Metric Embedding of Task-Specific Similarity

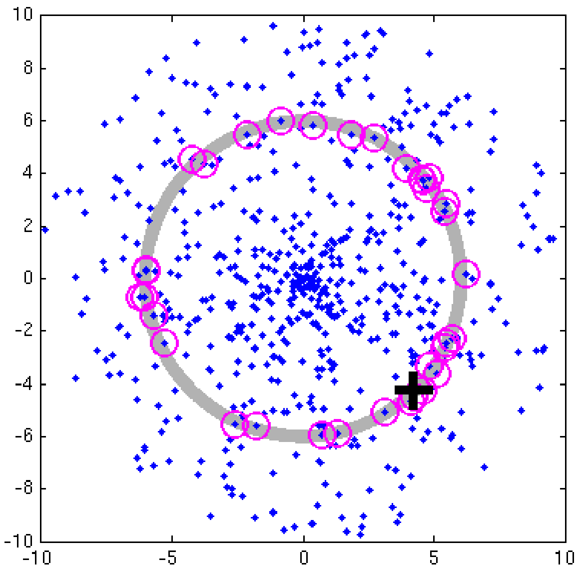
Greg Shakhnarovich
Brown University

joint work with Trevor Darrell (MIT)

August 19, 2006

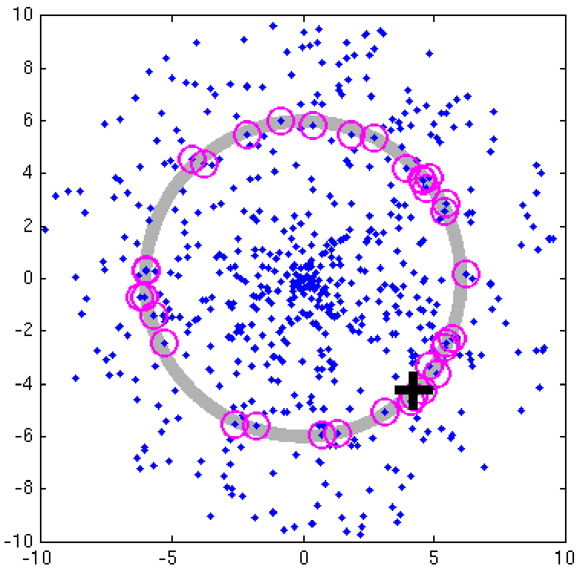
Task-specific similarity

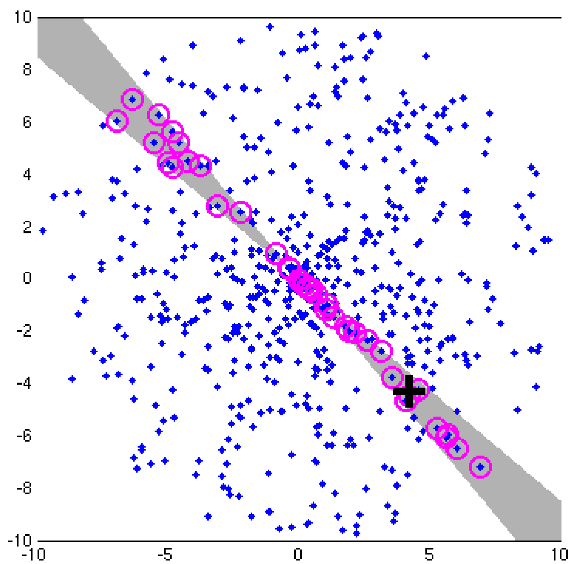
- A toy example:



Task-specific similarity

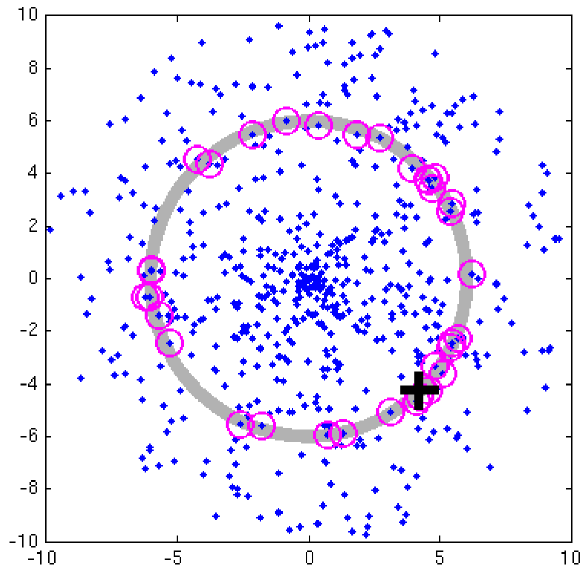
- A toy example:



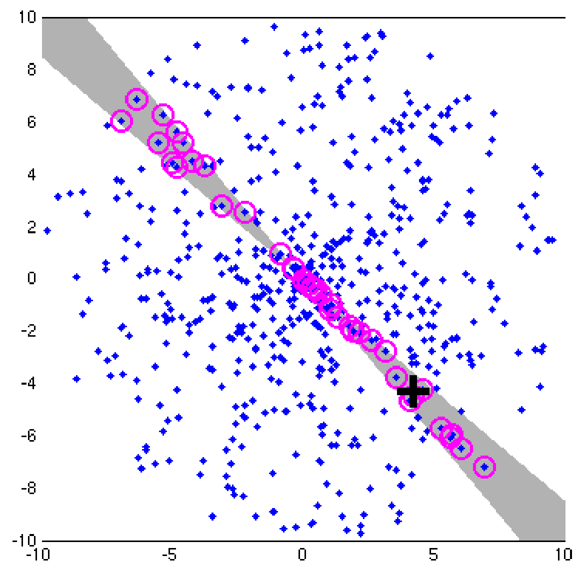


Task-specific similarity

- A toy example:



Norm



Angle

The problem

- Learn similarity from examples of what is similar [or not].
  - *Binary* similarity: for $\mathbf{x}, \mathbf{y} \in \mathcal{X}$

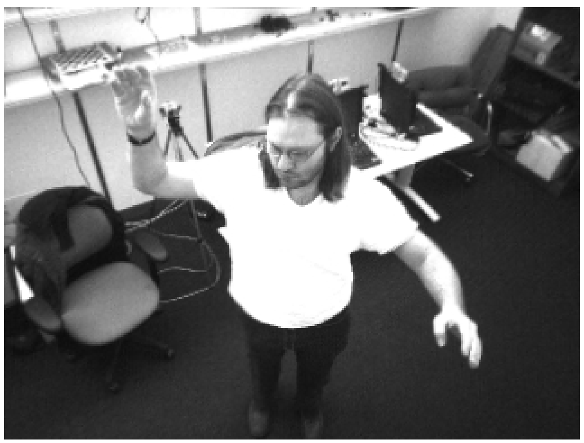
$$\mathcal{S}(\mathbf{x}, \mathbf{y}) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ and } \mathbf{y} \text{ are similar,} \\ -1 & \text{if they are dissimilar.} \end{cases}$$

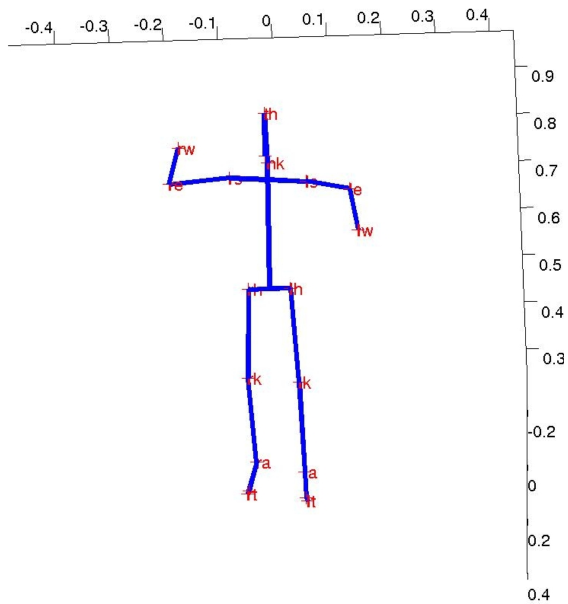
- Two goals in mind:
  - *Similarity detection*: judge whether two entities are similar.
  - *Similarity search*: given a *query* entity and a database, find examples in a database that are similar to the query.

- Our approach: learn an *embedding* of the data into a space where similarity corresponds to a simple distance.

Task-specific similarity

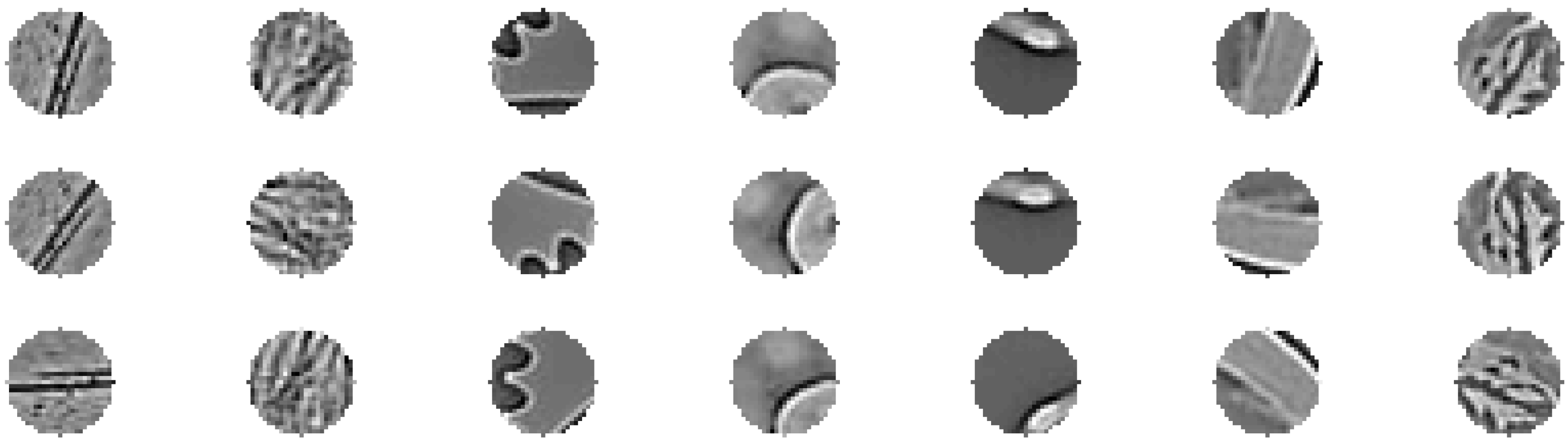
- Articulated pose:





Task-specific similarity

- Visual similarity of image patches:



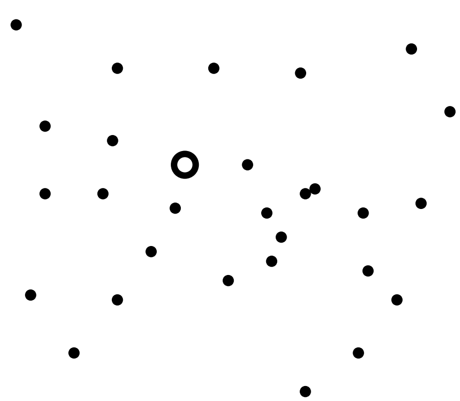
Related prior work

- Learning parametrized distance metric [Xing *et al*; Roweis], such as Mahalanobis distances.
- Lots of work on low-dimensional graph embedding (MDS, LLE,. . .) - but unclear how to generalize without relying on distance in $\mathcal{X}$.
- Embedding known distance [Athitsos *et al*]: assumes known distance, uses embedding to approximate/speed up.
- DistBoost [Hertz *et al*]: learning distance for classification / clustering setup.
- Locality sensitive hashing: fast similarity search.

Locality Sensitive Hashing [**Indyk** *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

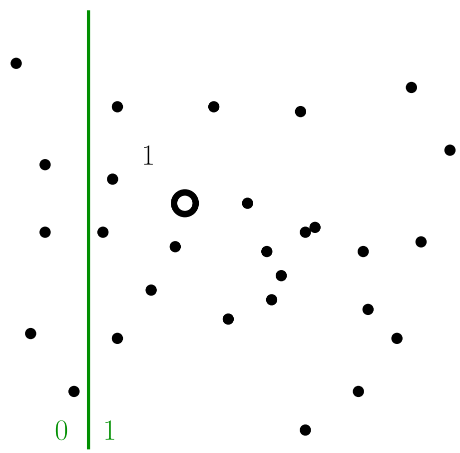
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

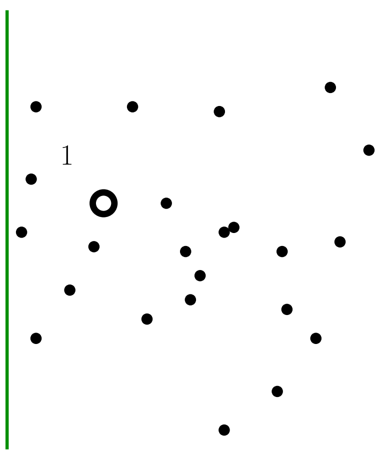
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

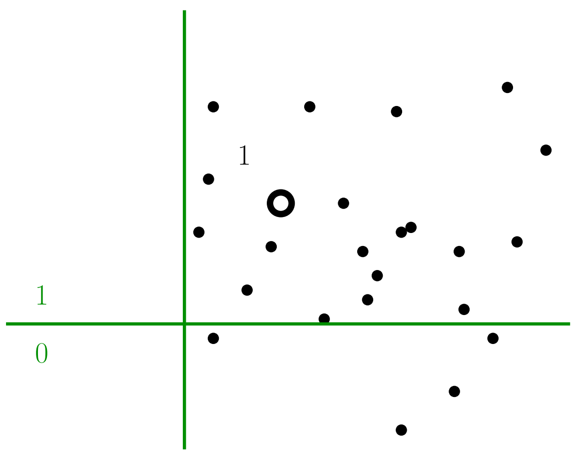
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

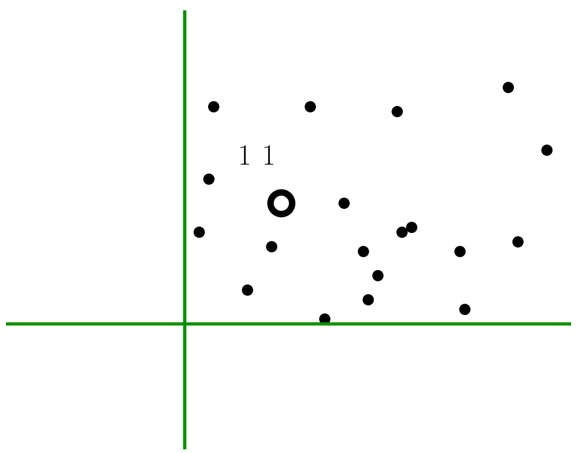
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

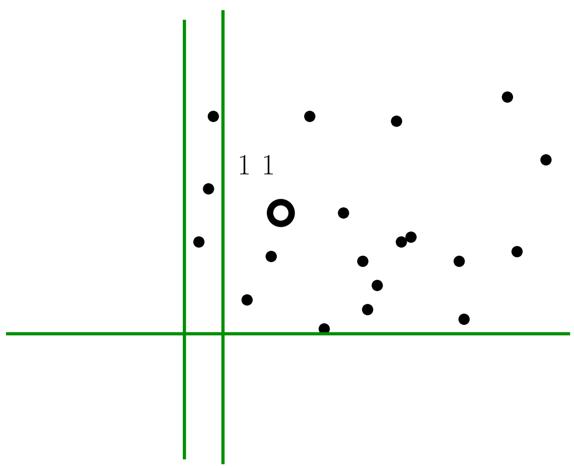
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

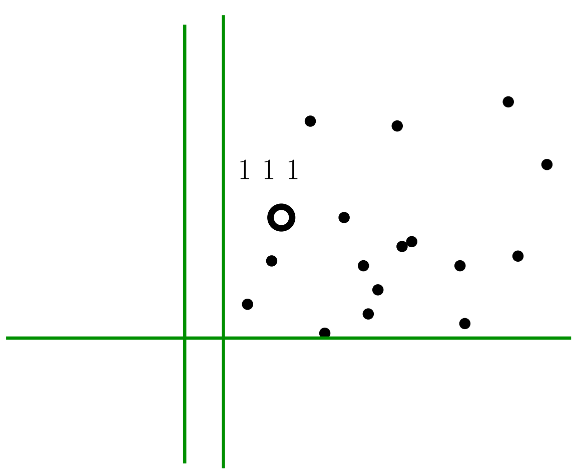
- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

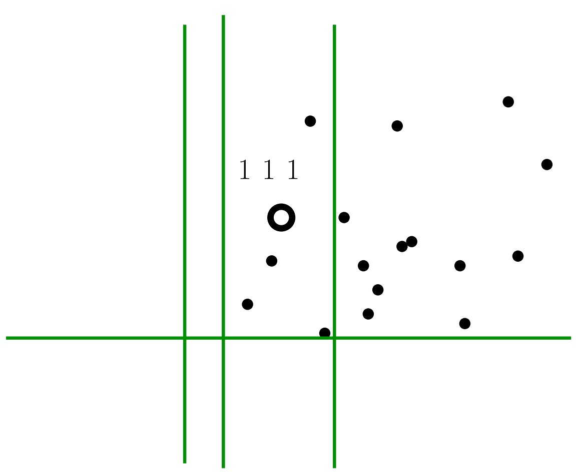
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

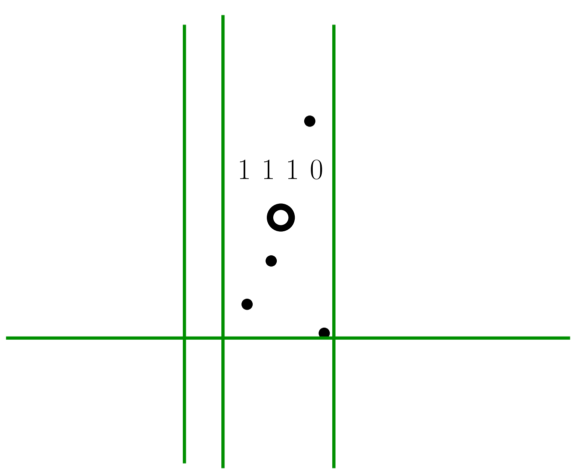
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

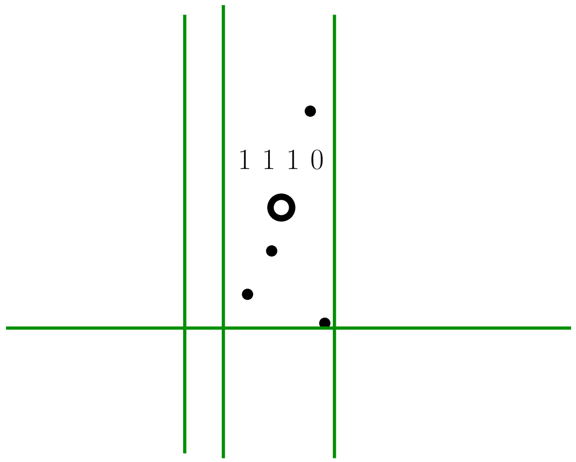
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:

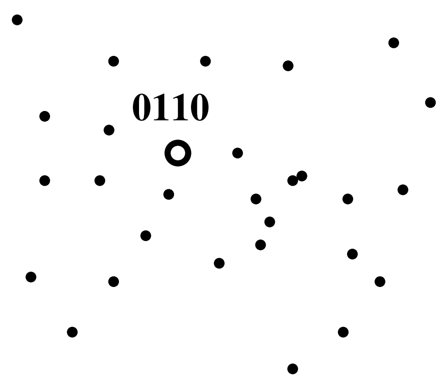


Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:

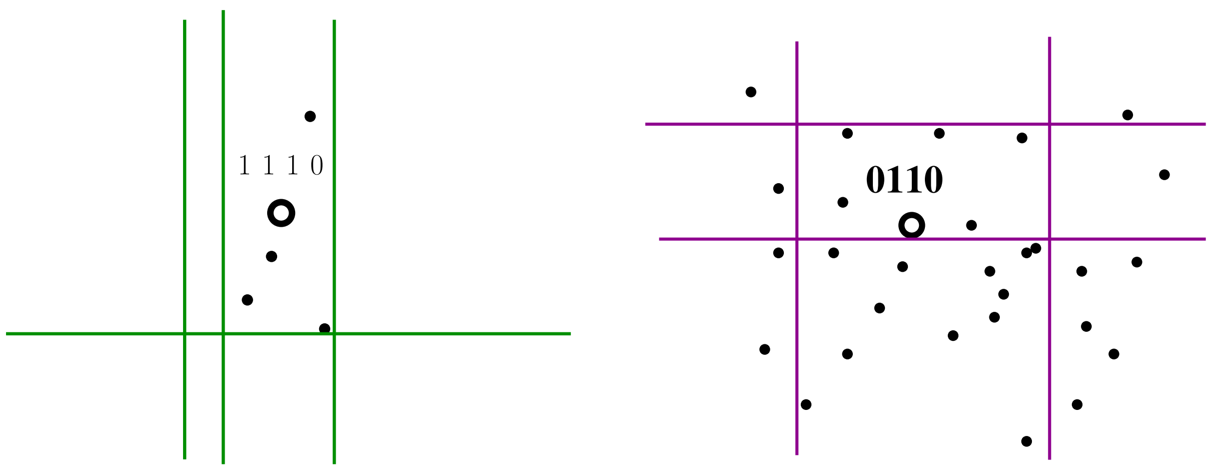




Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

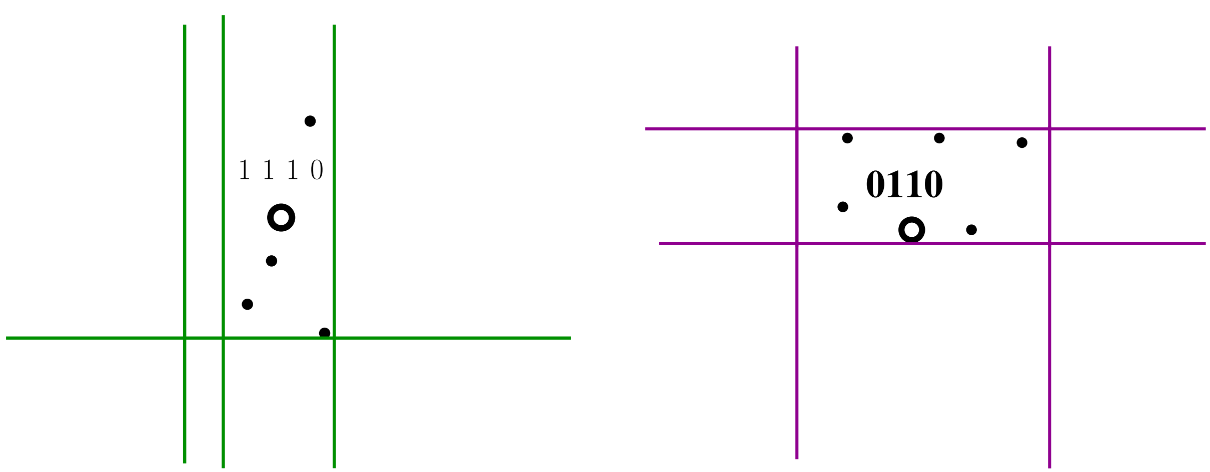
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

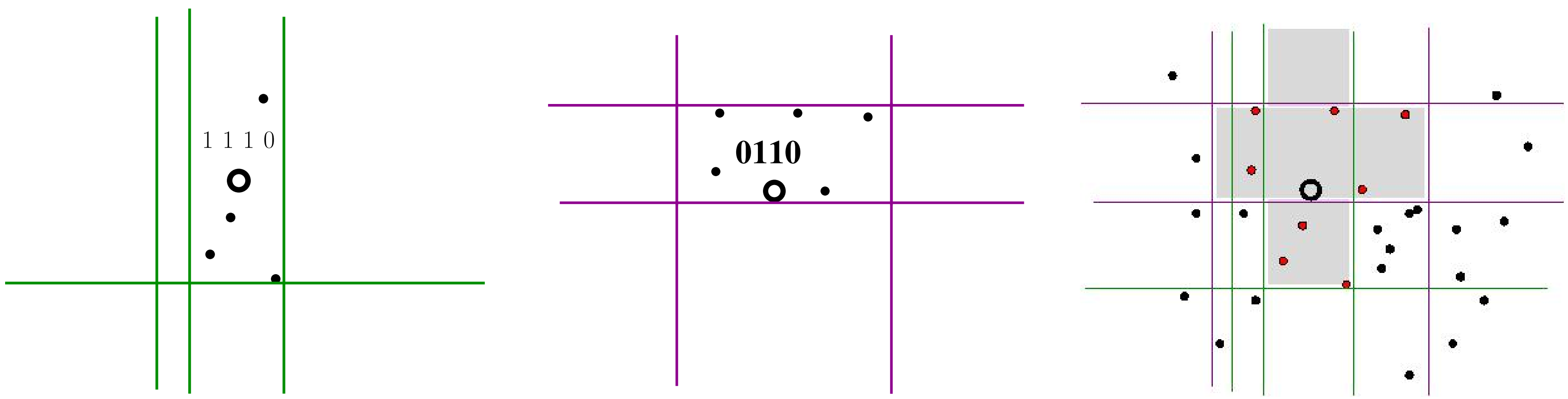
- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality Sensitive Hashing [Indyk *et al*]

- Algorithm for finding a (ϵ, r)-neighbor of $\mathbf{x}_0$ with high probability in sublinear time $O\left(N^{1/(1+\epsilon)}\right)$.

- Index the data by l random hash functions, and only search the union of the l buckets where the query falls:



Locality sensitive hashing [Indyk *et al*]

- A family $\mathcal{H}$ of functions is *locality sensitive* if

$$P_{h\sim U[\mathcal{H}]}\left(h(\mathbf{x}_0) = h(\mathbf{x}) \mid \|\mathbf{x}_0 - \mathbf{x}\| \le r\right) \ \ge \ p_1,$$
$$P_{h\sim U[\mathcal{H}]}\left(h(\mathbf{x}_0) = h(\mathbf{x}) \mid \|\mathbf{x}_0 - \mathbf{x}\| \ge R\right) \ \le \ p_2.$$

- Uses the *gap* between TP and FP rates;
  - "amplified" by concatenating functions into a hash key.

- Projections on random lines are locality sensitive w.r.t. L_p norms, $p \le 2$ [Gionis *et al*, Datar *et al*].

How is this relevant?

- LSH is excellent if L_p is all we want.
- L_p may not be a suitable “proxy” for $\mathcal{S}$: we may
  - “Waste” lots of bits on irrelevant features;
  - Miss pairs similar under $\mathcal{S}$ but not close w.r.t. L_p
- If we know what $\mathcal{S}$ is, may be able to analytically design *embedding* of $\mathcal{X}$ into L_1 space [Thaper&Indyk, Grauman&Darrell].
- We will instead *learn* LSH-style binary functions that fit $\mathcal{S}$ as conveyed by examples.

Our approach

- Given: pairs of similar [and pairs of dissimilar] examples, based on the "hidden" binary similarity $\mathcal{S}$.
- Two related tasks:
  - Similarity judgment: $S(\mathbf{x}, \mathbf{y}) = ?$
  - Given a query $\mathbf{x}_0$; need to find $\{\mathbf{x}_i \,:\, S(\mathbf{x}_i, \mathbf{x}_0) = +1\}$.
- Our solution to both problems: a *similarity sensitive* embedding

$$H(\mathbf{x}) \;=\; [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})]\,;$$

- We will learn $h_m(\mathbf{x}) \in \{0, 1\}$ and α_m.

Desired embedding properties

Embedding $H(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})]$:

- Rely on L_1 (= weighted Hamming) distance

$$\|H(\mathbf{x}_1) - H(\mathbf{x}_2)\| = \sum_{m=1}^{M} \alpha_m |h_m(\mathbf{x}_1) - h_m(\mathbf{x}_2)|$$

- H is *similarity sensitive*: for some R, want
  - high $P_{\mathbf{x}_1,\mathbf{x}_2 \sim p(\mathbf{x})}(\|H(\mathbf{x}_1) - H(\mathbf{x}_2)\| \leq R \,|\, \mathcal{S}(\mathbf{x}_1, \mathbf{x}_2) = +1)$,
  - low $P_{\mathbf{x}_1,\mathbf{x}_2 \sim p(\mathbf{x})}(\|H(\mathbf{x}_1) - H(\mathbf{x}_2)\| \leq R \,|\, \mathcal{S}(\mathbf{x}_1, \mathbf{x}_2) = -1)$.

- $\|H(\mathbf{x}) - H(\mathbf{y})\|$ is a proxy for $\mathcal{S}$.

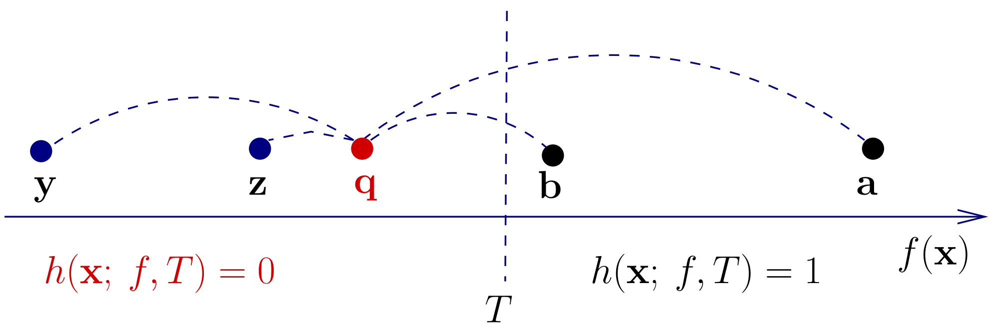
Projection-based classifiers

- For a projection $f : \mathcal{X} \to \mathbb{R}$, consider, for some $T \in \mathbb{R}$,

$$h(\mathbf{x};\ f,T) = \begin{cases} 1 & \text{if } f(\mathbf{x}) \le T \\ 0 & \text{if } f(\mathbf{x}) > T. \end{cases}$$

- This defines a simple similarity classifier of *pairs*:

$$c(\mathbf{x},\mathbf{y};\ f,T) = +1 \iff h(\mathbf{x};\ f,T) = h(\mathbf{y};\ f,T)$$

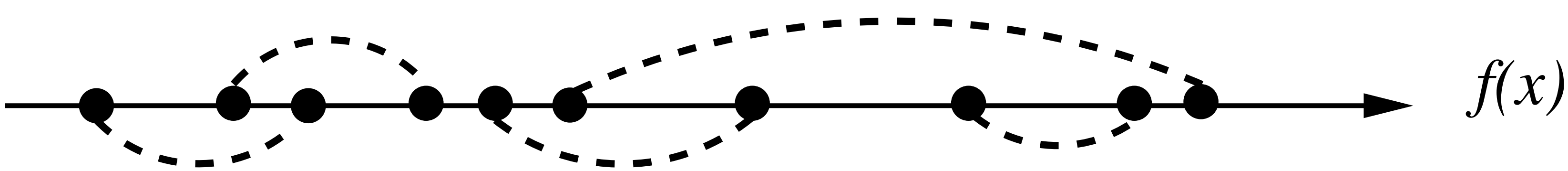


$$c(\mathbf{q},\mathbf{y};\ f,T) = c(\mathbf{q},\mathbf{z};\ f,T) = +1,$$
$$c(\mathbf{q},\mathbf{a};\ f,T) = c(\mathbf{q},\mathbf{b};\ f,T) = -1.$$

Selecting the threshold

Algorithm for selecting the threshold based on similar/dissimilar pairs:

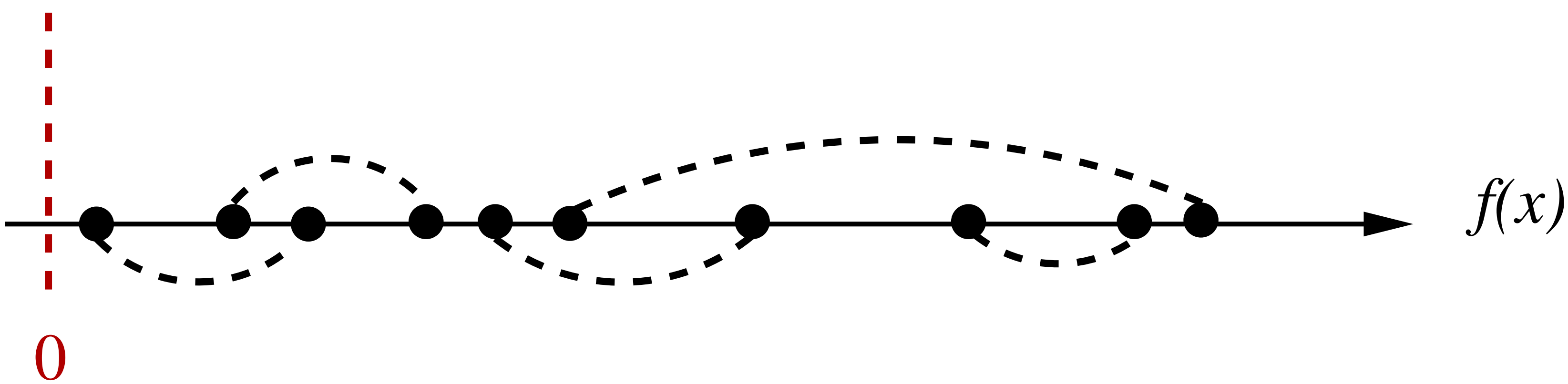
- For a moment, we focus on N positive pairs only.
- Sort the $2N$ values of $f(\mathbf{x})$.
- Need to check at most $2N+1$ values of T, and count the number of pairs that are dissected (a “bad” event).



Selecting the threshold

Algorithm for selecting the threshold based on similar/dissimilar pairs:

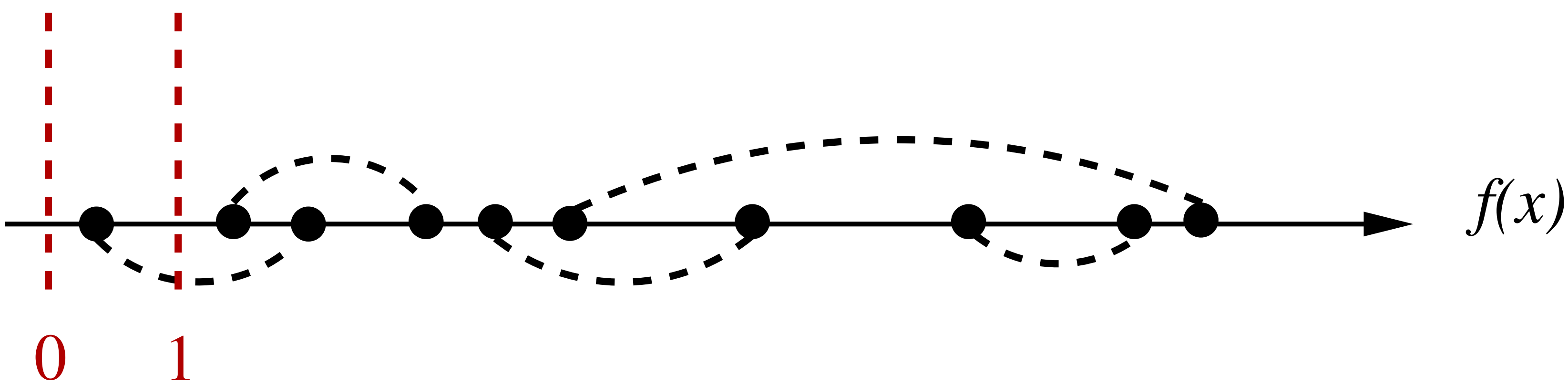
- For a moment, we focus on N positive pairs only.
- Sort the $2N$ values of $f(\mathbf{x})$.
- Need to check at most $2N+1$ values of T, and count the number of pairs that are dissected (a "bad" event).



Selecting the threshold

Algorithm for selecting the threshold based on similar/dissimilar pairs:

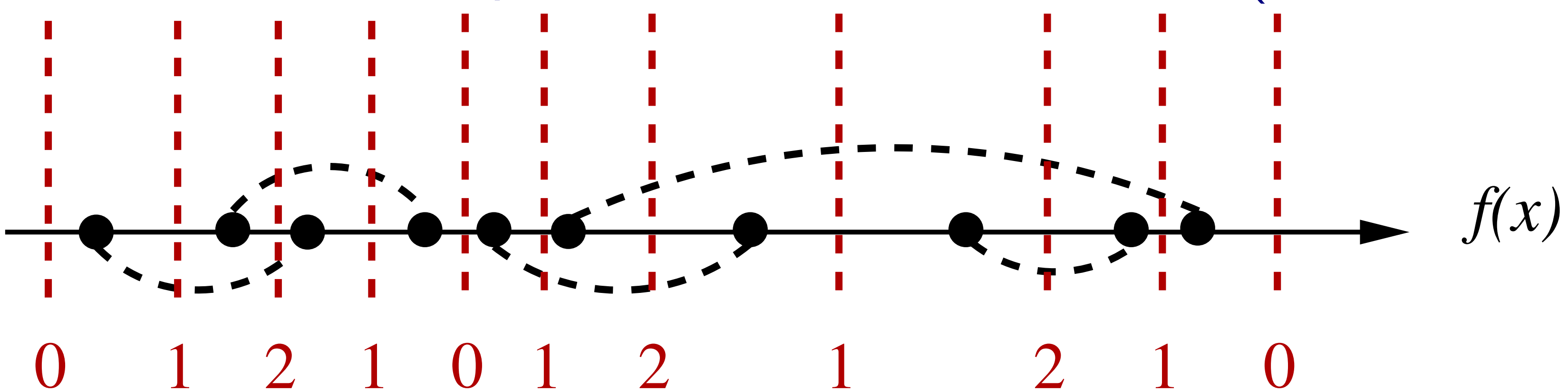
- For a moment, we focus on N positive pairs only.
- Sort the $2N$ values of $f(\mathbf{x})$.
- Need to check at most $2N+1$ values of T, and count the number of pairs that are dissected (a "bad" event).



Selecting the threshold

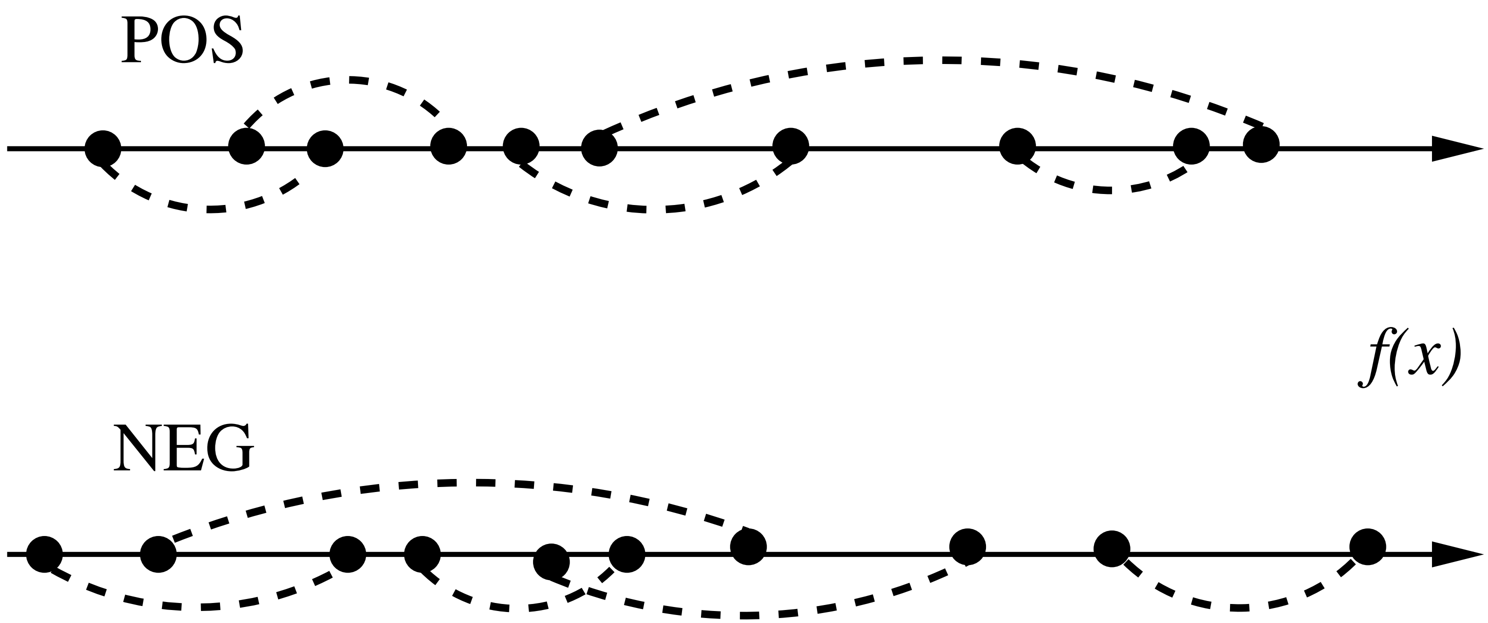
Algorithm for selecting the threshold based on similar/dissimilar pairs:

- For a moment, we focus on N positive pairs only.
- Sort the $2N$ values of $f(\mathbf{x})$.
- Need to check at most $2N+1$ values of T, and count the number of pairs that are dissected (a "bad" event).



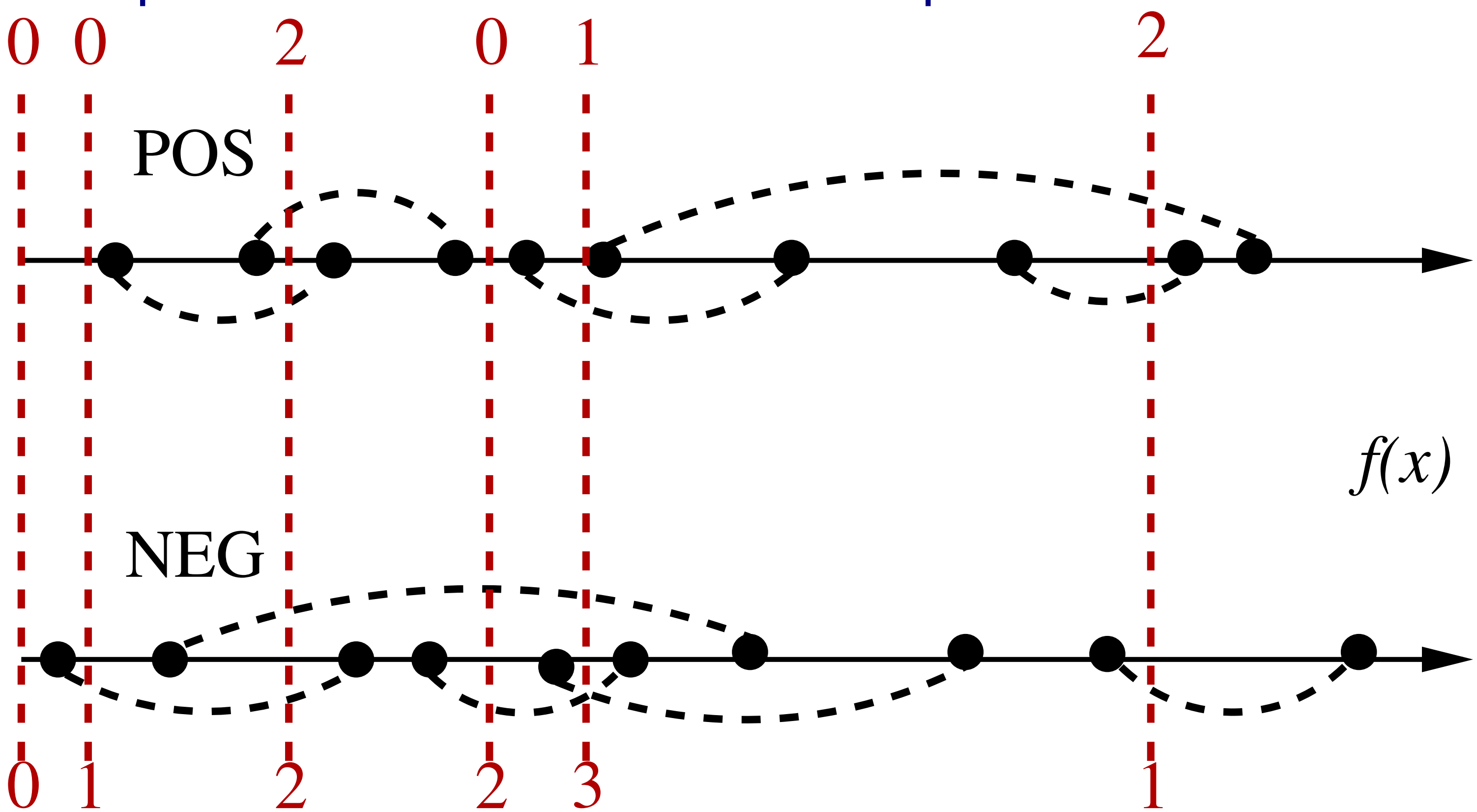
Selecting the threshold

Also need to consider negative examples (dissimilar pairs), and estimate the *gap*:
true positive rate TP minus false positive rate FP.



Selecting the threshold

Also need to consider negative examples (dissimilar pairs), and estimate the *gap*:
true positive rate TP minus false positive rate FP.



Optimization of $h(\mathbf{x}; f, T)$

- Objective (TP – FP gap):

$$\underset{f,T}{\operatorname{argmax}} \sum_{i=1}^{N^+} c(\mathbf{x}_{i1}^+, \mathbf{x}_{i1}^+) - \sum_{j=1}^{N^-} c(\mathbf{x}_{j1}^-, \mathbf{x}_{j1}^-)$$

- Parametric projection families, e.g. $f(\mathbf{x}) = \sum_j \theta_j \mathbf{x}_{(d_j^1)}^{p_1} \mathbf{x}_{(d_j^2)}^{p_2}$

- "Soft" versions of h and c make the gap differentiable w.r.t. θ, T:

$$h(\mathbf{x}; f, T) \;=\; \frac{1}{1 + \exp\left(f(\mathbf{x}) - T\right)}$$

$$c(\mathbf{x}, \mathbf{y}) \;=\; 4\left(h(\mathbf{x}) - 1/2\right)\left(h(\mathbf{y}) - 1/2\right)$$

Ensemble classifiers

- A weighted embedding $H(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})]$.
- Each $h_m(\mathbf{x})$ defines a classifier c_m. Together they form an *ensemble* classifier of similarity:

$$C(\mathbf{x}, \mathbf{y}) \; = \; \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m c_m(\mathbf{x}, \mathbf{y})\right).$$

- We will construct the ensemble of bit-valued hs by a greedy algorithm based on AdaBoost (operting on the corresponding cs).

Boosting [Schapire *et al*]

- Assumes access to *weak learner* that can at every iteration m produce a classifier c_m better than chance.

- Main idea: maintain a distribution $W_m(i)$ on the training examples, and update it according to the prediction of c_m:
  - If $c_m(\mathbf{x}_i)$ is correct, then $W_{m+1}(i)$ goes down;
  - Otherwise, $W_{m+1}(i)$ increases ("steering" c_{m+1} towards it.)

- Our examples are *pairs*, weak classifiers are thresholded projections.

- To evaluate threshold, we will calculate the weight of separated pairs, rather than count them.

BoostPro

Given pairs $(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ labeled with $l_i = \mathcal{S}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$:

Initialize weights $W_1(i)$, to uniform.
for all $m = 1, \ldots, M$ **do**
    Find $\langle f^*, T^* \rangle = \mathrm{argmax}_{f,T}\, r_m(f, T)$ using gradient descent on

$$r_m(f, T) \;=\; \sum_{i=1}^{N} W_m(i) l_i c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}).$$

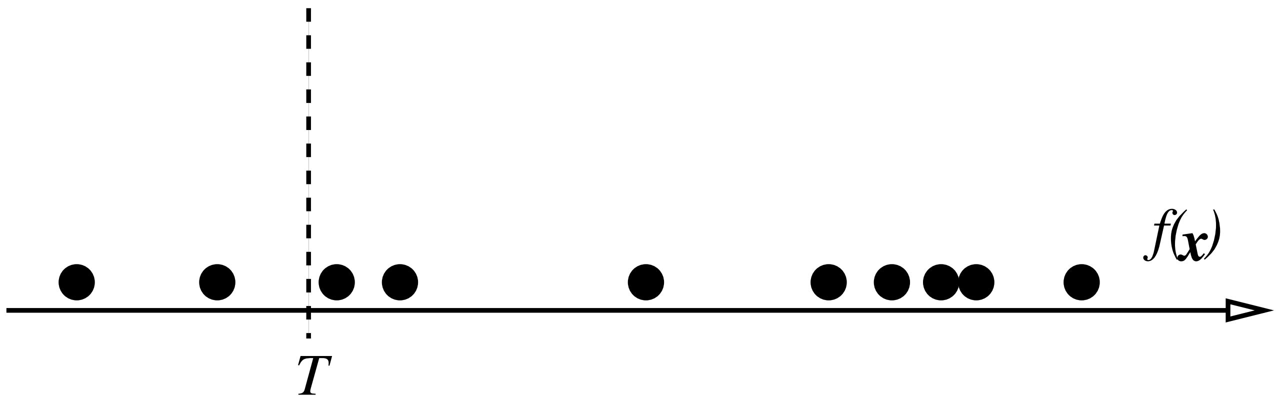
    Set $h_m \;\equiv\; h(\mathbf{x};\; f^*, T^*)$.
    Set α_m (see Boosting papers.)
    Update weights: $W_{m+1}(i) \propto W_m(i) \exp\left(-l_i c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})\right)$.

Similarity is a rare event

- In many domains: vast majority of possible pairs are negative.
  - People's poses, image patches, documents,...
- A reasonable approximation of the sampling process:
  - Independently draw $\mathbf{x}, \mathbf{y}$ from the data distribution.
  - $f(\mathbf{x})$, $f(\mathbf{y})$ drawn from $p(f(\mathbf{x}))$.
  - Label $(\mathbf{x}, \mathbf{y})$ negative.

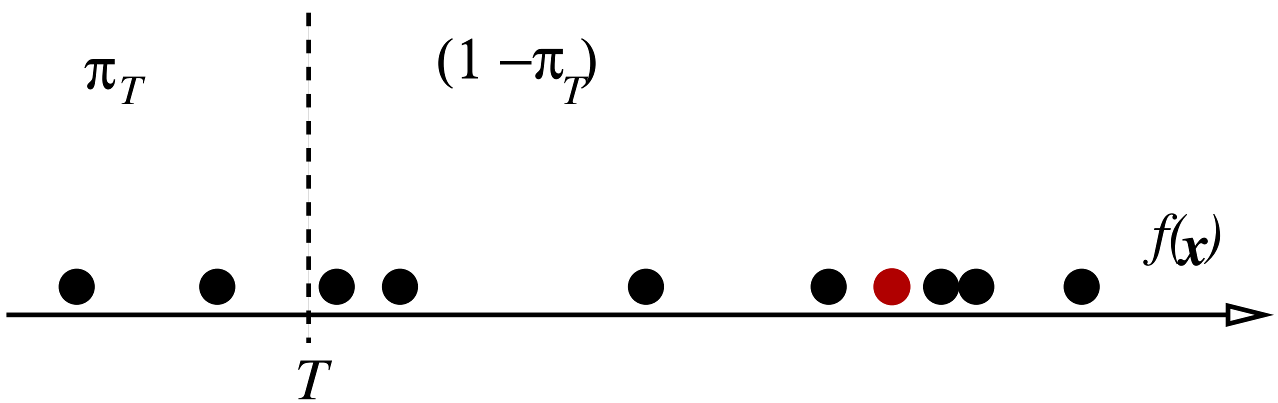
Similarity is a rare event

- In many domains: vast majority of possible pairs are negative.
  - People's poses, image patches, documents,...
- A reasonable approximation of the sampling process:
  - Independently draw $\mathbf{x}, \mathbf{y}$ from the data distribution.
  - $f(\mathbf{x})$, $f(\mathbf{y})$ drawn from $p(f(\mathbf{x}))$.
  - Label $(\mathbf{x}, \mathbf{y})$ negative.



Similarity is a rare event

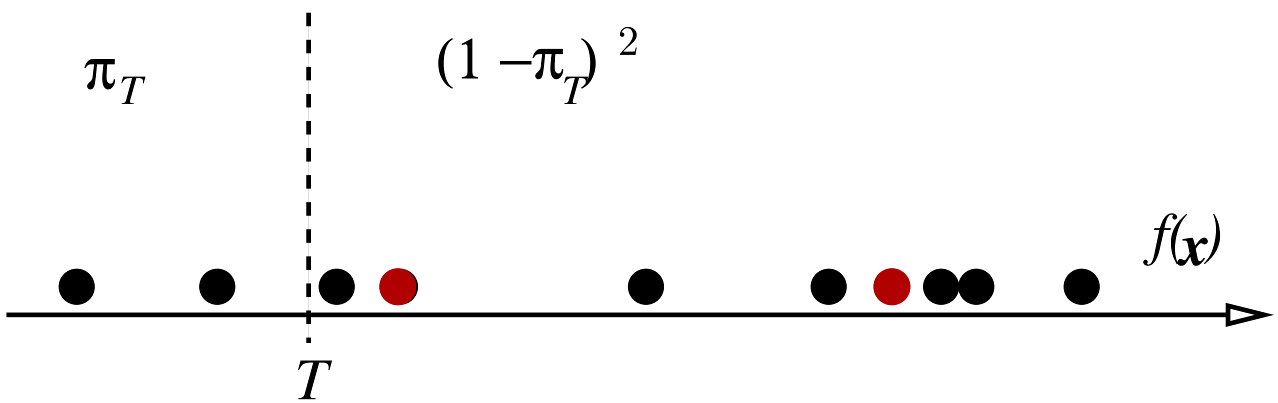
- In many domains: vast majority of possible pairs are negative.
  - People's poses, image patches, documents,...
- A reasonable approximation of the sampling process:
  - Independently draw $\mathbf{x}, \mathbf{y}$ from the data distribution.
  - $f(\mathbf{x})$, $f(\mathbf{y})$ drawn from $p(f(\mathbf{x}))$.
  - Label $(\mathbf{x}, \mathbf{y})$ negative.



$$\pi_T = \Pr(f(\mathbf{x}) \leq T)$$

Similarity is a rare event

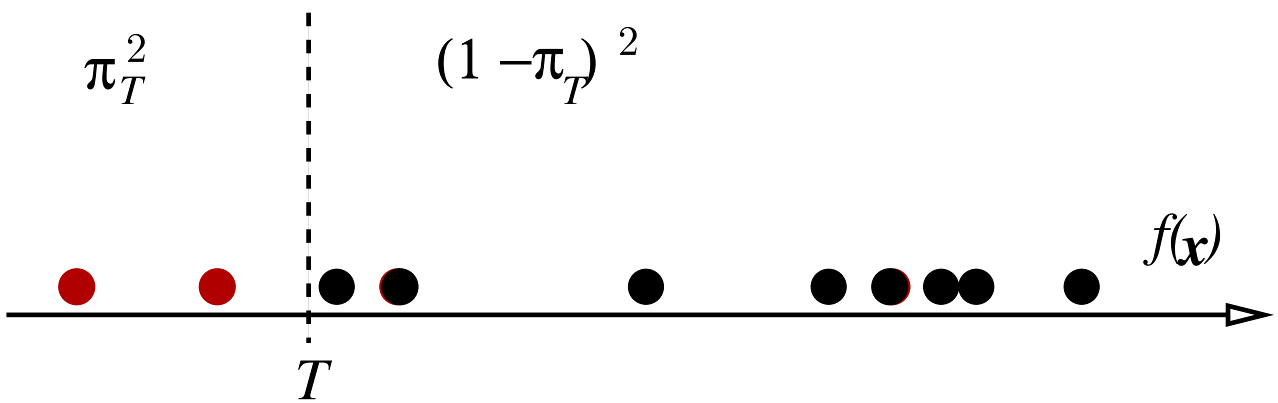
- In many domains: vast majority of possible pairs are negative.
  - People's poses, image patches, documents,...
- A reasonable approximation of the sampling process:
  - Independently draw $\mathbf{x}, \mathbf{y}$ from the data distribution.
  - $f(\mathbf{x})$, $f(\mathbf{y})$ drawn from $p(f(\mathbf{x}))$.
  - Label $(\mathbf{x}, \mathbf{y})$ negative.



$$\pi_T = \Pr(f(\mathbf{x}) \leq T)$$

Similarity is a rare event

- In many domains: vast majority of possible pairs are negative.
  - People's poses, image patches, documents,...
- A reasonable approximation of the sampling process:
  - Independently draw $\mathbf{x}, \mathbf{y}$ from the data distribution.
  - $f(\mathbf{x})$, $f(\mathbf{y})$ drawn from $p(f(\mathbf{x}))$.
  - Label $(\mathbf{x}, \mathbf{y})$ negative.



$$\pi_T = \Pr(f(\mathbf{x}) \le T)$$

Semi-supervised setup

- Given similar pairs and unlabeled $\mathbf{x}_1, \ldots, \mathbf{x}_N$.
- Estimate TP rate as before.
- FP rate:
  - Estimate the CDF of $f(\mathbf{x})$; let $\pi_T = \hat{P}(f(\mathbf{x}) \leq T)$.
  - Then $\widehat{\mathsf{FP}} = \pi_T^2 + (1 - \pi_T)^2$.
- Note: this means FP $\geq 1/2$ [Ke *et al*].

BoostPro in a semi-supervised setup

- "Normal" boosting assumes positive and negative examples.
- Intuition: each unlabeled example $\mathbf{x}_i$ represents *all* possible pairs $(\mathbf{x}_i, \mathbf{y})$; those are, w.h.p., negative under our assumption.
- Two distributions:
  - $W_m(j)$ on positive pairs, as before.
  - $S_m(i)$ on unlabeled (single) examples.
- Instead of $c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ use the *expectation* $E_{\mathbf{y}}\left[c_m(\mathbf{x}_i, \mathbf{y})\right]$ to calculate the objective and set the weights.

Semi-supervised boosting: details

- Probability of misclassifying a negative $(\mathbf{x}_j, \mathbf{y})$:

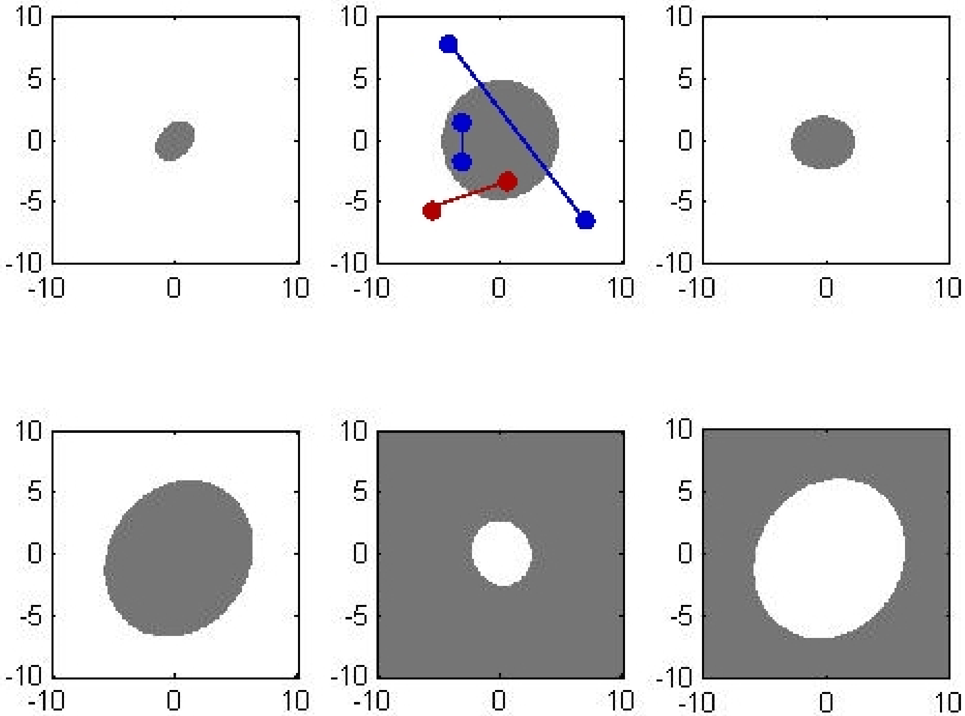
$$P_j \;=\; h(\mathbf{x}_j; f, T)\pi + (1 - h(\mathbf{x}_j; f, T))(1 - \pi).$$

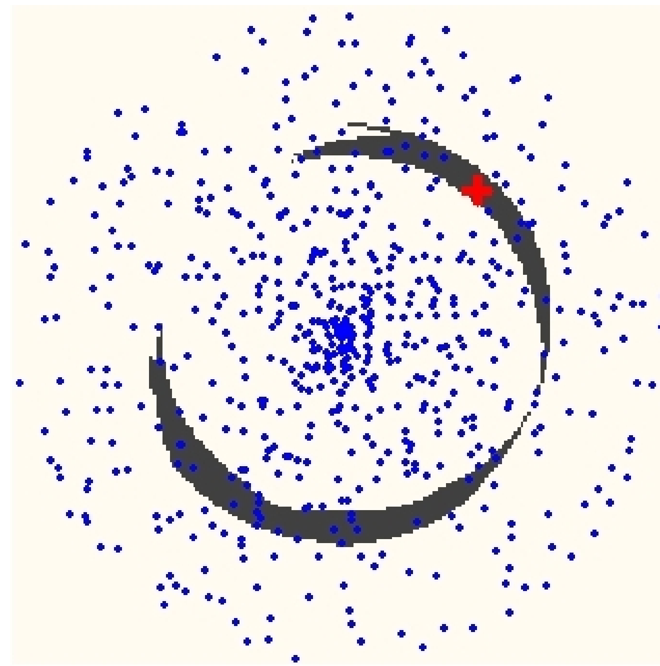
- $E_{\mathbf{y}}\left[c(\mathbf{x}_j, \mathbf{y})\right] = P_j \cdot (+1) + (1 - P_j) \cdot (-1) = 2P_j - 1.$

- Modified boosting objective:

$$\begin{aligned} r \;&=\; \sum_{i=1}^{N_p} W(i) c(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) \;-\; \sum_{j=1}^{N} S(j) E_{\mathbf{y}}\left[c(\mathbf{x}_j, \mathbf{y})\right] \\ &=\; \sum_{i=1}^{N_p} W(i) c(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) \;-\; \sum_{j=1}^{N} S_j (2P_j - 1). \end{aligned}$$

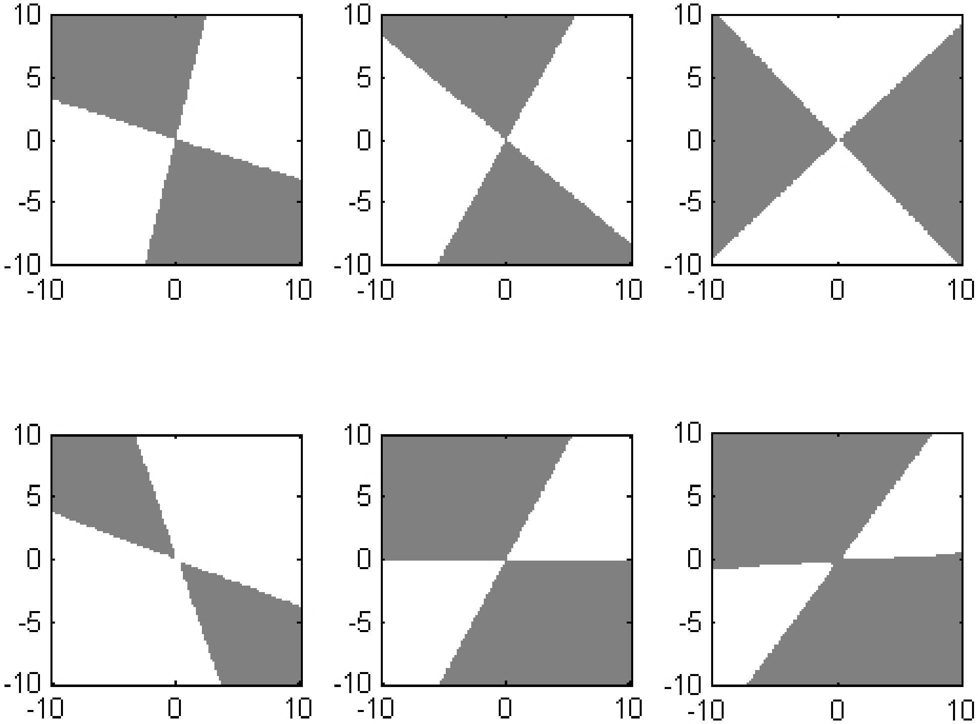
Results: Toy problems

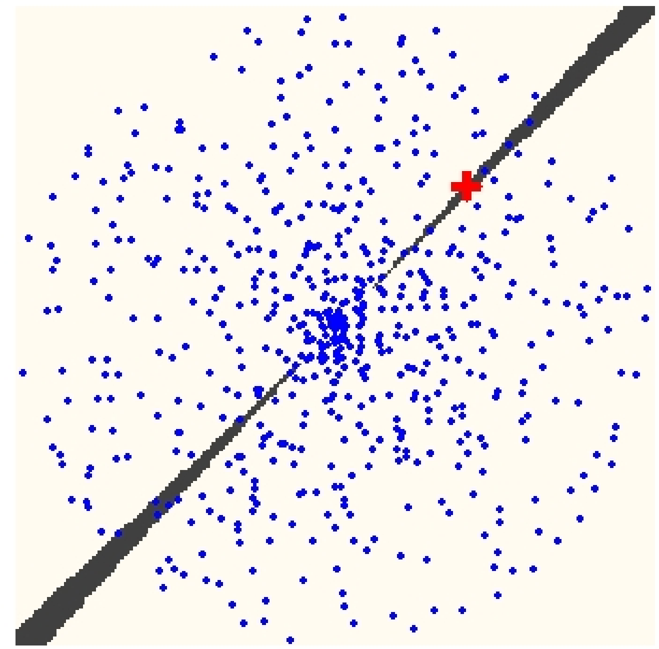




M=100, trained on 600 unlabeled points + 1,000 similar pairs

Results: Toy problems





Results: UCI data sets

Data Set	L_1	PSH	BoostPro	M
MPG	13.9436 $\pm$ 5.1276	10.7168 $\pm$ 4.3401	7.4905 $\pm$ 2.5907	180$\pm$ 20
CPU	37.9810 $\pm$ 5.2729	59.3767 $\pm$ 17.4186	9.0846 $\pm$ 0.9953	115 $\pm$ 48
Housing	26.5211 $\pm$ 6.8080	13.8464 $\pm$ 9.2756	13.8436 $\pm$ 8.4188	210 $\pm$ 28
Abalone	4.7816 $\pm$ 0.5180	5.0842 $\pm$ 0.4960	4.7602 $\pm$ 0.4384	43 $\pm$ 8
Census	2.493×10^9 $\pm$ 3.3×10^8	2.237×10^9 $\pm$ 3.2×10^8	1.566×10^9 $\pm$ 2.4×10^8	49 $\pm$ 10

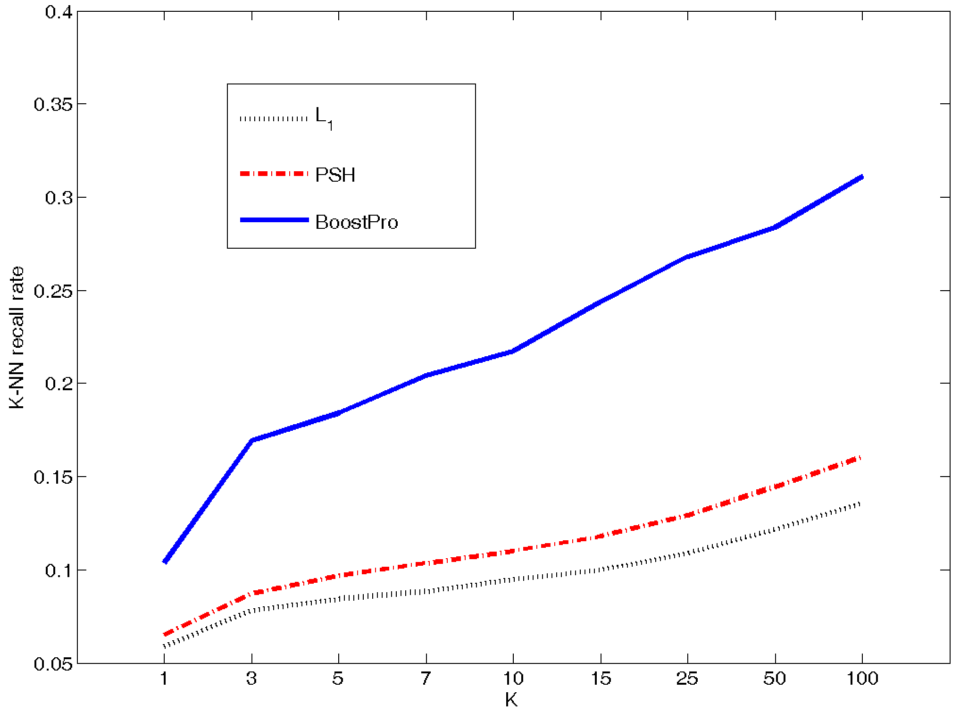
Test error on regression benchmark data from UCI/Delve. Mean $\pm$ std. deviation of MSE using locally-weighted regression. The last column shows the values of M (dimension of embedding H). Similarity defined in terms of target function values.

Results: pose retrieval

Input — 3 nearest neighbors in H

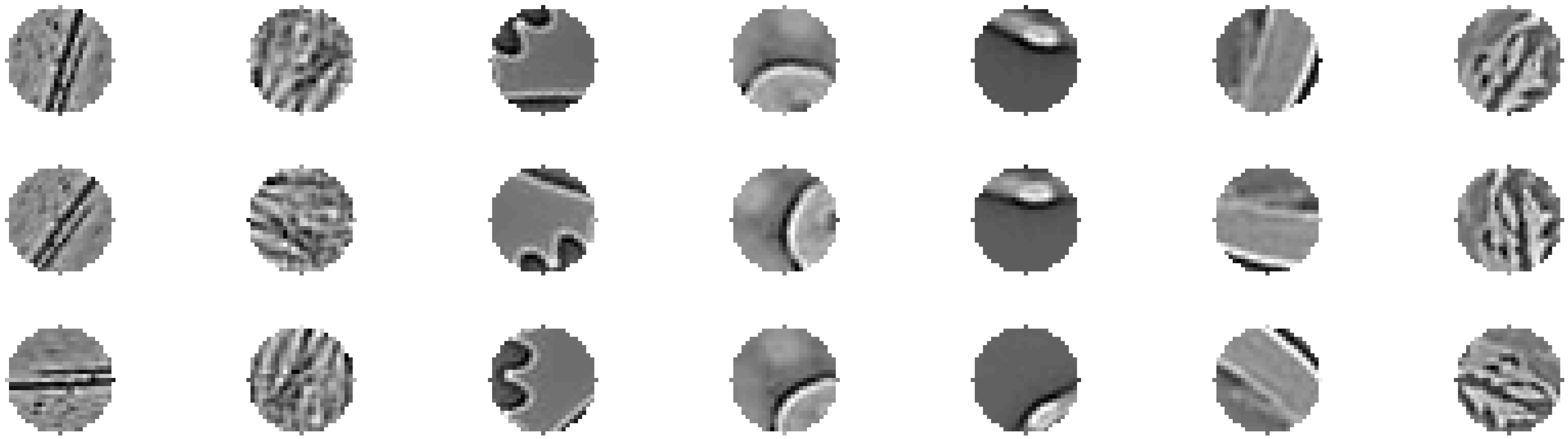
H built with semi-supervised BoostPro, on 200,000 examples; $M = 1400$

Results: pose retrieval



Recall for k-NN retrieval. For each value of k, the fraction of true k-NN w.r.t. pose within the k-NN w.r.t. an image-based similarity measure is plotted. Black dotted line: L_1 on EDH. Red dash-dot: PSH. Blue solid: BoostPro, M=1000.

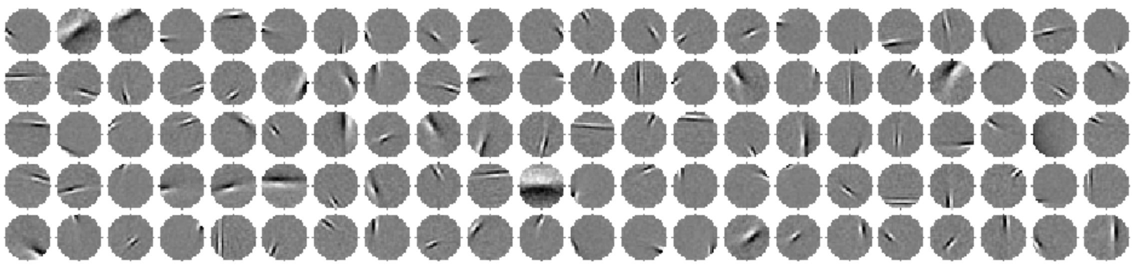
Visual similarity of image patches



- Define two patches to be similar under any rotation and mild shift ($\pm$ 4 pixels).
- Should be covered by many "reasonable" similarity definitions.

Descriptor 1: sparse overcomplete code

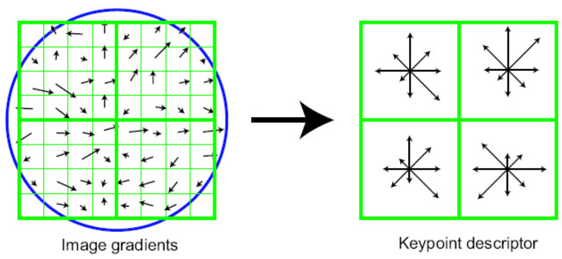
- Generative model of patches [Olshausen&Field]



- Very unstable under transformation, hence L_1 is not a good proxy for similarity.
- With BoostPro, improvement in area under ROC from 0.56 to 0.68.

Descriptor 2: SIFT

- Scale-Invariant Feature Transform [Lowe]

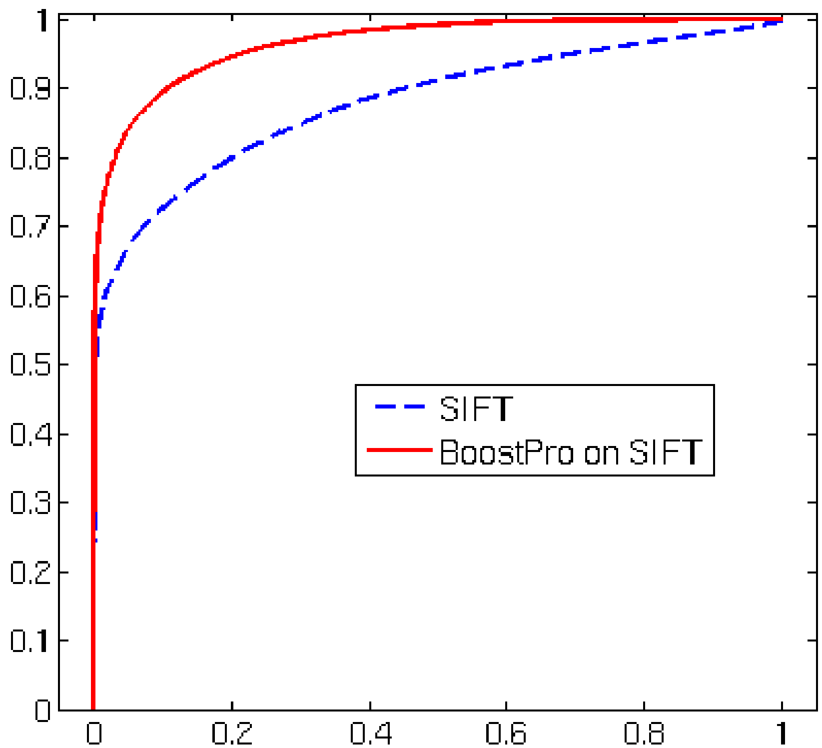


- Histogram of gradient magnitude and orientation within the region, normalized by orientation and at an appropriate scale.

- Discriminative (can not generate a patch).

- Designed specifically to make two similar patches match.

Results



- Area under ROC curve:

L_1 on SIFT	L_1 on H(SIFT)
0.8794	0.9633

Conclusions

- It is beneficial to learn similarity directly for the task rather than rely on the "default" distance.
- Key property of our approach: similarity search reduced to L_1 neighbor search (thus can be done in sublinear time.)
- Most useful for:
  - Regression and multi-label classification;
  - When large amounts of labeled data are available;
  - When L_p distance is not a good proxy for similarity.

Open questions

- What kinds of similarity concepts can we learn?
- How do we explore the space of projections more efficiently?
- Factorization of similarity.
- Combining unsupervised and supervised similarity learning for image regions.

Questions ?..