BoostMap: A Method for Efficient Approximate Similarity Rankings

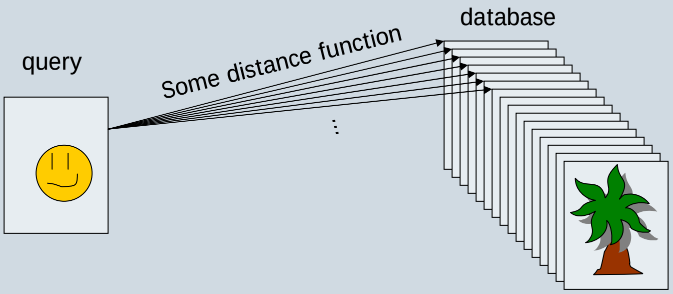
Vassilis Athitsos, Jonathan Alon,
Stan Sclaroff, and George Kollios
CVPR'04

Presenter: Boris Babenko
CSE 252C, FALL2006

The Problem

- For any general recognition task, there is usually a database of labeled images
- When a novel image is seen, a distance is computed between this image and every image in the database.

The Problem: an illustration



The Problem

- The distance function can be anything! Can be non-metric, bizarre, etc.
- Each query requires *n* distance calculations for a database of size *n*.
- What if the distance function is very complicated and expensive computationally?

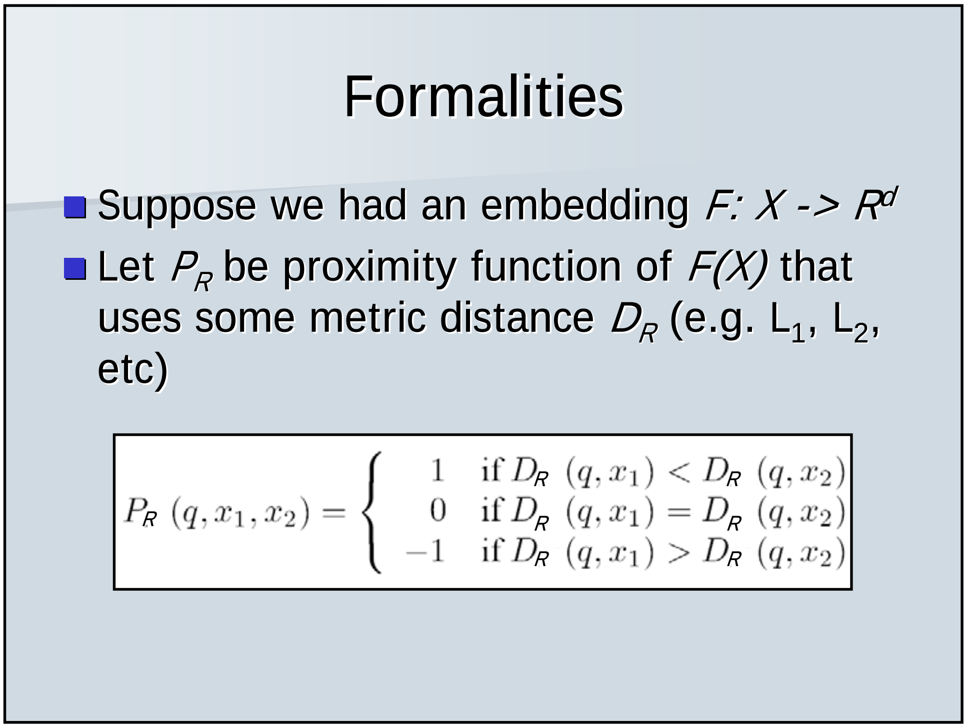
The Solution: BoostMap

- BoostMap is a method that can reduce the number of expensive distance calculations down to some $d << n$
- It works for ANY distance function

Formalities

- Let X be a set of objects, and $D_X(x1,x2)$ be a distance measure between objects of this set.
- Let $(q,x1,x2)$ be a triplet of objects from the set
- Define the Proximity Function $P_X(q,x1,x2)$

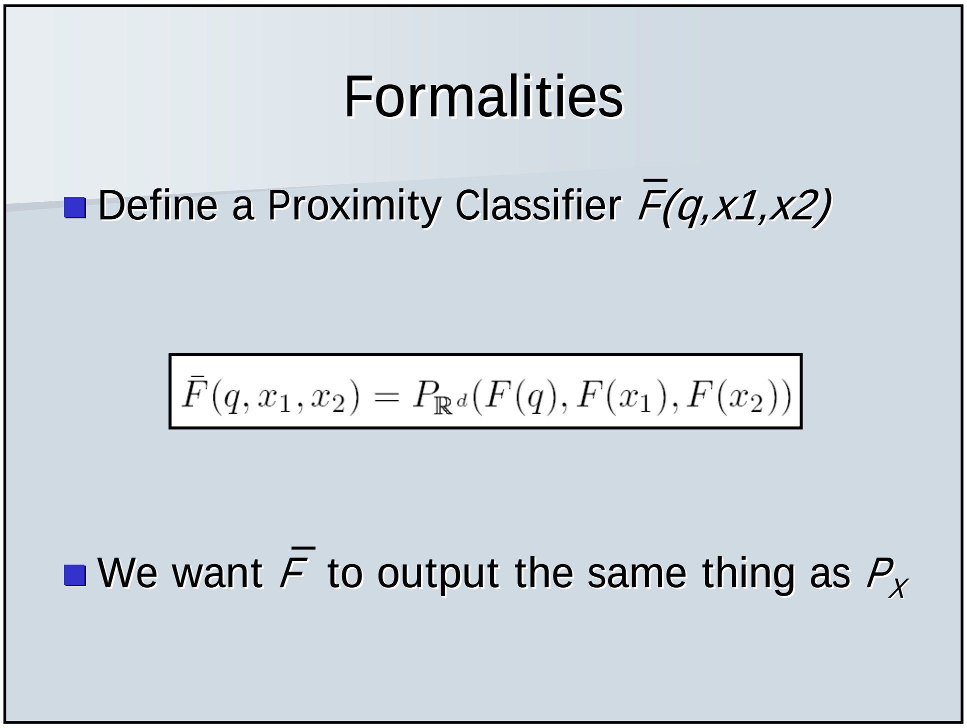
$$P_X(q,x_1,x_2) = \begin{cases} 1 & \text{if } D_X(q,x_1) < D_X(q,x_2) \\ 0 & \text{if } D_X(q,x_1) = D_X(q,x_2) \\ -1 & \text{if } D_X(q,x_1) > D_X(q,x_2) \end{cases}$$



Formalities

- Suppose we had an embedding $F: X \rightarrow R^d$
- Let P_R be proximity function of $F(X)$ that uses some metric distance D_R (e.g. L_1, L_2, etc)

$$P_R(q, x_1, x_2) = \begin{cases} 1 & \text{if } D_R(q, x_1) < D_R(q, x_2) \\ 0 & \text{if } D_R(q, x_1) = D_R(q, x_2) \\ -1 & \text{if } D_R(q, x_1) > D_R(q, x_2) \end{cases}$$



Formalities

- Define a Proximity Classifier $\bar{F}(q,x1,x2)$

$$\bar{F}(q, x_1, x_2) = P_{\mathbb{R}^d}(F(q), F(x_1), F(x_2))$$

- We want $\bar{F}$ to output the same thing as P_X

Computing Error

- For a single triple *(q,x1,x2)*

$$G(\bar{F}, q, x_1, x_2) = \frac{|P_X(q, x_1, x_2) - \bar{F}(q, x_1, x_2)|}{2}$$

- For all your data

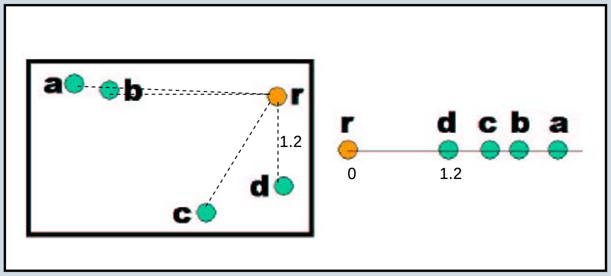
$$G(\bar{F}) = \frac{\sum_{(q,x_1,x_2)\in X^3} G(\bar{F}, q, x_1, x_2)}{|X|^3} \, .$$

How do we get the embedding F?

- Let’s think about simpler embeddings *F: X -> R*
- Generate many random simple embeddings and throw them into AdaBoost
- Our final embedding will be a linear combination of the simple embeddings

1D Embeddings

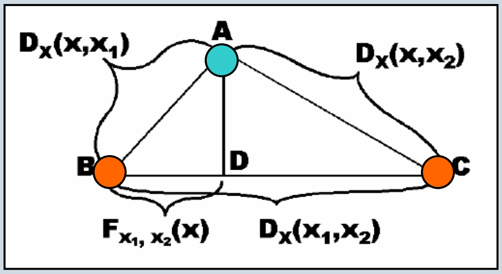
- Use a reference object r



- Classifies 46 out of 60 triplets correct. Incorrect: (b, a, c); (c, b, d); (d, b, r)

1D Embeddings

- Use “pivot points”



Boost 1D embedding

- How many people are not familiar with boosting?
- Use training data (which can be generated by using the original distance function D_x)
- AdaBoost outputs a set of *d* 1D embeddings, and a weight for each.

Final BoostMap Embedding

- Weghted L1 distance that combines the chosen 1D embeddings and their weights.
- Suppose we chose *d* embeddings. To compute the embedded distance between X_u and X_v:

$$D_{\mathbb{R}^d}((u_1, ..., u_d), (v_1, ..., v_d)) = \sum_{j=1}^{d} (\alpha_j |u_j - v_j|) \, .$$

What do we end up with?

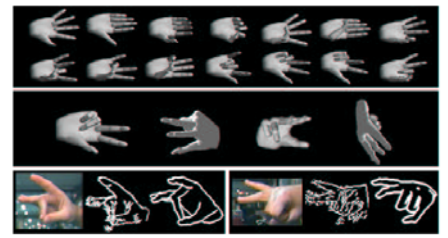
- An embedding *F: X -> R^d* which uses up to *2d* reference objects.
- A weighted L1 metric in this R^d space.
- We know that the embedding in some sense preserves the proximity.

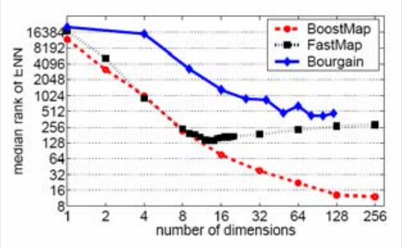
At Run-time

- Suppose we want to compare object *Q* (query) to objects *X1, X2... Xn* in the DB.
- Need to compute *d* embeddings of *Q:* O(d) calls to D_x
- Compute weighted L1 distance between *Q* and *X1, X2... Xn* – much cheaper than computing D_x n times.

Does it work?

- Hand experiment

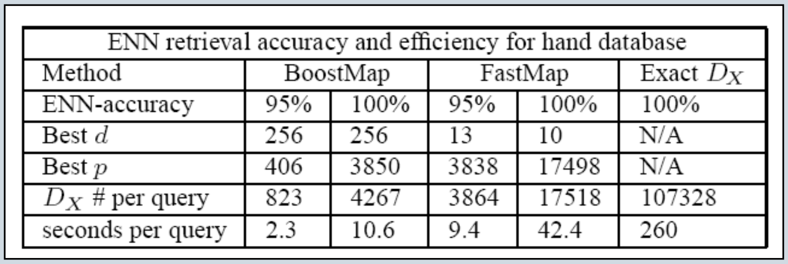




- Original distance measure: Chamfer distance (takes 260s to query)

Does it work?

- Hand experiment

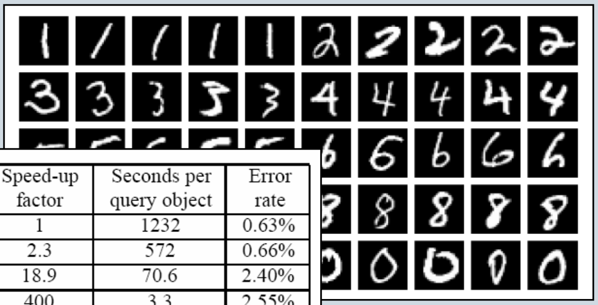


ENN retrieval accuracy and efficiency for hand database

Method	BoostMap		FastMap		Exact D_X
ENN-accuracy	95%	100%	95%	100%	100%
Best d	256	256	13	10	N/A
Best p	406	3850	3838	17498	N/A
D_X # per query	823	4267	3864	17518	107328
seconds per query	2.3	10.6	9.4	42.4	260

Does it work?

- Shape contexts



Method	Distances per query object	Speed-up factor	Seconds per query object	Error rate
brute force	20,000	1	1232	0.63%
vp-trees [24]	8594	2.3	572	0.66%
CNN [10]	1060	18.9	70.6	2.40%
Zhang [25]	50	400	3.3	2.55%
BoostMap	800	25	53.3	0.74%
BoostMap-C	800	25	53.3	0.72%
Cascade	149	134	9.9	0.75%
Cascade-C	92.5	216	6.2	0.74%

Questions?