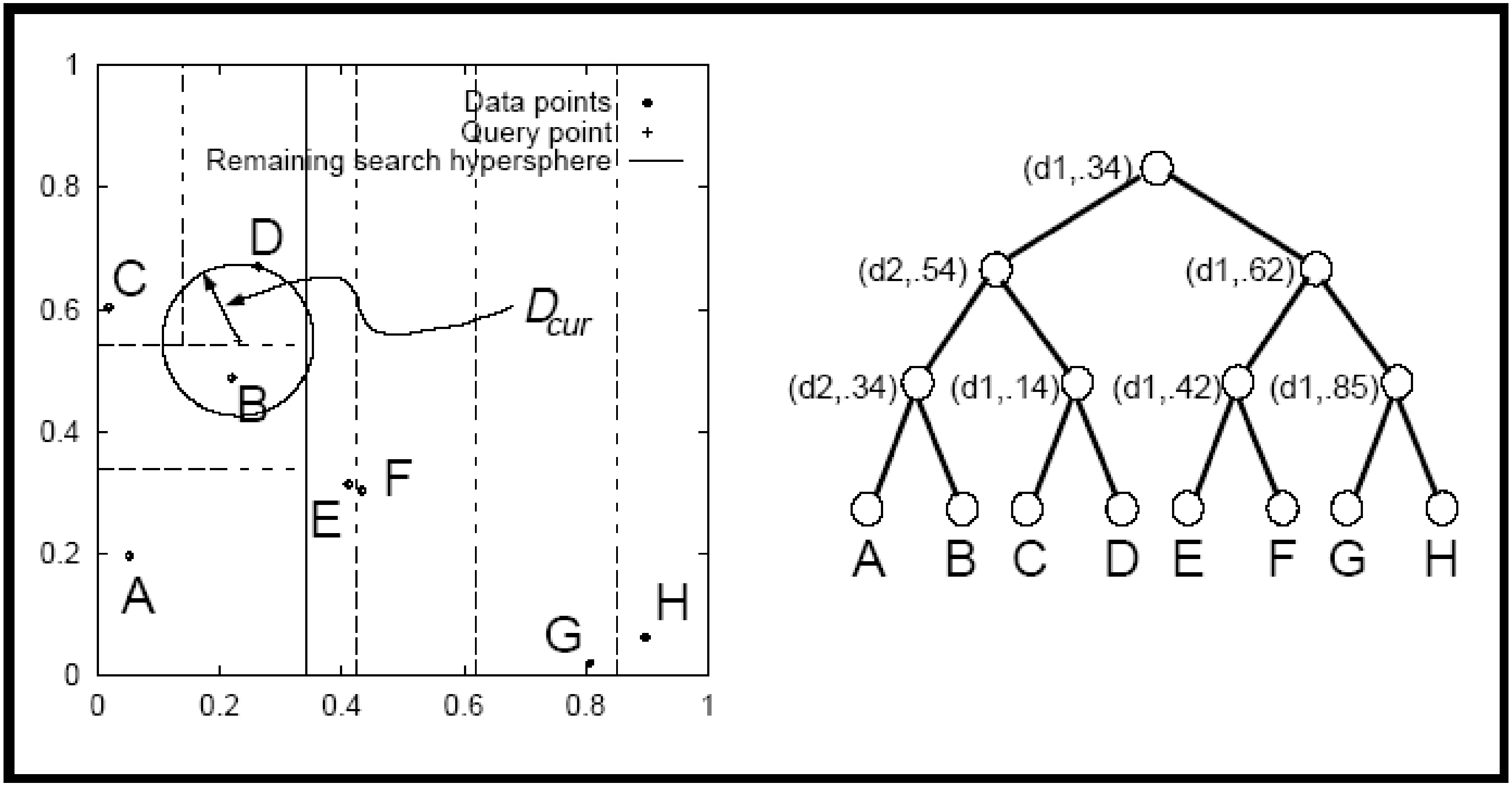
kd-trees: search

- Find the leaf containing the query point
  - This may rule out much of the data set, since a NN must not be farther than this leaf!
- Back up, and explore the nodes which *may* contain the query.
- Very efficient in low dimensions; expected search time $O(\log N)$ (but exponential in d!).
- When $d > 20$, often achieves worst case – almost linear.

BBF: Best Bin First

- An approximation of NN search [Beis & Lowe '99].
- kd-tree: search bins closest *in the tree structure*
- BBF: search bins closest in space (i.e. distance from query to the bin boundary).
- Approximation: only search m candidates, settle for the best among them.
  - Note: must compare to *restricted* search with "standard" kd-tree.

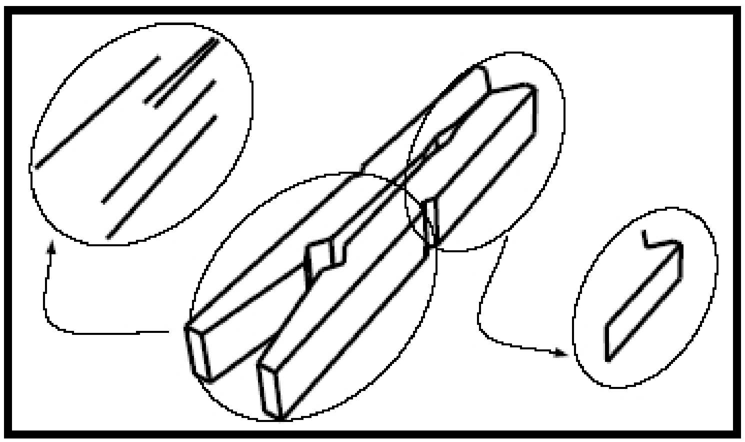
BBF: Best Bin First



[Beis & Lowe '99]

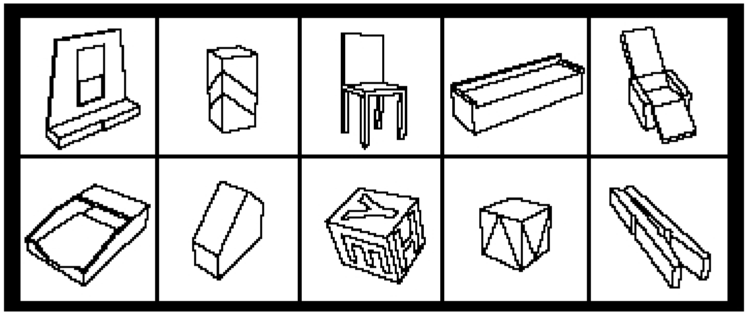
Shape indexing with BBF

- Extract geometric features (3- and 4-segment groups);

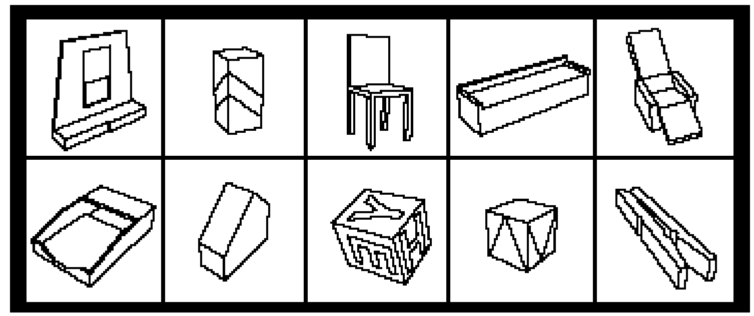


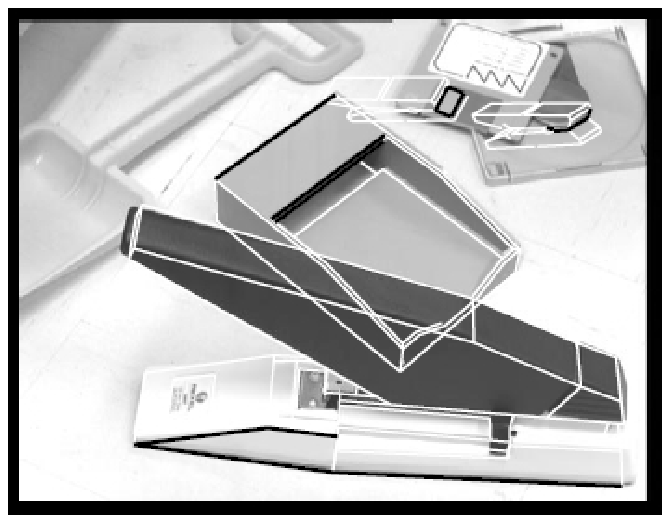
- Index the data set, find k NN for each feature;
- Use k-NN density estimator to rank hypothesis probabilities;
- Verify hypotheses

Shape indexing with BBF

- Database: random viewpoints of 

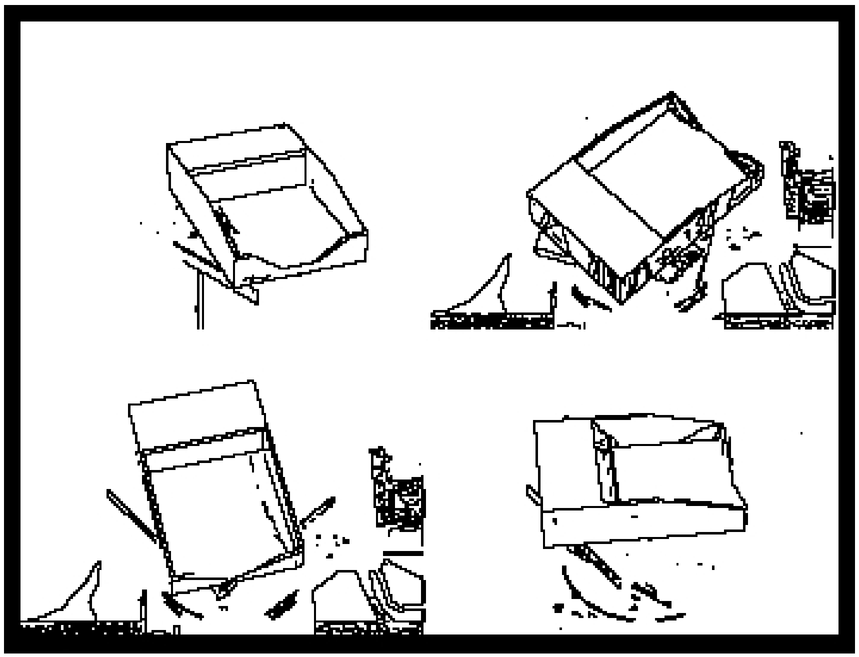
Shape indexing with BBF

- Database: random viewpoints of 

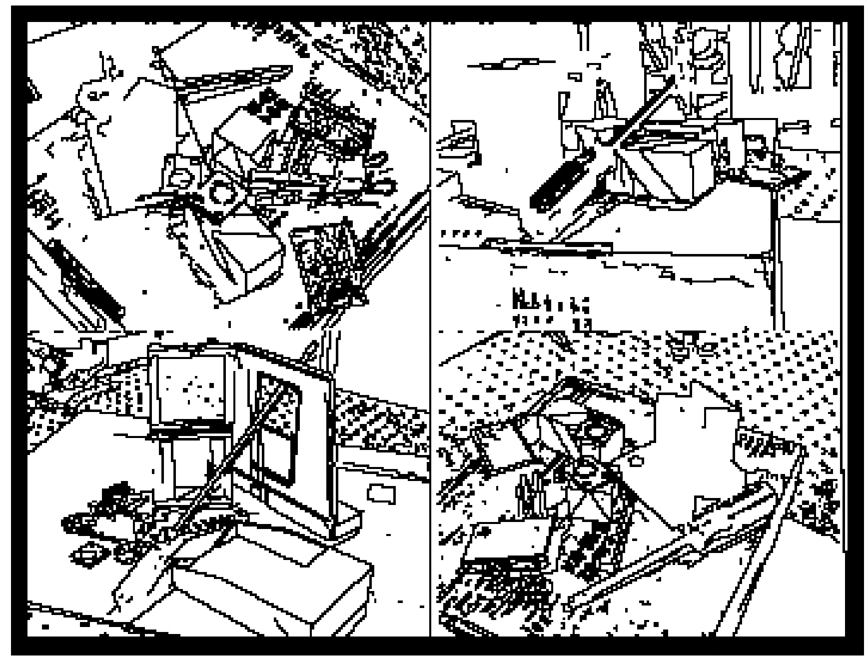


Shape indexing with BBF

"Easy": 99.5%



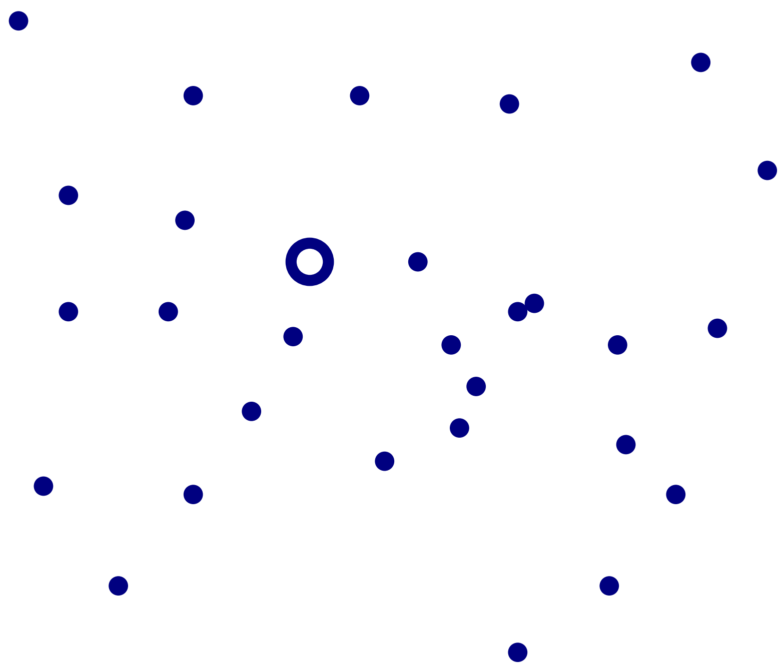
"Difficult": 50%



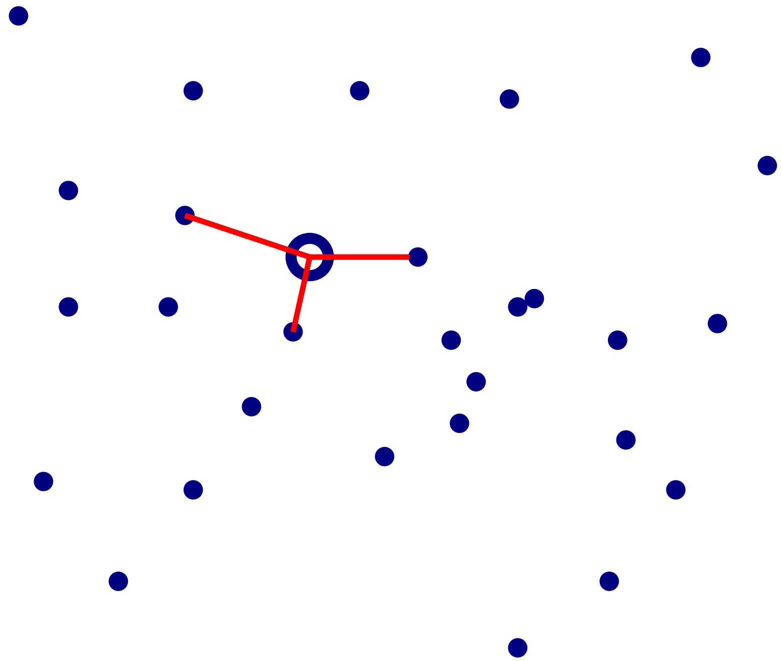
Search in very high dimensions

- BBF provides an answer for dimensions up to 20.
- What happens when d=100? 1,000?
- May relax the search objective and resort to randomization...

Approximate near~~est~~ neighbors

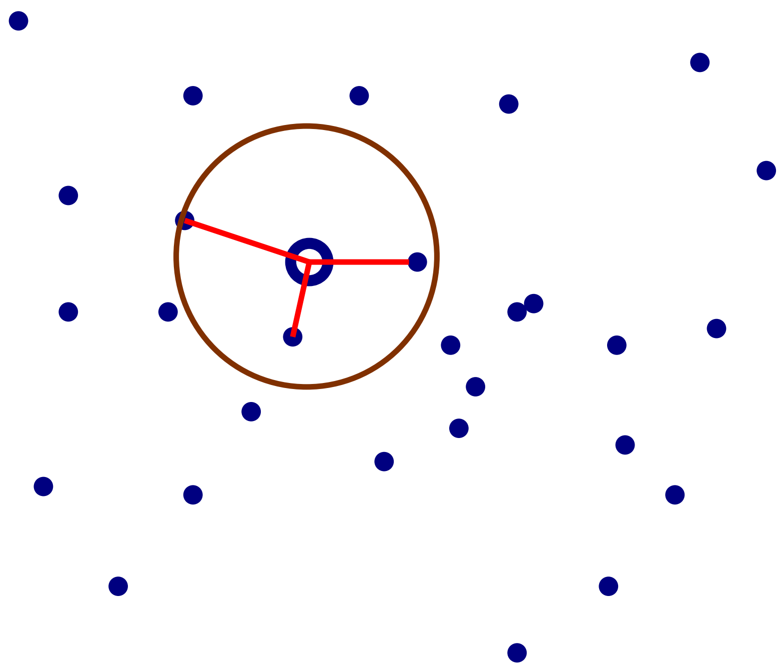


Approximate near~~est~~ neighbors



k nearest neighbors

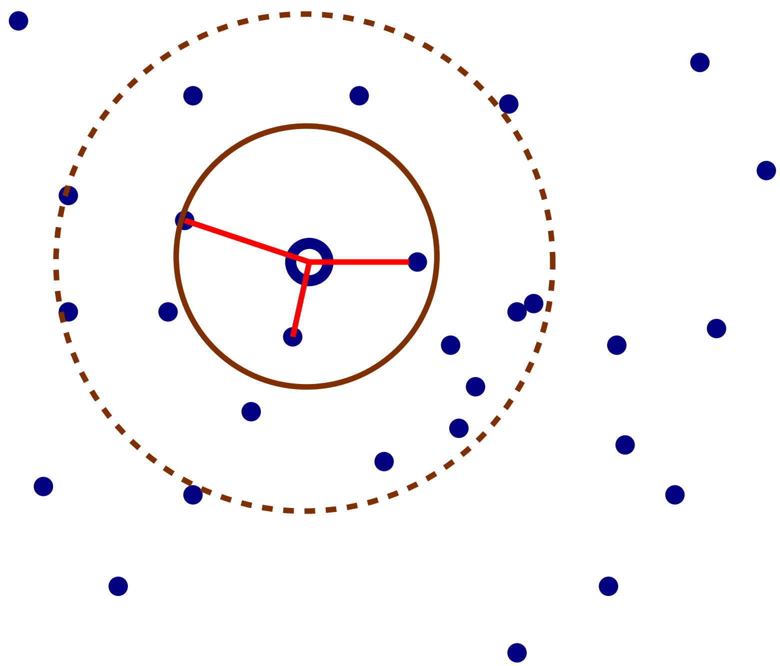
Approximate near~~est~~ neighbors



k nearest neighbors

r-neighbors:
within radius r from $\mathbf{x}_0$

Approximate near~~est~~ neighbors



k nearest neighbors

r-neighbors:
within radius r from $\mathbf{x}_0$

(ϵ, r)-neighbors:
within radius $(1+\epsilon)r$

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:
  - Query time $O\left(dn^{1/(1+\epsilon)}\right)$

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:
  - Query time $O\left(dn^{1/(1+\epsilon)}\right)$
  - Storage $O\left(dn + n^{1+1/(1+\epsilon)}\right)$

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:
  - Query time $O\left(dn^{1/(1+\epsilon)}\right)$
  - Storage $O\left(dn + n^{1+1/(1+\epsilon)}\right)$
- Practical meaning, with 10^6 examples $\times 10000$ features, for $\epsilon = 1$:

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:
  - Query time $O\left(dn^{1/(1+\epsilon)}\right)$
  - Storage $O\left(dn + n^{1+1/(1+\epsilon)}\right)$
- Practical meaning, with 10^6 examples $\times 10000$ features, for $\epsilon = 1$:
  - Assume each feature is a `float` (4×10^{10} data bytes).

LSH: Fast search for (ϵ, r)-neighbors

- [Gionis *et al* '99, Indyk&Motwani '98]
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$:
  - Query time $O\left(dn^{1/(1+\epsilon)}\right)$
  - Storage $O\left(dn + n^{1+1/(1+\epsilon)}\right)$
- Practical meaning, with 10^6 examples $\times 10000$ features, for $\epsilon = 1$:
  - Assume each feature is a `float` $(4 \times 10^{10}$ data bytes).
  - The algorithm requires 4.1×10^{10} bytes storage,
  - Query requires about $O(10^7)$ byte operations,
  - Compared to $O(10^{10})$ for exhaustive search.

LSH: intuition

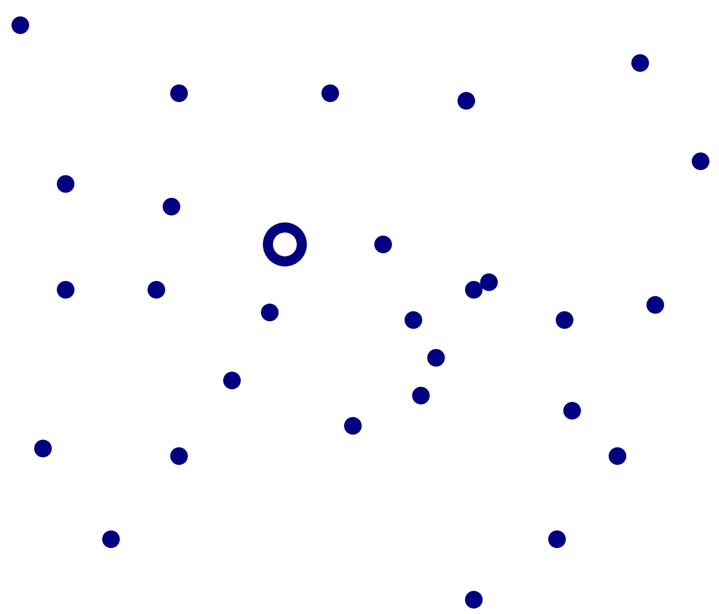
- Preprocessing: index the data by l hash tables

LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points

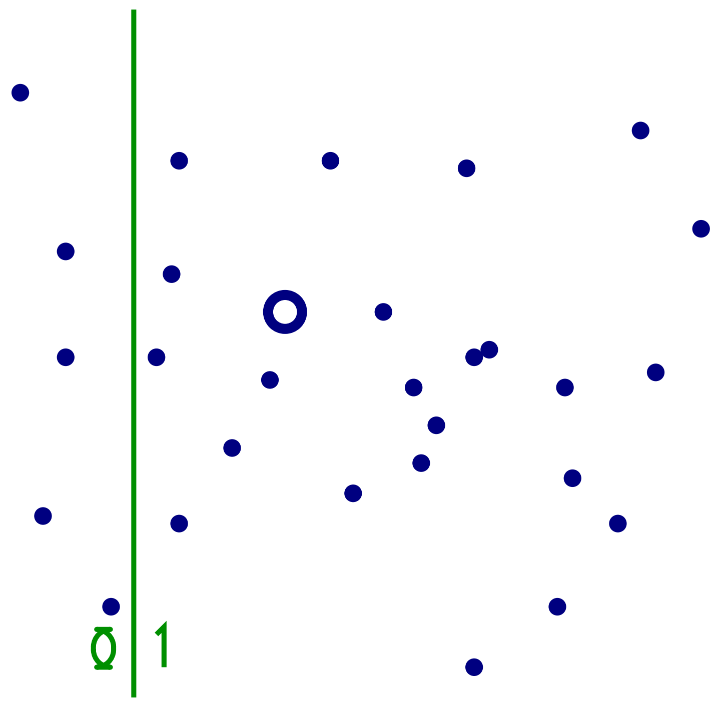
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



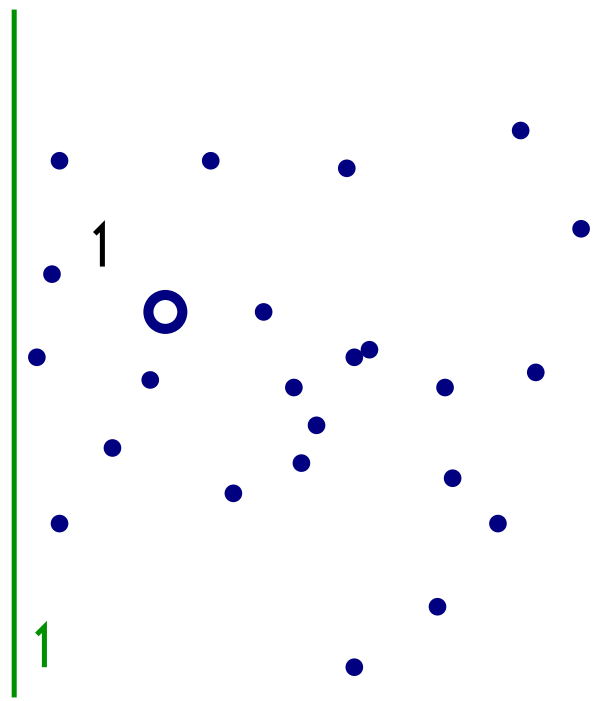
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



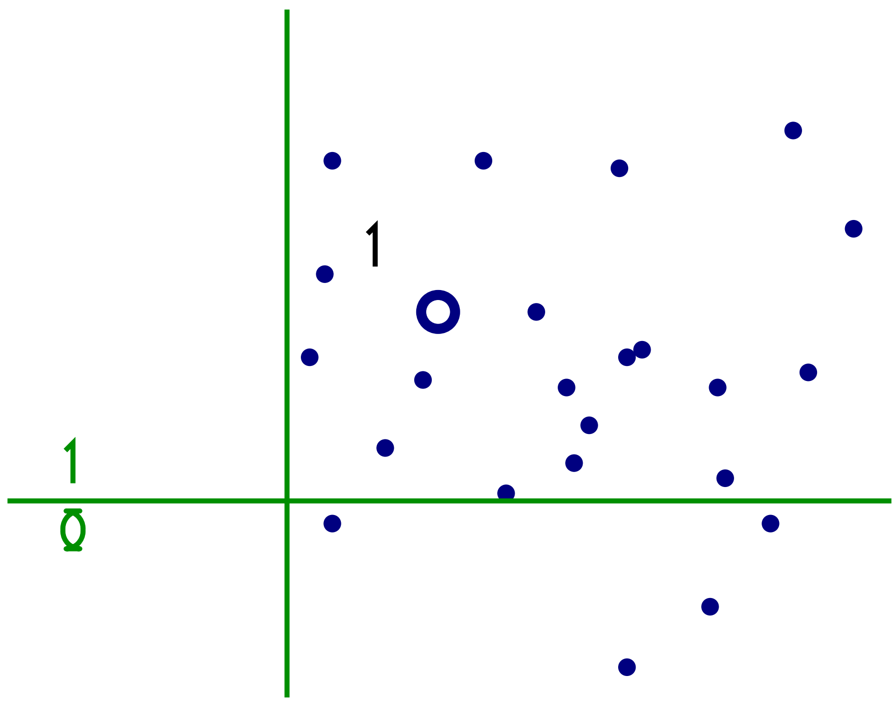
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



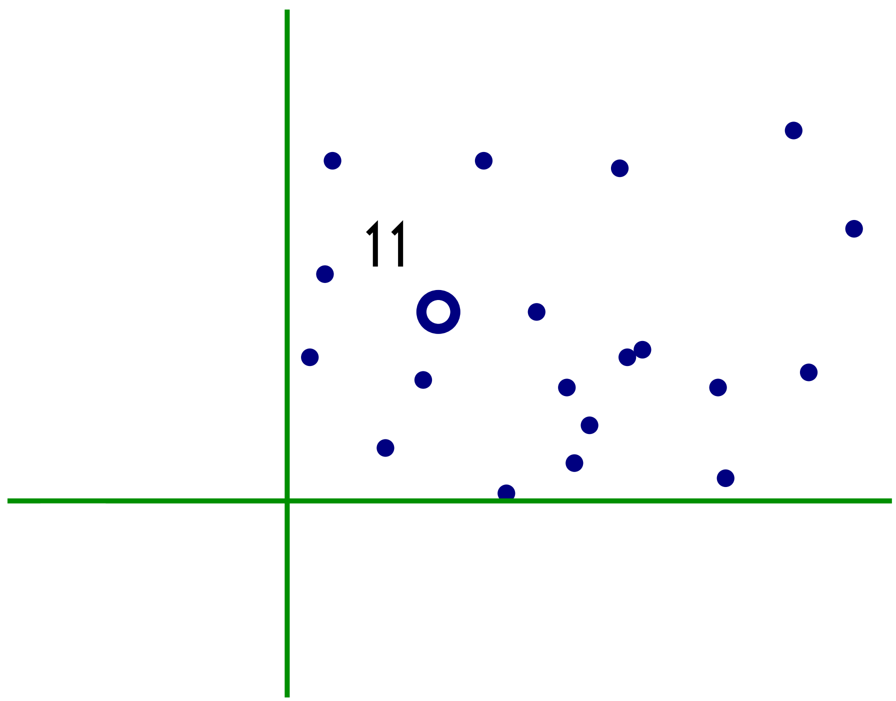
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



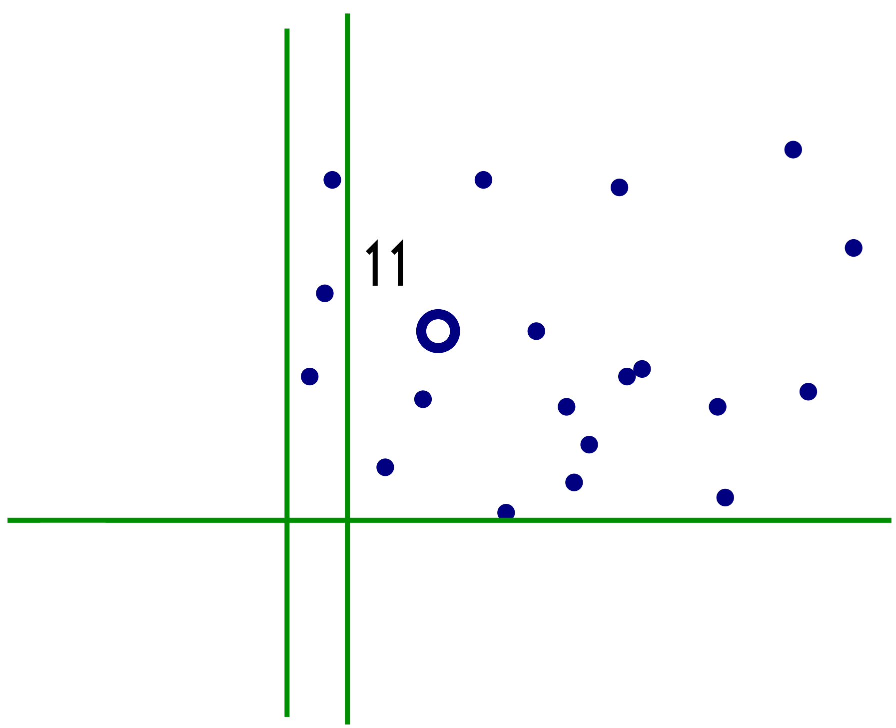
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



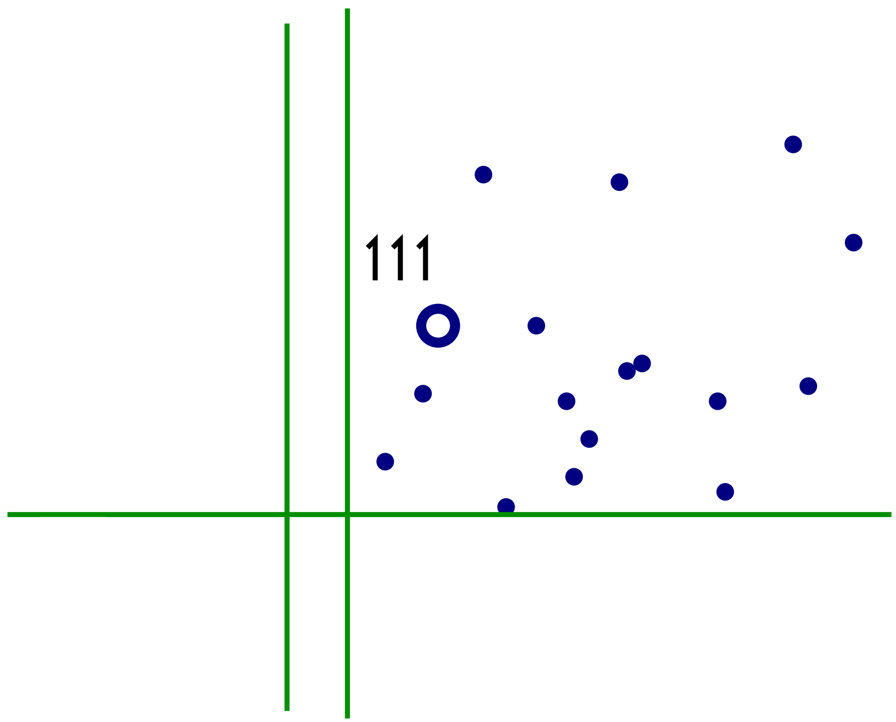
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



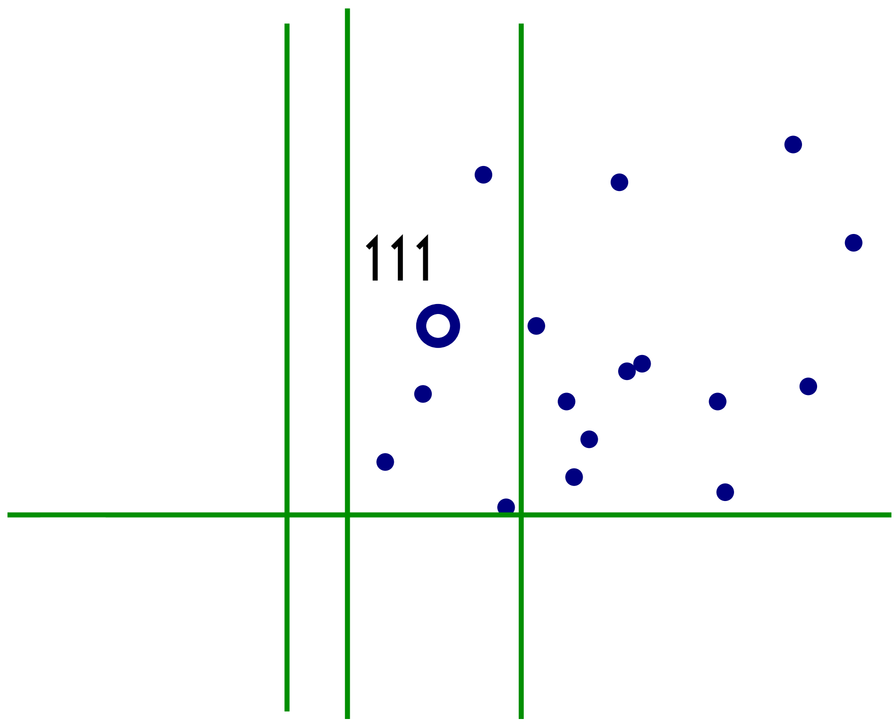
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



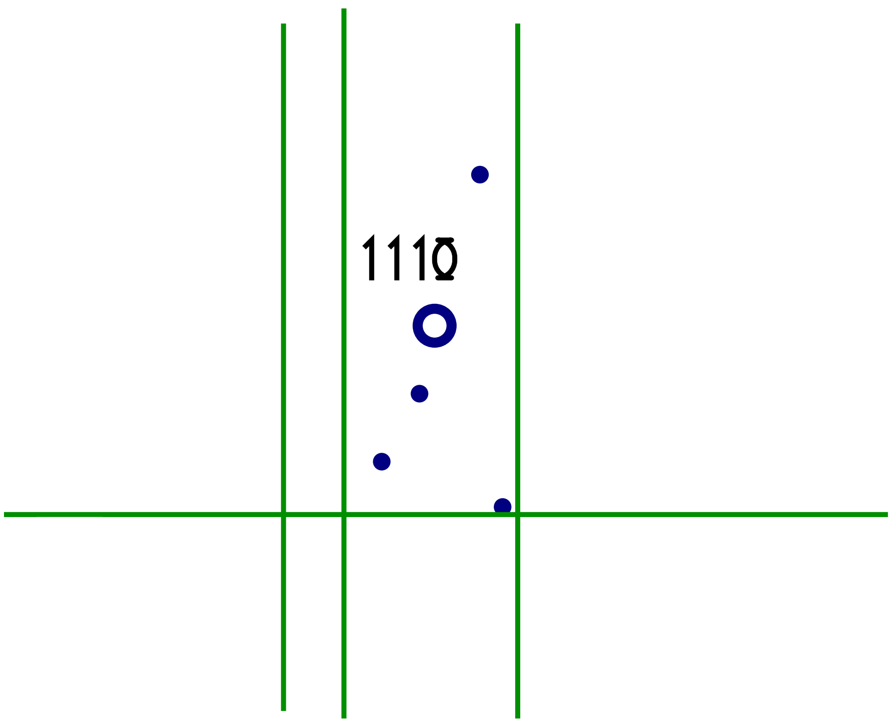
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



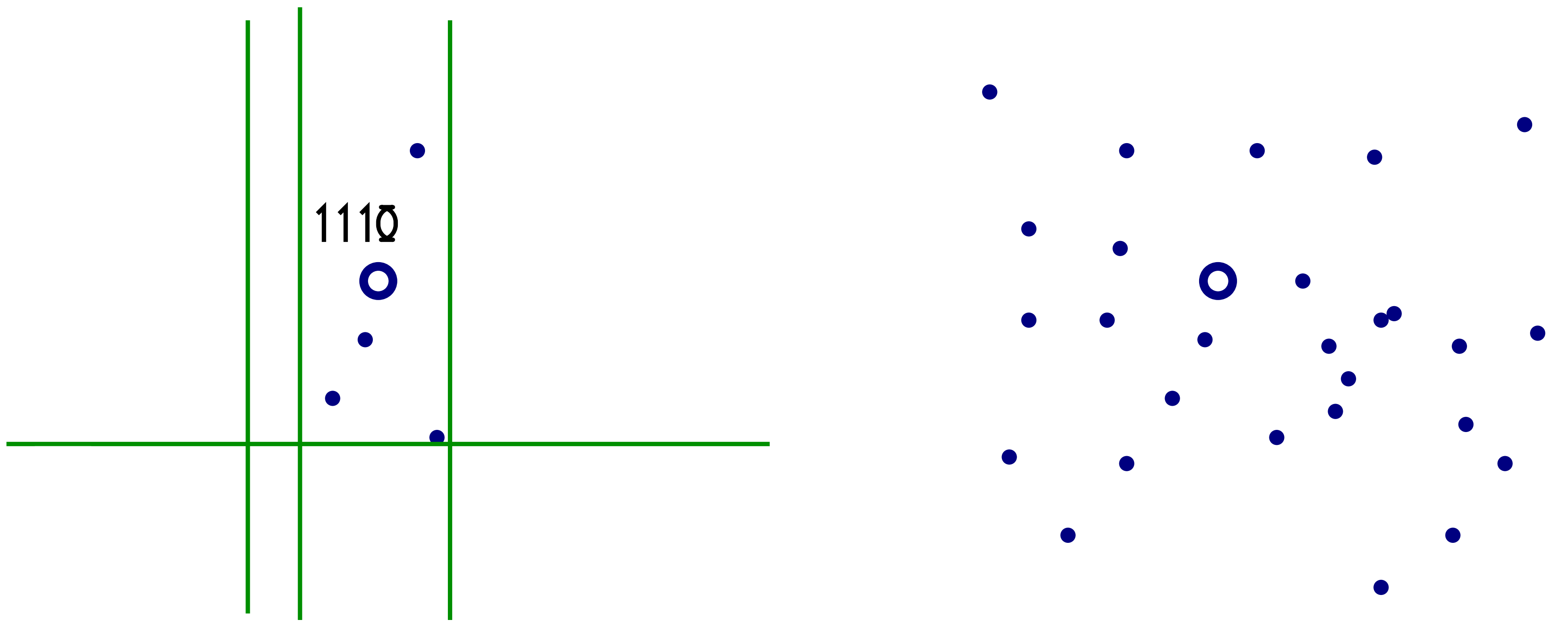
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



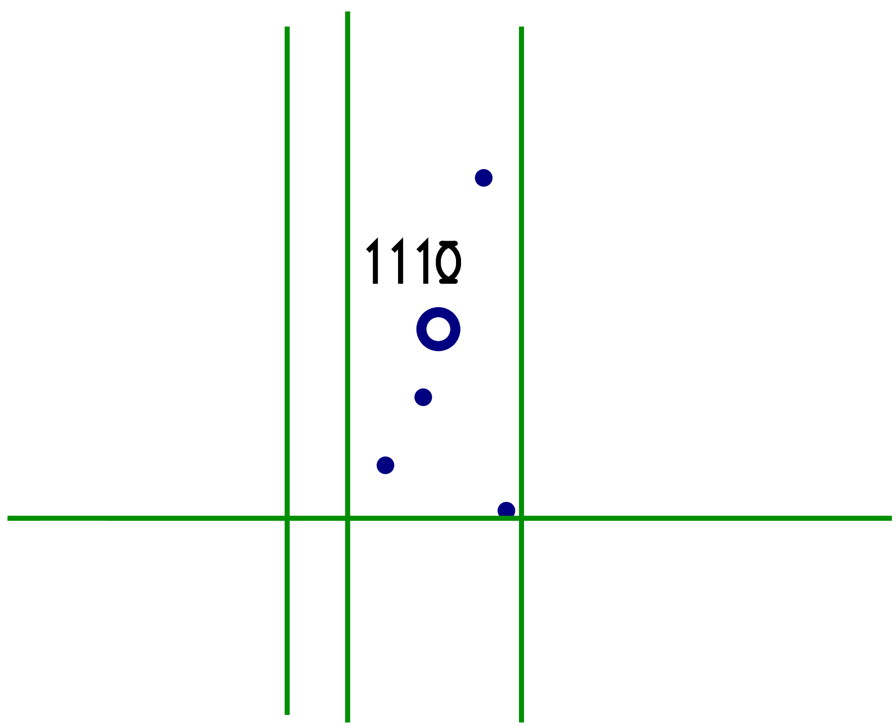
LSH: intuition

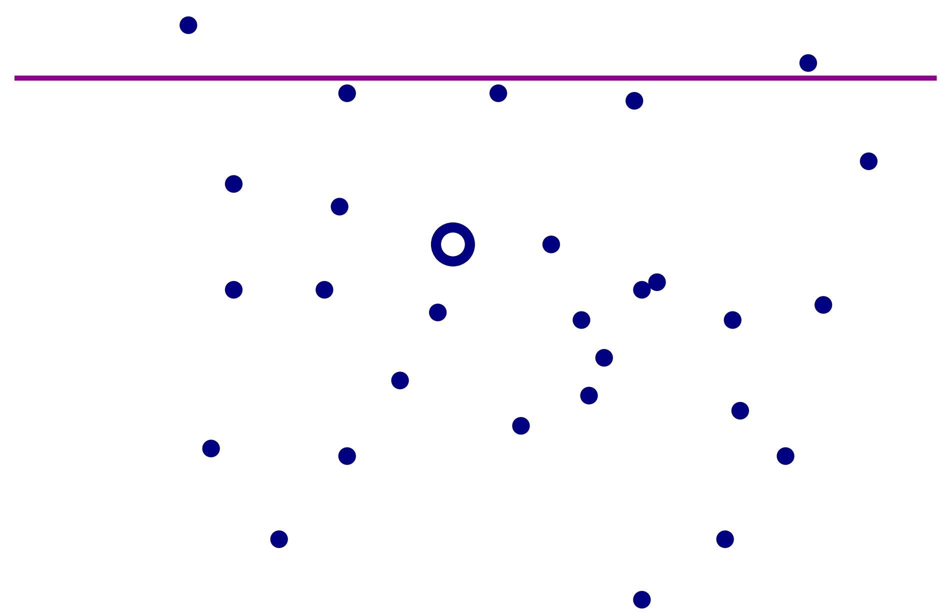
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



LSH: intuition

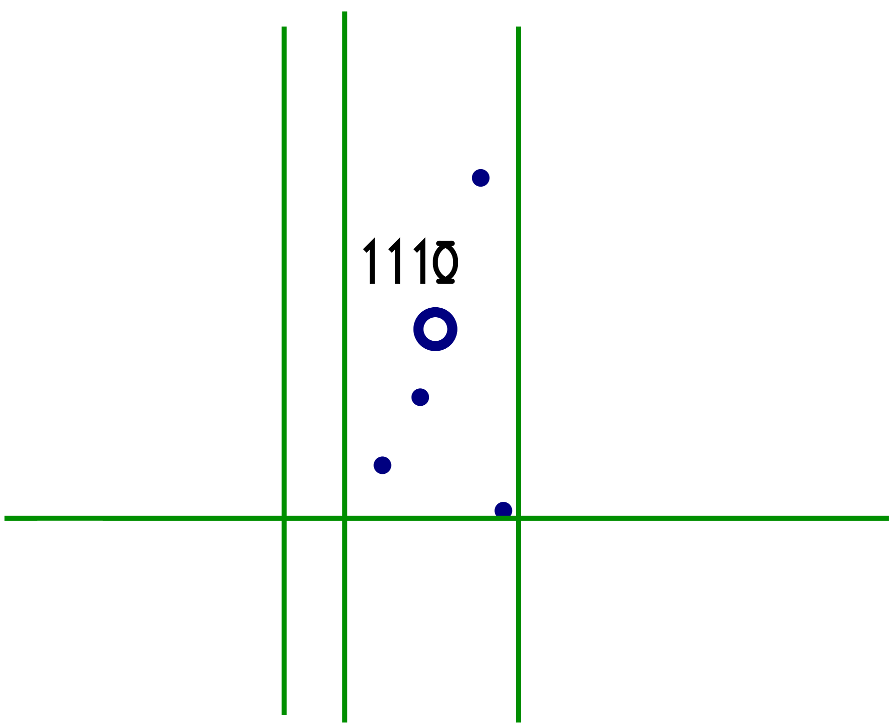
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points

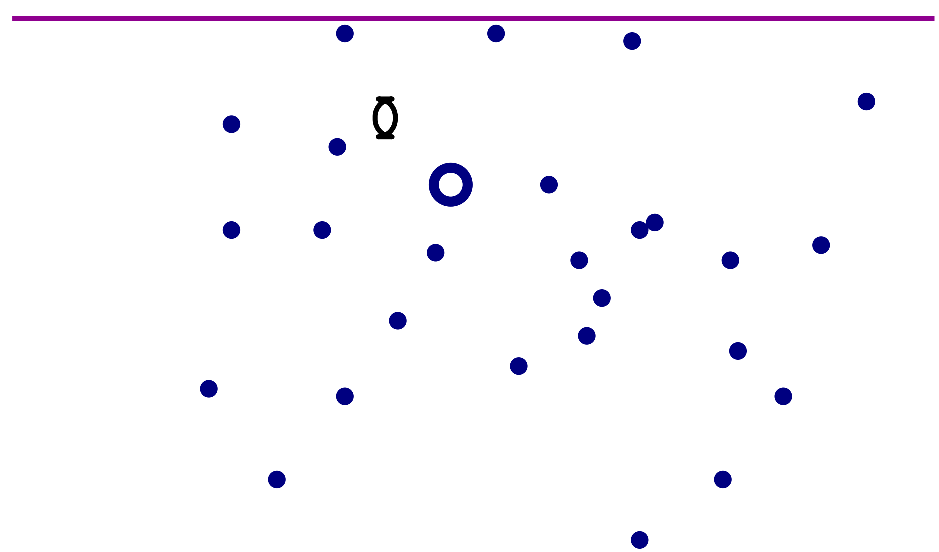




LSH: intuition

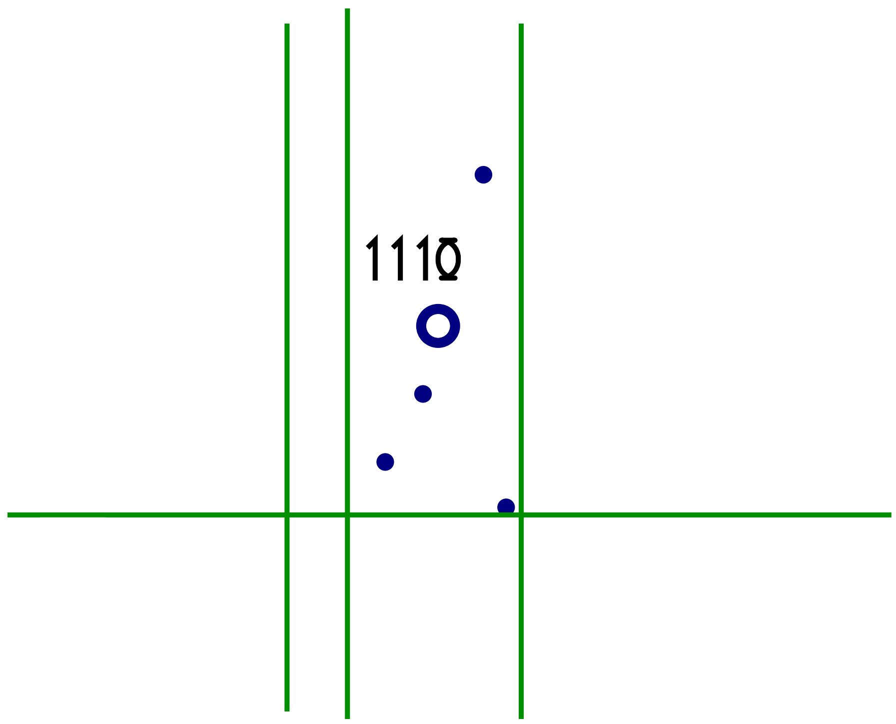
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points

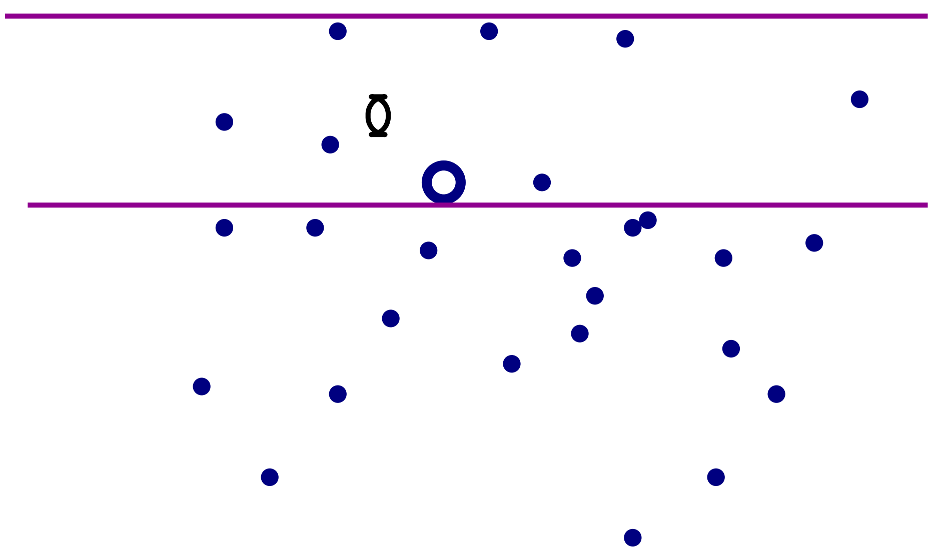




LSH: intuition

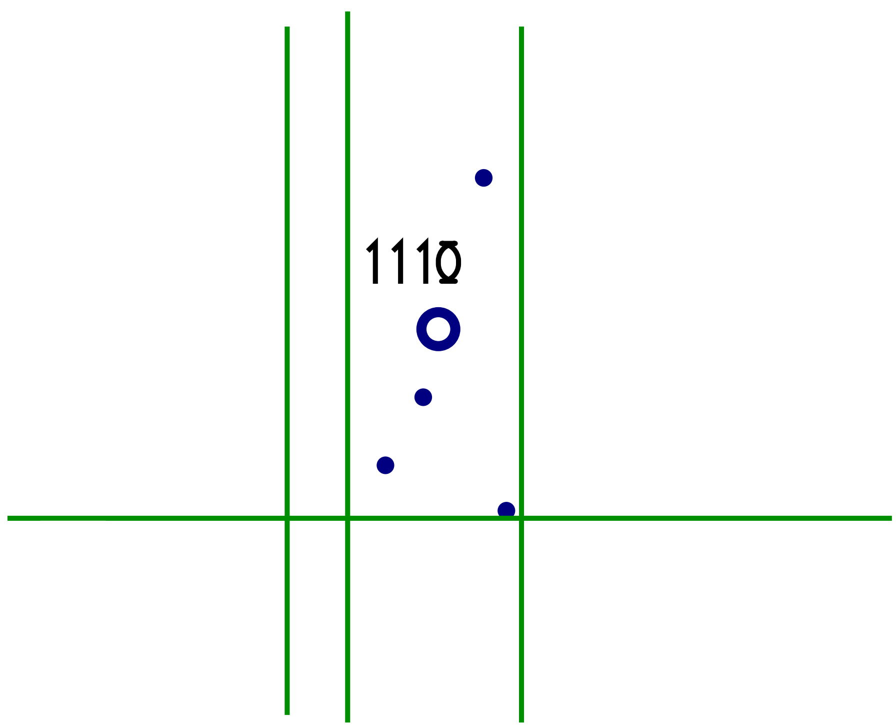
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points

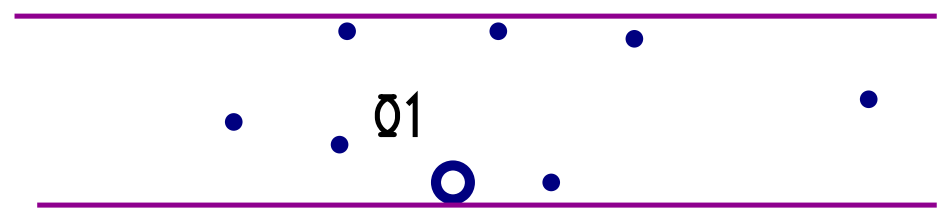




LSH: intuition

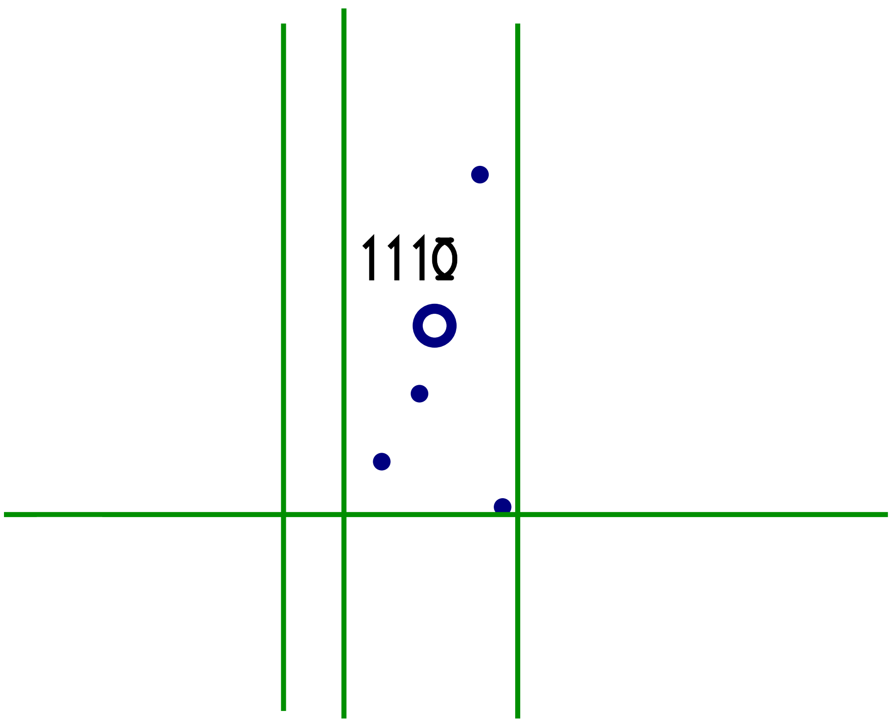
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points

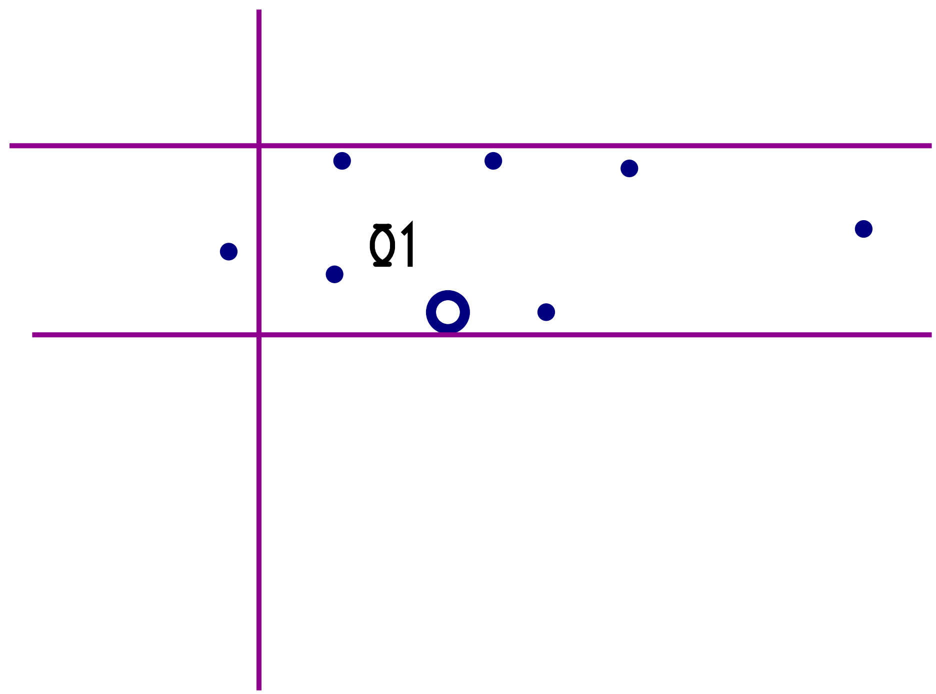




LSH: intuition

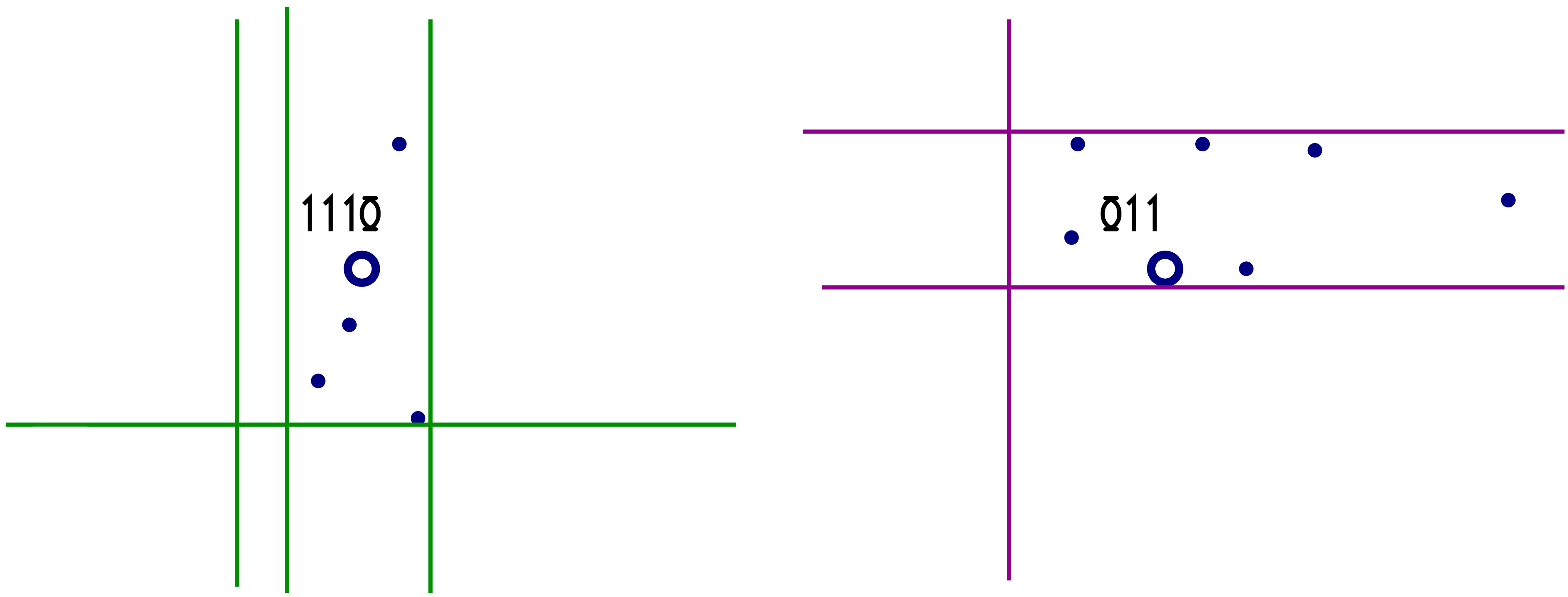
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points





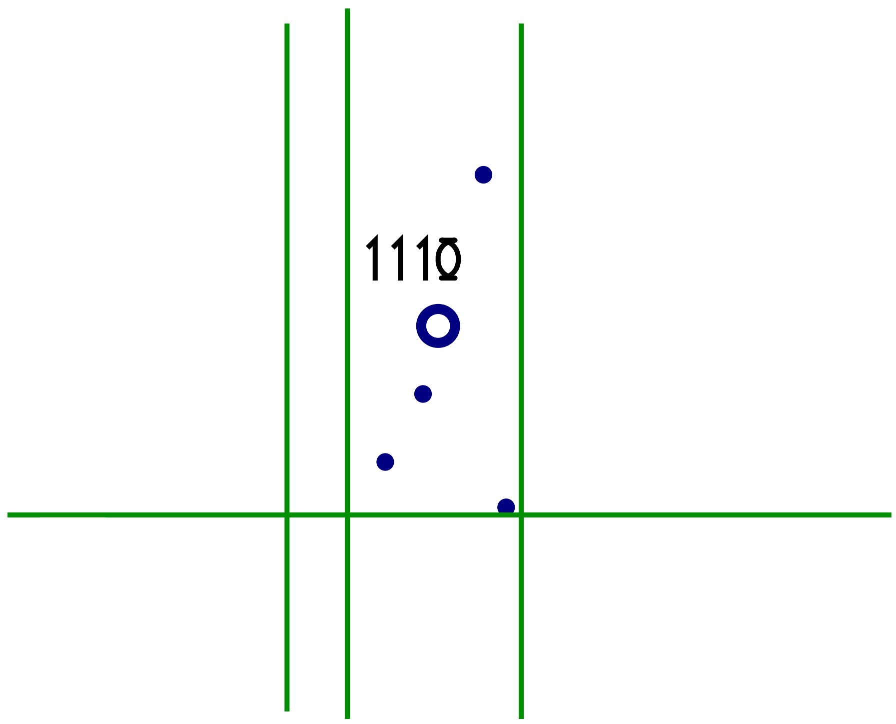
LSH: intuition

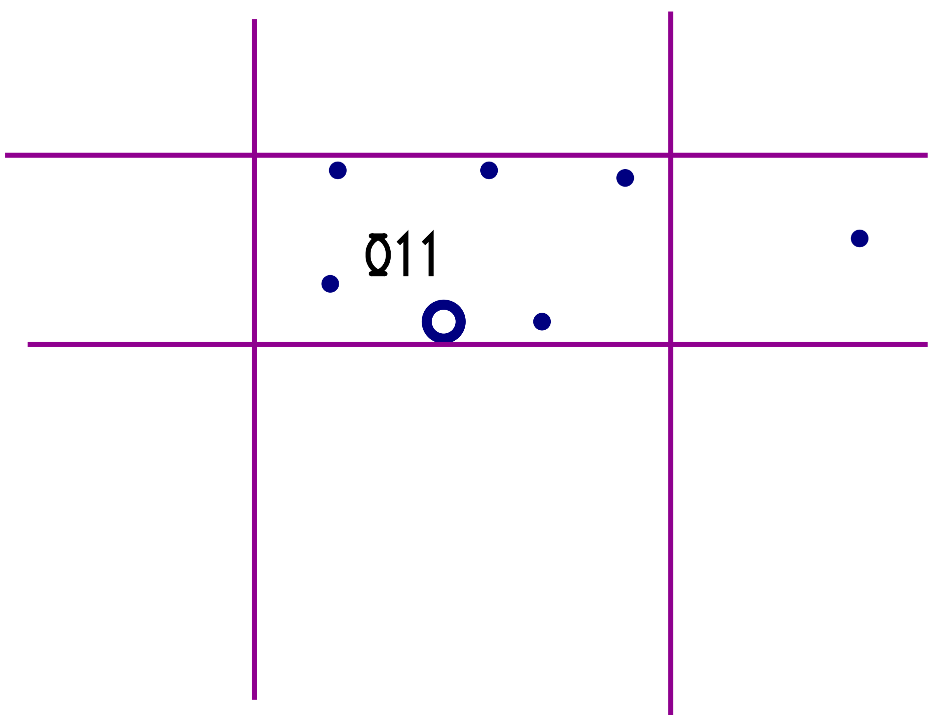
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



LSH: intuition

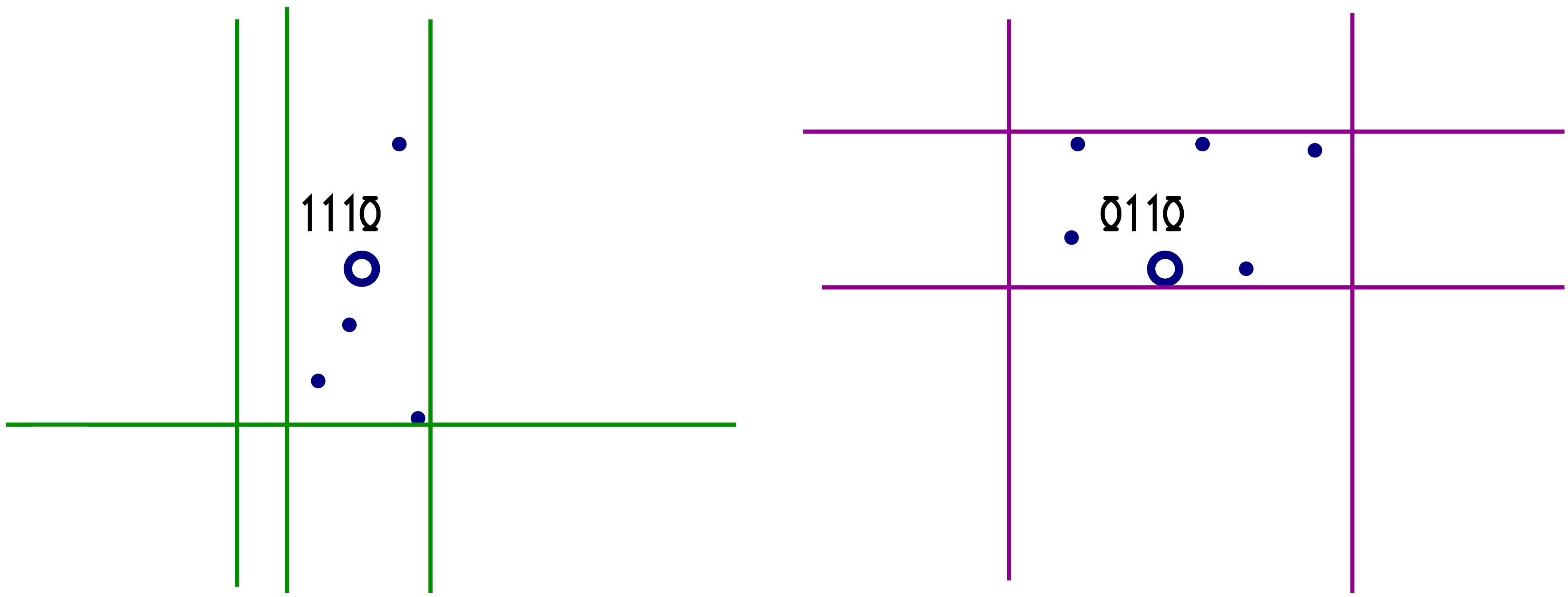
- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points





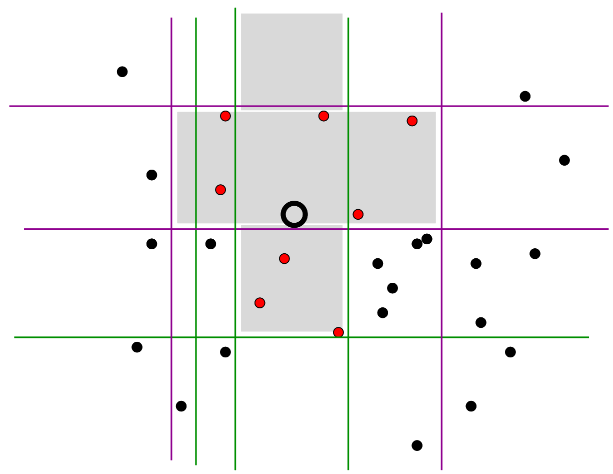
LSH: intuition

- Preprocessing: index the data by l hash tables
- Hash functions are unlikely to separate close points



LSH: intuition

- Exhaustively search the union of the matching buckets.



- More bits per key = smaller buckets, less to search.
- More hash tables = more to search but unlikely to miss neighbors.

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \;\rightarrow\; \{-1, +1\}$$

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \rightarrow \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \ \rightarrow \ \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \ \geq \ p_1$

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \rightarrow \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \geq p_1$,
    Probability of "good" collision

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \;\rightarrow\; \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \;\geq\; p_1$
  - if $d(\mathbf{x}_0, \mathbf{x}) \geq R$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \;\leq\; p_2$.

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \;\rightarrow\; \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \;\geq\; p_1$
  - if $d(\mathbf{x}_0, \mathbf{x}) \geq R$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \;\leq\; p_2$.
    Probability of "bad" collision

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \rightarrow \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr(h(\mathbf{x}_0) = h(\mathbf{x})) \geq p_1$
  - if $d(\mathbf{x}_0, \mathbf{x}) \geq R$ then $\Pr(h(\mathbf{x}_0) = h(\mathbf{x})) \leq p_2$.

- Useful if $p_1 > p_2$, $r < R$.

Locality-sensitive hash functions

- Let $\mathcal{H}$ be a family of bit-valued hash functions

$$h : \mathcal{X} \rightarrow \{-1, +1\}$$

- This is a (r, R, p_1, p_2)-sensitive hash family, if for a random $h \in \mathcal{H}$
  - if $d(\mathbf{x}_0, \mathbf{x}) < r$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \geq p_1$
  - if $d(\mathbf{x}_0, \mathbf{x}) \geq R$ then $\Pr\left(h(\mathbf{x}_0) = h(\mathbf{x})\right) \leq p_2$.

- Useful if $p_1 > p_2$, $r < R$.

- For $\mathcal{X} = \mathbb{R}^n$ and $d \equiv L_1$: axis-parallel decision stumps are locality-sensitive.

LSH: review of the algorithm

- Choose a locality-sensitive family $\mathcal{H}$ of hash functions.
- Build l indepedent hash tables:
  - Construct a k-bit hash function $g(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_k(\mathbf{x}))$ with randomly selected $h_j \in \mathcal{H}$.
  - Store each example $\mathbf{x}_i$ in the bucket $g(\mathbf{x}_j)$.
- On query $\mathbf{x}_0$: exhaustively search the union of $g_1(\mathbf{x}_0), \ldots, g_l(\mathbf{x}_0)$ for (ϵ, r)-neighbors of $\mathbf{x}_0$.

LSH: properties

- l hash tables with k-bit hash functions.
- Hash function family: probability of "good" collision $\geq p_1$, "bad" collision $\leq p_2$.

LSH: properties

- l hash tables with k-bit hash functions.
- Hash function family: probability of "good" collision $\geq p_1$, "bad" collision $\leq p_2$.
- Probability of collision in *one* table: at least p_1^k for similar points, at most p_2^k for dissimilar ones.

LSH: properties

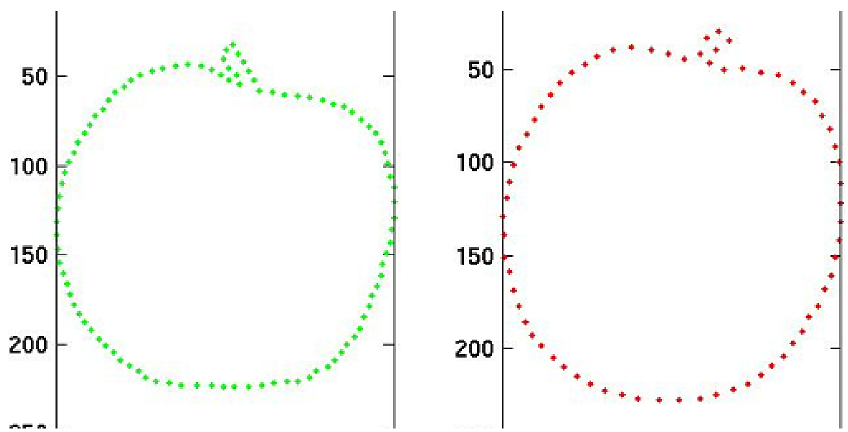
- l hash tables with k-bit hash functions.
- Hash function family: probability of "good" collision $\geq p_1$, "bad" collision $\leq p_2$.
- Probability of collision in *one* table: at <u>least</u> p_1^k for similar points, at <u>most</u> p_2^k for dissimilar ones.
- The probability of finding at least one similar point $\geq \ 1 - (1 - p_1^k)^l$
- The expected number of points to search $\leq \ lNp_2^k$

LSH: properties

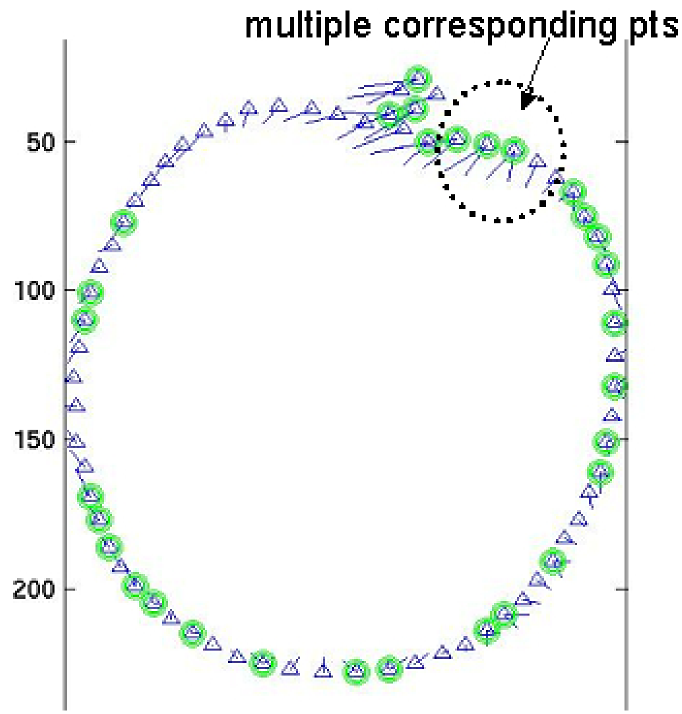
- l hash tables with k-bit hash functions.
- Hash function family: probability of "good" collision $\geq p_1$, "bad" collision $\leq p_2$.
- Probability of collision in *one* table: at least p_1^k for similar points, at most p_2^k for dissimilar ones.
- The probability of finding at least one similar point $\geq \ 1-(1-p_1^k)^l$
- The expected number of points to search $\leq \ lNp_2^k$
- Parameters k, l can be set based on p_1, p_2.

NN in non-Euclidean spaces

- The distance may not be a norm.
- *Embedding* into a norm (possibly with *distortion*).
- Example: Earth-Mover's Distance $$\mathrm{EMD}(\mathrm{P},\mathrm{Q}) = \min_{F} \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}\, GD(p_i, q_j)}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}$$

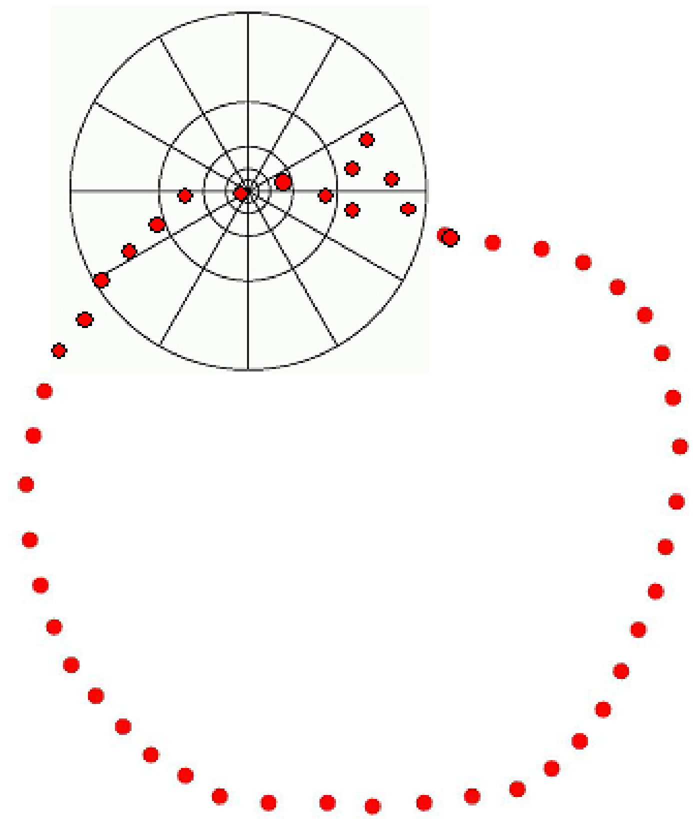


EMD



EMD flow

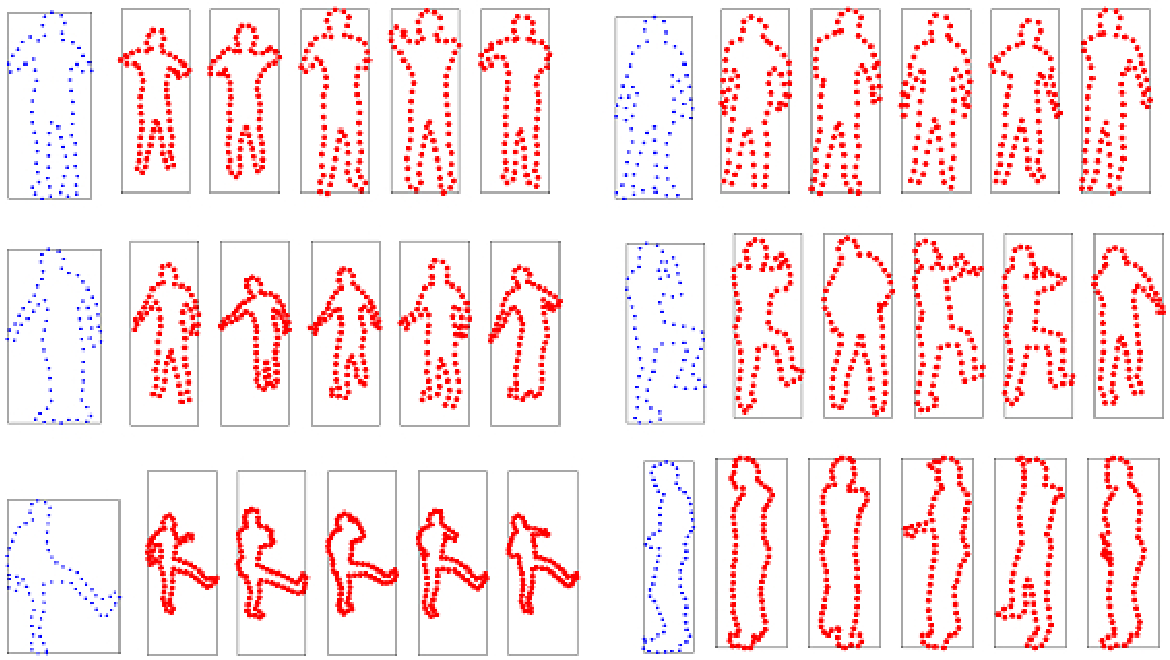
Shape contexts [Belongie *et al*]



A *local shape descriptor*: histogram in each contour point describes the shape (other contour points) in the vicinity.

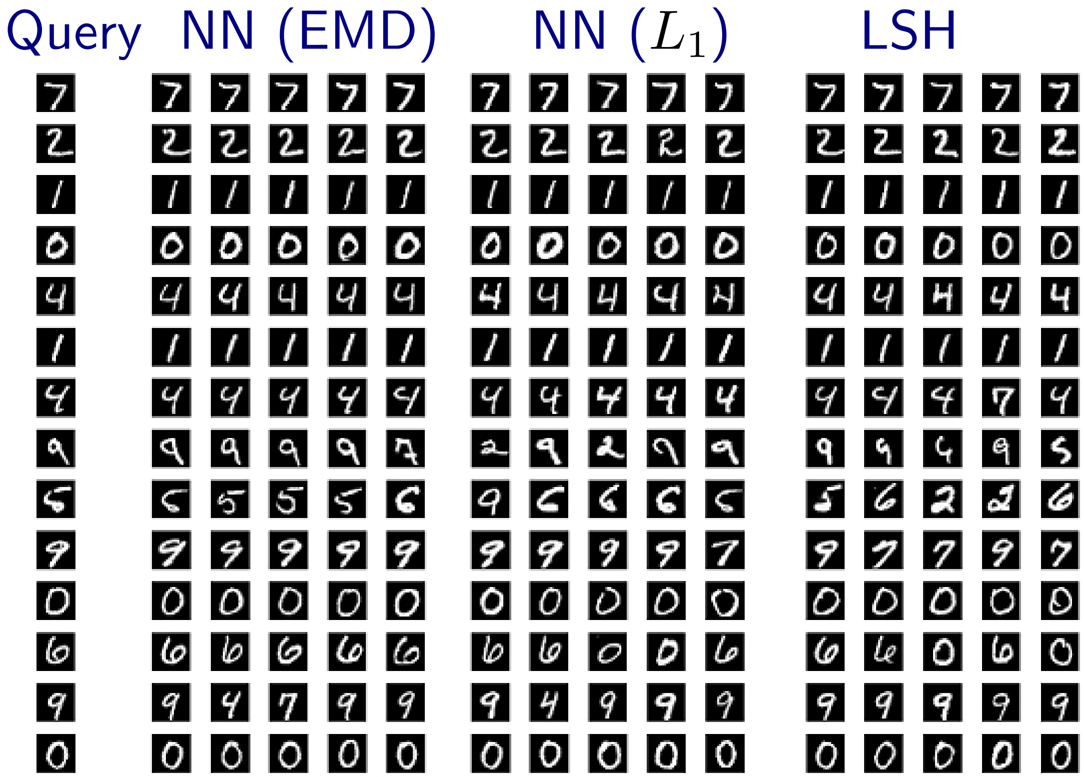
Shape retrieval with LSH

[Grauman & Darrell '04]: tests on body silhouettes (PCA on shape contexts $\longrightarrow$ EMD $\longrightarrow$ embedding in L_1)



Shape retrieval with LSH

[Grauman & Darrell '04]: tests on hand-written digits.

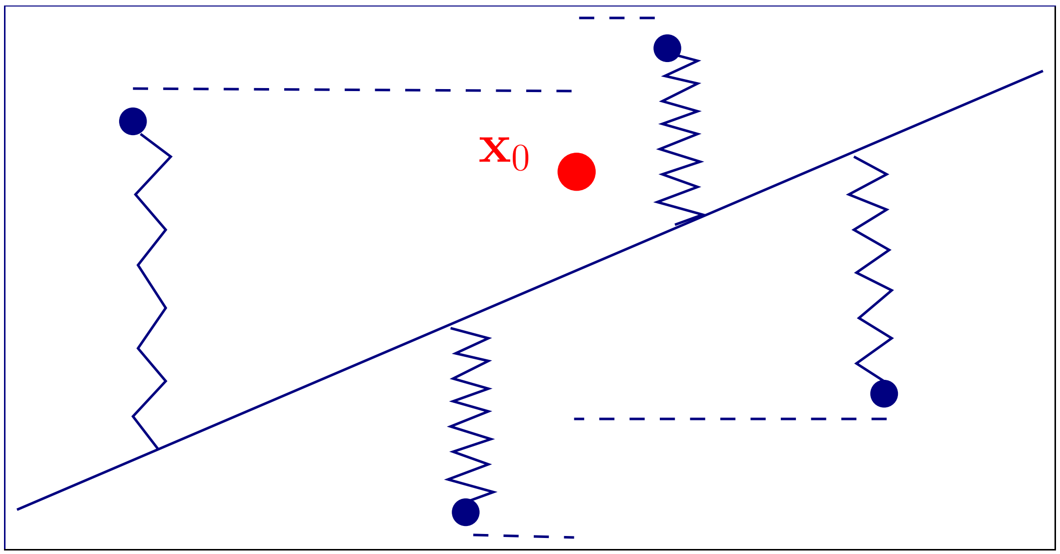


Example-based estimation (regression)

- Labels $\theta = f(\mathbf{x})$
- Simple k-NN method: $\hat{f}(\mathbf{x}_0) = \frac{1}{k}\sum_{\mathbf{x}_j \in \text{ NN of } \mathbf{x}_0} f(\mathbf{x}_j)$
- Problem: if neighborhood is sparse relative to f,
- Possible remedy: pay more attention to the examples closer to $\mathbf{x}_0$.

LWR - intuition

- Excellent introduction in [Atkeson,Moore,Schaal].



- The relative strengths of the springs depend on the kernel.

LWR

- Fit the model $\theta = g\left(\mathbf{x}; \beta\right)$ to observations within a small neighborhood of the query point $\mathbf{x}_0$:

$$\beta^*(\mathbf{x}_0) = \underset{\beta}{\operatorname{argmin}} \sum_i L\left(g(\mathbf{x}_i; \beta), \theta_i\right) K\left(d(\mathbf{x}_i, \mathbf{x}_0)\right)$$

LWR

- Fit the model $\theta = g(\mathbf{x}; \beta)$ to observations within a small neighborhood of the query point $\mathbf{x}_0$:

$$\beta^*(\mathbf{x}_0) = \underset{\beta}{\operatorname{argmin}} \sum_i L\left(g(\mathbf{x}_i; \beta), \theta_i\right) K\left(d(\mathbf{x}_i, \mathbf{x}_0)\right) \quad \text{where}$$

  – L is the *loss* function,

LWR

- Fit the model $\theta = g(\mathbf{x}; \beta)$ to observations within a small neighborhood of the query point $\mathbf{x}_0$:

$$\beta^*(\mathbf{x}_0) = \underset{\beta}{\operatorname{argmin}} \sum_i L\left(g(\mathbf{x}_i; \beta), \theta_i\right) K\left(d(\mathbf{x}_i, \mathbf{x}_0)\right) \quad \text{where}$$

  – L is the *loss* function,
  – K is the *kernel*, which determines the weight falloff with increasing distance from $\mathbf{x}_0$.

Robust LWR

- Introduced by [Cleveland].

Robust LWR

- Introduced by [Cleveland].
- Iterative process:

Robust LWR

- Introduced by [Cleveland].
- Iterative process:

(1) Set the weights according to $K(\mathbf{x}_0, \mathbf{x}_i)$.

Robust LWR

- Introduced by [Cleveland].

- Iterative process:

(1) Set the weights according to $K(\mathbf{x}_0, \mathbf{x}_i)$.
(2) Run weighted least squares.

Robust LWR

- Introduced by [Cleveland].

- Iterative process:

(1) Set the weights according to $K(\mathbf{x}_0, \mathbf{x}_i)$.
(2) Run weighted least squares.
(3) Estimate the residuals in θ

Robust LWR

- Introduced by [Cleveland].

- Iterative process:

(1) Set the weights according to $K(\mathbf{x}_0, \mathbf{x}_i)$.
(2) Run weighted least squares.
(3) Estimate the residuals in θ
(4) Reweight according to residuals and go back to (2)

Robust LWR

- Introduced by [Cleveland].
- Iterative process:

(1) Set the weights according to $K(\mathbf{x}_0, \mathbf{x}_i)$.
(2) Run weighted least squares.
(3) Estimate the residuals in θ
(4) Reweight according to residuals and go back to (2)

- Takes care of outliers.

Back to pose estimation

- LWR provides means of example-based regression.
- LSH allows rapid search for similar examples.
- Problem: similarity in the parameter space does not have to be consistent with similarity in feature space.
  - Need to derive useful similarity measure, and to modify the LSH apparatus as necessary.

Filters and hash functions

- Consider simple family of *binary* hash functions:

$$h_{\phi,T}(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$$

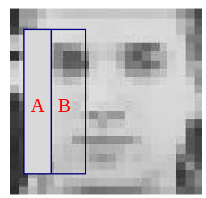
for given filter ϕ and threshold T.

Filters and hash functions

- Consider simple family of *binary* hash functions:

$$h_{\phi,T}(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$$

  for given filter ϕ and threshold T. Examples:

  – Box filters [Viola&Jones] 

  $\phi(\mathbf{x}) = \sum_A - \sum_B$.

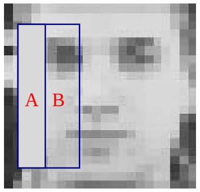
Filters and hash functions

- Consider simple family of *binary* hash functions:

$$h_{\phi,T}(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$$

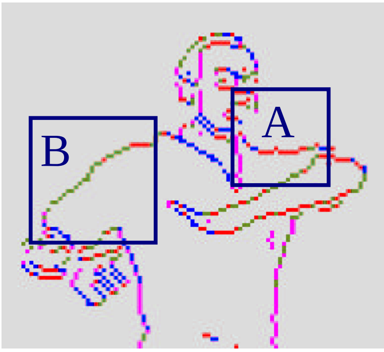
for given filter ϕ and threshold T. Examples:

– Box filters [Viola&Jones]



$\phi(\mathbf{x}) = \sum_A - \sum_B$.

– Edge direction histograms



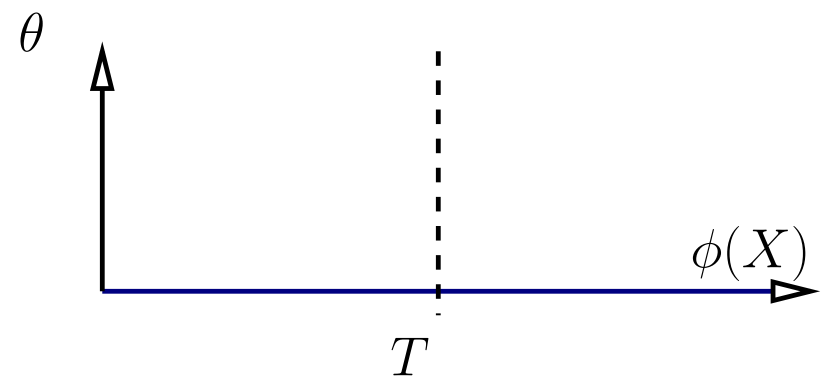
$\phi(\mathbf{x}) = \sum_A horiz.$

Hash functions and classification

- $h_{\phi,T}$ applied on a paired examples will either:

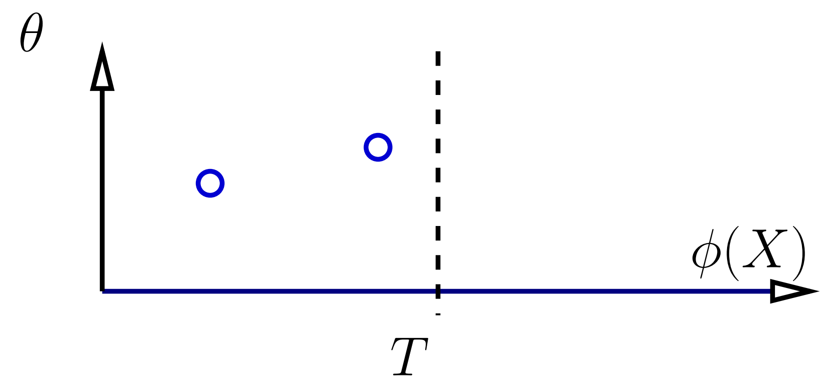
Hash functions and classification

- $h_{\phi,T}$ applied on a paired examples will either:



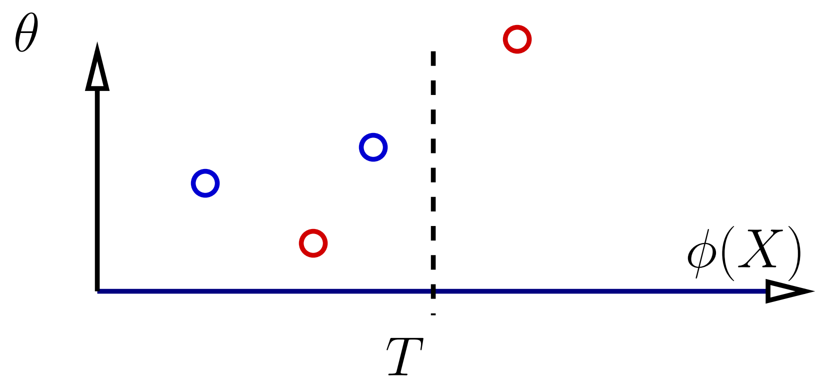
Hash functions and classification

- $h_{\phi,T}$ applied on a paired examples will either:
  - Place both in the same bin



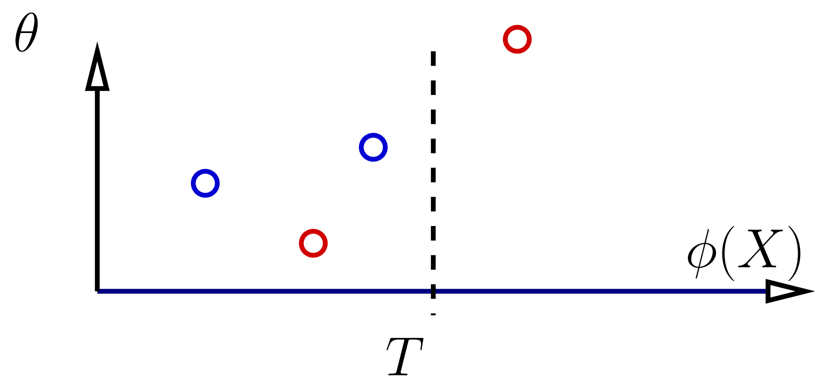
Hash functions and classification

- $h_{\phi,T}$ applied on a paired examples will either:
  - Place both in the same bin ,or
  - Separate between them.



Hash functions and classification

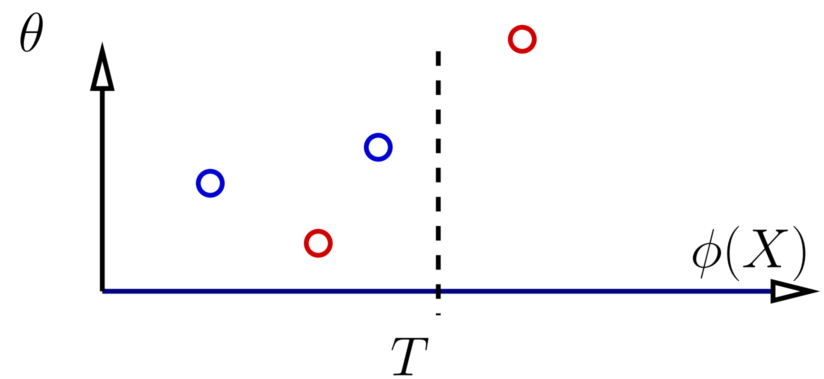
- $h_{\phi,T}$ applied on a paired examples will either:
  - Place both in the same bin ,or
  - Separate between them.



-  

Hash functions and classification

- $h_{\phi,T}$ applied on a paired examples will either:
  - Place both in the same bin ,or
  - Separate between them.



-    

Hash functions and classification

- $h \equiv h_{\phi,T}(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise.} \end{cases}$
- Let $(\mathbf{x}_i, \mathbf{x}_j)$ be a *paired example*.

Hash functions and classification

- $h \equiv h_{\phi,T}(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise.} \end{cases}$
- Let $(\mathbf{x}_i, \mathbf{x}_j)$ be a *paired example*.
- We define the label of $(\mathbf{x}_i, \mathbf{x}_j)$:

$$y_{ij} = \begin{cases} +1 & \text{if } \theta_i \text{ and } \theta_j \text{ are } r\text{-similar, i.e. } d_\theta\left(\theta_i, \theta_j\right) < r, \\ -1 & \text{otherwise.} \end{cases}$$

i.e. a paired example is positive if the two components have similar (within r) parameters.

Hash functions and classification

- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\qquad y_{ij} = \begin{cases} +1 & \text{if } d_\theta\left(\theta_i, \theta_j\right) < r, \\ -1 & \text{otherwise.} \end{cases}$

Hash functions and classification

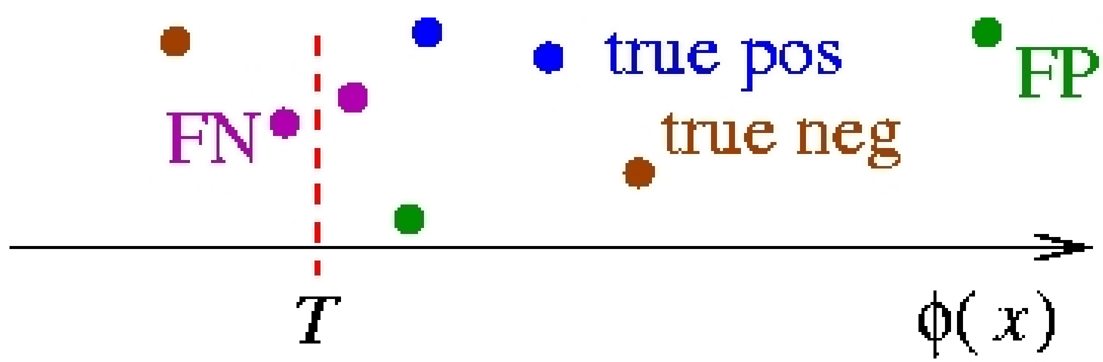
- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\quad y_{ij} = \begin{cases} +1 & \text{if } d_\theta(\theta_i, \theta_j) < r, \\ -1 & \text{otherwise.} \end{cases}$
- We define the *paired* classifier associated with h by

$$\hat{y}_h(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} +1 & \text{if } h(\mathbf{x}_i) = h(\mathbf{x}_j), \\ -1 & \text{otherwise.} \end{cases}$$

Hash functions and classification

- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\qquad y_{ij} = \begin{cases} +1 & \text{if } d_\theta(\theta_i, \theta_j) < r, \\ -1 & \text{otherwise.} \end{cases}$
- We define the *paired* classifier associated with h by

$$\hat{y}_h(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} +1 & \text{if } h(\mathbf{x}_i) = h(\mathbf{x}_j), \\ -1 & \text{otherwise.} \end{cases}$$



Hash functions and classification

- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\quad y_{ij} = \begin{cases} +1 & \text{if } d_\theta(\theta_i, \theta_j) < r, \\ -1 & \text{otherwise.} \end{cases}$

$$\hat{y}_h(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} +1 & \text{if } h(\mathbf{x}_i) = h(\mathbf{x}_j), \\ -1 & \text{otherwise.} \end{cases}$$

- Collision properties of h vs. accuracy of $\hat{y}_h$:

Hash functions and classification

- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\quad y_{ij} = \begin{cases} +1 & \text{if } d_\theta\left(\theta_i, \theta_j\right) < r, \\ -1 & \text{otherwise.} \end{cases}$

$$\hat{y}_h(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} +1 & \text{if } h(\mathbf{x}_i) = h(\mathbf{x}_j), \\ -1 & \text{otherwise.} \end{cases}$$

- Collision properties of h vs. accuracy of $\hat{y}_h$:
  - $p_1(h)$ is the *true negative rate* of $\hat{y}_h$,
  - $p_2(h)$ is the *false positive rate* of $\hat{y}_h$.

Hash functions and classification

- $h(\mathbf{x}) = \begin{cases} +1 & \text{if } \phi(\mathbf{x}) \geq T, \\ -1 & \text{otherwise} \end{cases}$ $\quad y_{ij} = \begin{cases} +1 & \text{if } d_\theta(\theta_i, \theta_j) < r, \\ -1 & \text{otherwise.} \end{cases}$

$$\hat{y}_h(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} +1 & \text{if } h(\mathbf{x}_i) = h(\mathbf{x}_j), \\ -1 & \text{otherwise.} \end{cases}$$

- Collision properties of h vs. accuracy of $\hat{y}_h$:
  - $p_1(h)$ is the *true negative rate* of $\hat{y}_h$,
  - $p_2(h)$ is the *false positive rate* of $\hat{y}_h$.

- Optimizing w.r.t. p_1 and $1 - p_2$ is a straightforward machine learning task.

Learning parameter-sensitive hash functions

Some practical aspects:

- Very unbalanced problem: few positive examples, very many very diverse negative examples.
- Only need to look at finite set of thresholds T
  - One-pass algorithm (in the ICCV '03 paper)
- Need to balance the two objectives: increasing p_1, decreasing p_2.
  - Not tied together, in contrast to the common learning practice.

PSH

- Given: set of labeled examples $\{\langle \mathbf{x}, \theta \rangle\}$, set of filters$\{\phi\}$.
- Sample paired training set:
  - positive example $=$ pair of images with similar θ,
  - negative example $=$ pair of images with far θ.
- Evaluate all $\phi(\mathbf{x})$, for each find optimal T.
- Select features with $p_1(h_{\phi,T}), p_2(h_{\phi,T})$ within target bounds.
- Proceed with LSH.

Articulated pose estimation

- 13 DOF (including torso rotation).
- Edge direction histograms: almost 12,000 features
- 500,000 syn thetic images (POSER)

Paired examples

POS

POS

AND

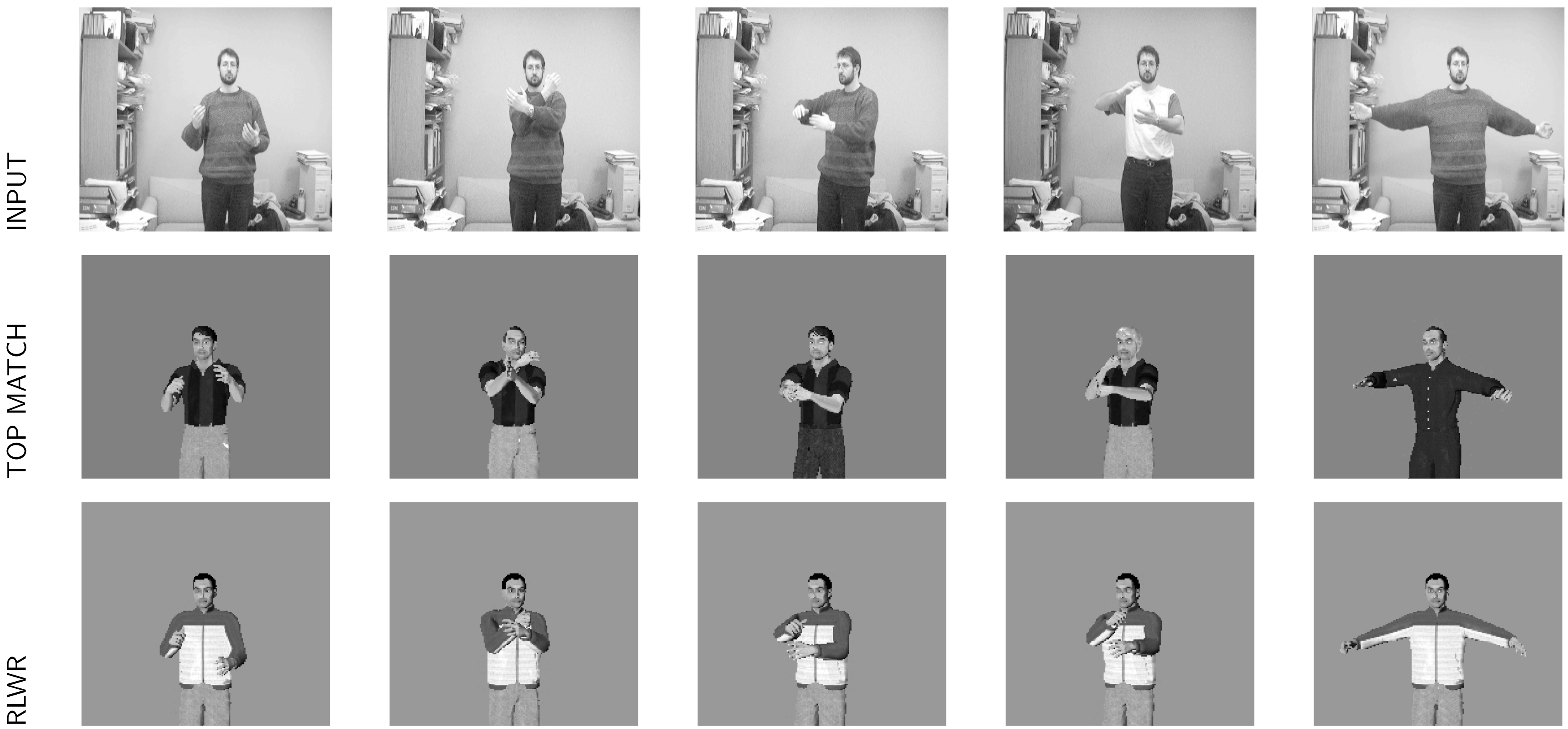
NEG

NEG

Articulated pose: experiments

- Selected 213 features / hash functions, with
  $p_1 \ = \ .85, \qquad p_2 \ = \ 0.52.$
- Used $l = 80$ hash tables, $k = 19$ bit hash functions.
- According to the theoretical analysis:
  - Probability of success: 0.985
  - Expected number of comparisons: 130

Results on real data



More results

INPUT

TOP MATCH

RLWR

Summary: NN in vision

- NN-based methods may be efficient for recognition and estimation in vision, when large data sets are available.
- Synthetically generated data may be useful in these scenarios.
- Using a single NN usually is not a good idea
- LWR, local density models usually are.

Summary: recipes

- Low dimension ($d \leq 10$), exact NN: use kd-trees
- Moderate dimension ($10 \geq d \leq 20$), approximate NN: use BBF
- High dimension: use LSH for approximate NN
- Consider whether r-neighbors, rather then NN, are the goal.
- Use embeddings for fast similarity search in metric spaces
- Carefully choose the distance function!

References

[1] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[2] J. S. Beis and D. G. Lowe. Indexing without invariants in 3D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1000–1015, 1999.

[3] Serge Belongie, Jitendra Malik, and Jan Puzicha. Matching shapes. In *International Conference on Computer Vision*, pages 454–463, Los Alamitos, CA, July 9–12 2001. IEEE Computer Society.

[4] W. S. Cleveland. Robust locally weighted regression and smoothing scatter plots. *Journal of American Statistical Association*, 74(368):829–836, 1979.

[5] W. S. Cleveland and S. J. Delvin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of American Statistical Association*, 83(403):596–610, 1988.

[6] T. M. Cover. Rates of Convergence for Nearest Neighbor Procedures. In *Proc. 1st Ann. Hawaii Conf. Systems Theory*, pages 413–415, January 1968.

[7] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, January 1967.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & sons, New York, second edition, 2001.

[9] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 518–529, San Francisco, September 1999. Morgan Kaufmann.

[10] K. Grauman and T. Darrell. Fast Contour Matching Using Approximate Earth-Mover's Distance. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Washington, DC, 2004. to appear.

[11] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 604–613, New York, May 23–26 1998. ACM Press.

[12] B. Leibe and B. Schiele. Analyzing contour and appearance based methods for object categorization. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, 2003.

[13] S. Niyogi and W. T. Freeman. Example-Based Head Tracking. Technical report, MERL, 1996.

[14] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *International Conference on Computer Vision*, Nice, France, October 2003.