# Dealing with the Curse of Dimensionality

CS 189

Alexei Efros

Fall 2015

# How to deal with curse of dimensionality?

Problem: $d > N$

Remedies:

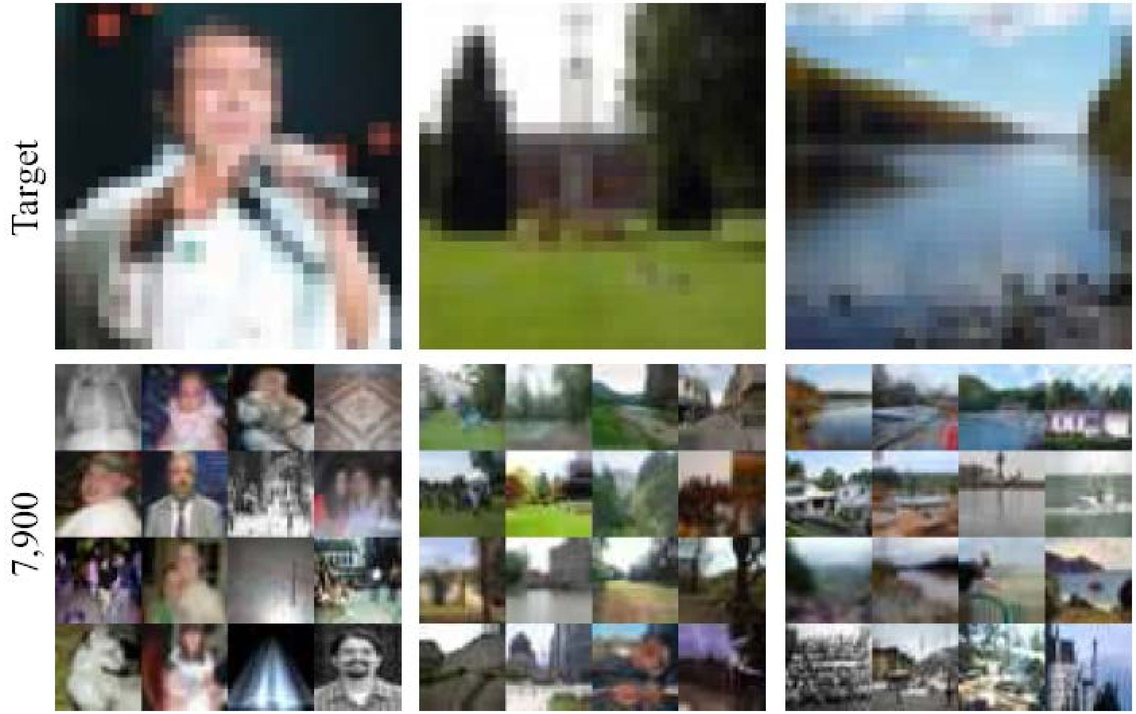
1. Get more data
   - Make N larger
2. Make better features
   - Make d smaller
3. Reason in lower "intrinsic dimension"
   - Be clever ☺. Effectively make d smaller
     - Dimensionality reduction (e.g. PCA)
     - Regularization
     - Better distance metric

# 1. Get More Data!

# Lots

# Of

# Images

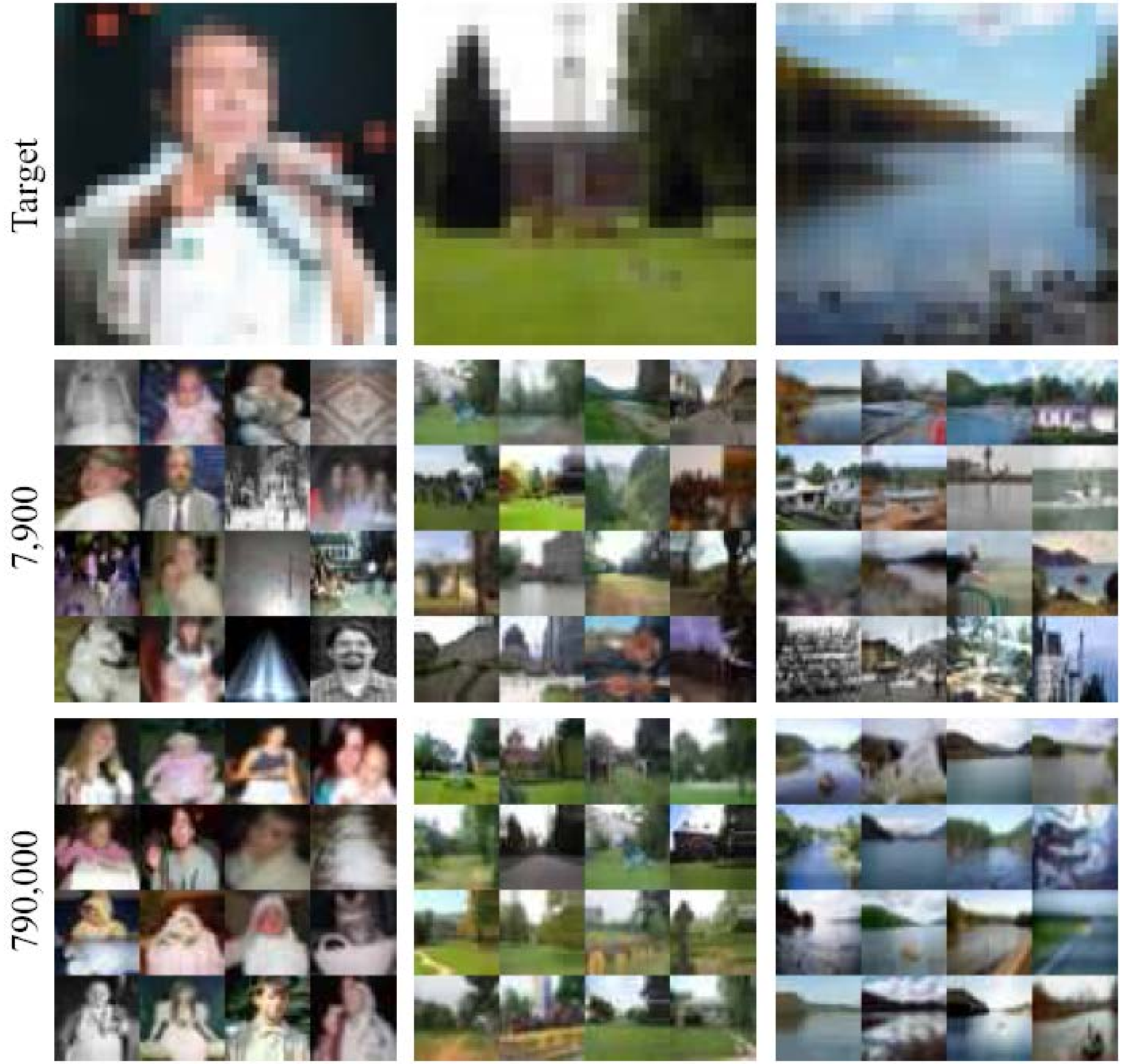



A. Torralba, R. Fergus, W.T.Freeman. PAMI 2008

# Lots

# Of

# Images

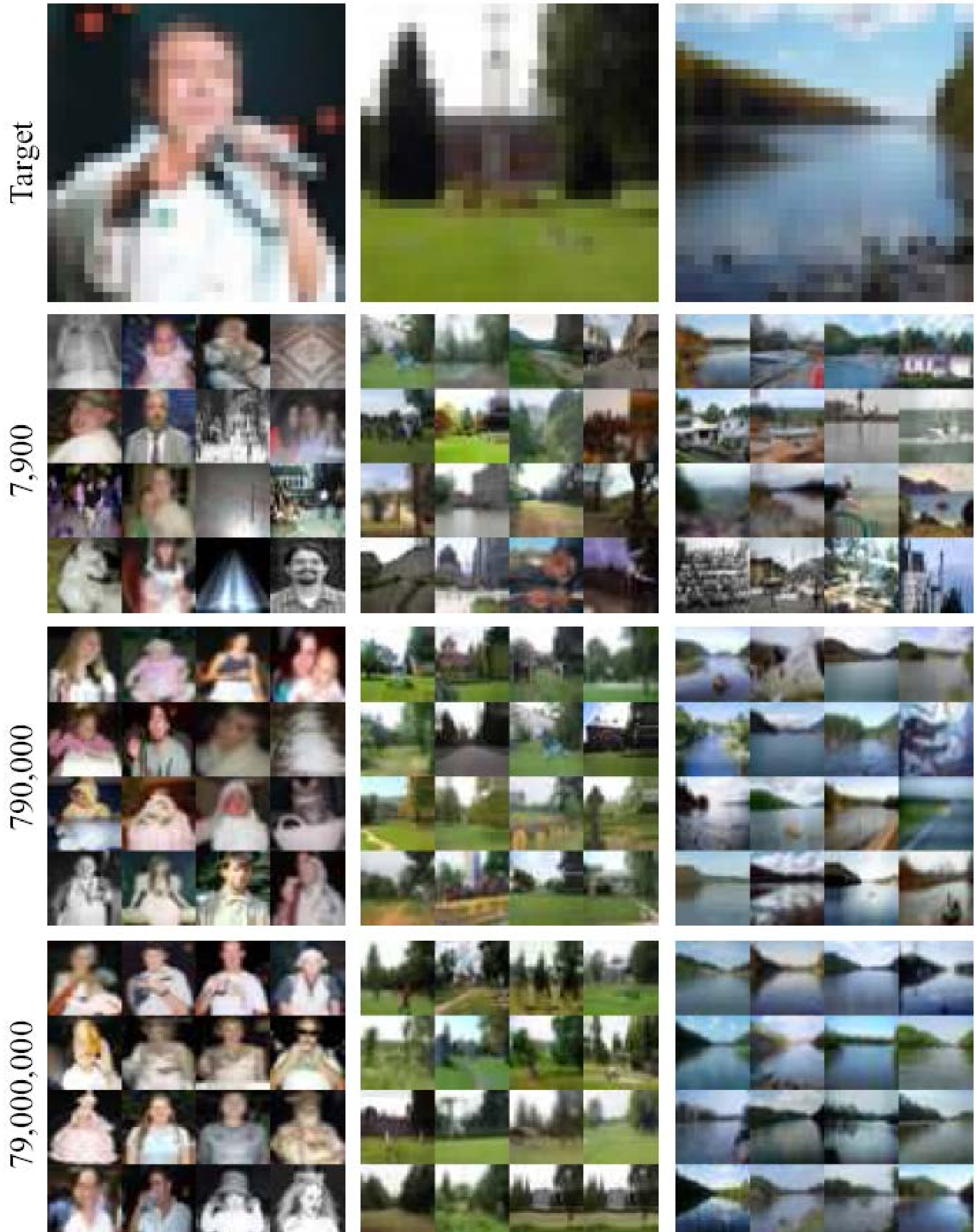



A. Torralba, R. Fergus, W.T.Freeman. PAMI 2008

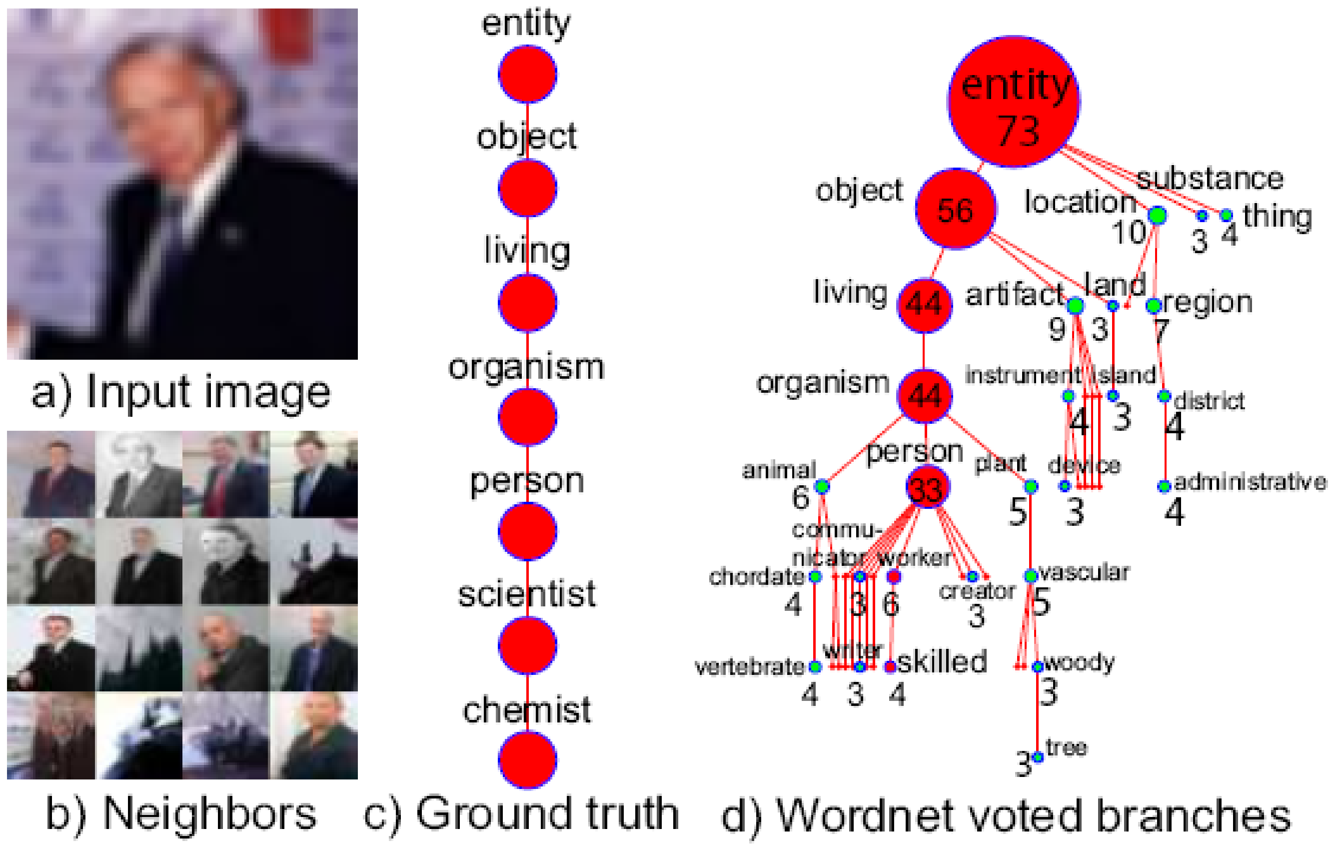
Lots

Of

Images



# 80 Million Tiny Images [PAMI'08]

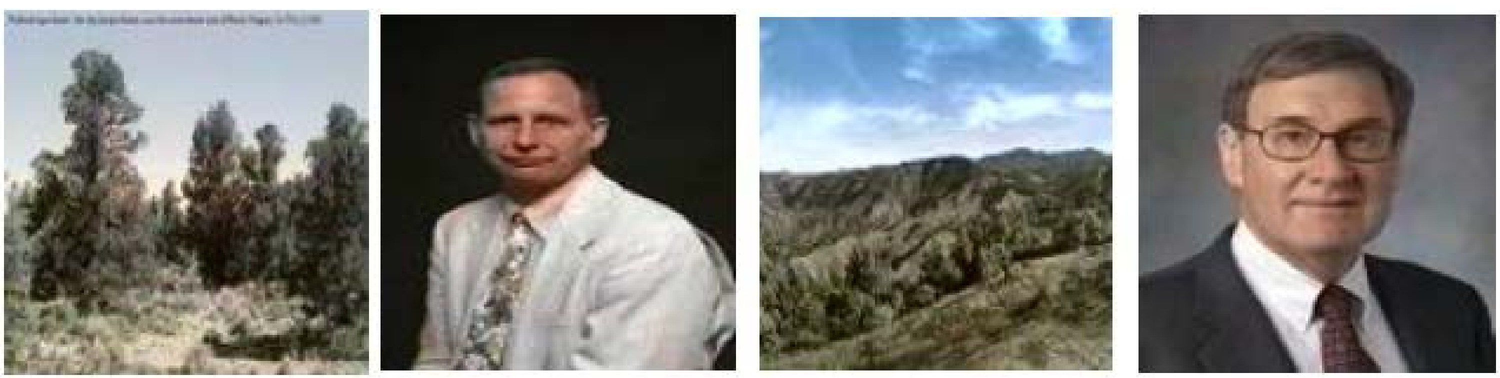



Torralba, Fergus, Freeman, PAMI 2008

# Automatic Colorization

Grayscale input High resolution

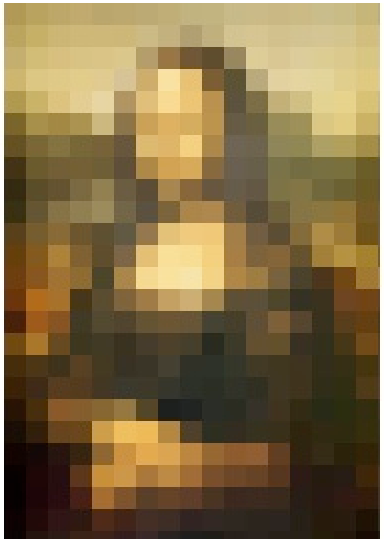
Colorization of input using average



A. Torralba, R. Fergus, W.T.Freeman. 2008

# 2. Make better features

- Reducing D
- E.g.



- What about text?

# Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

# Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary  Salton & McGill (1983)

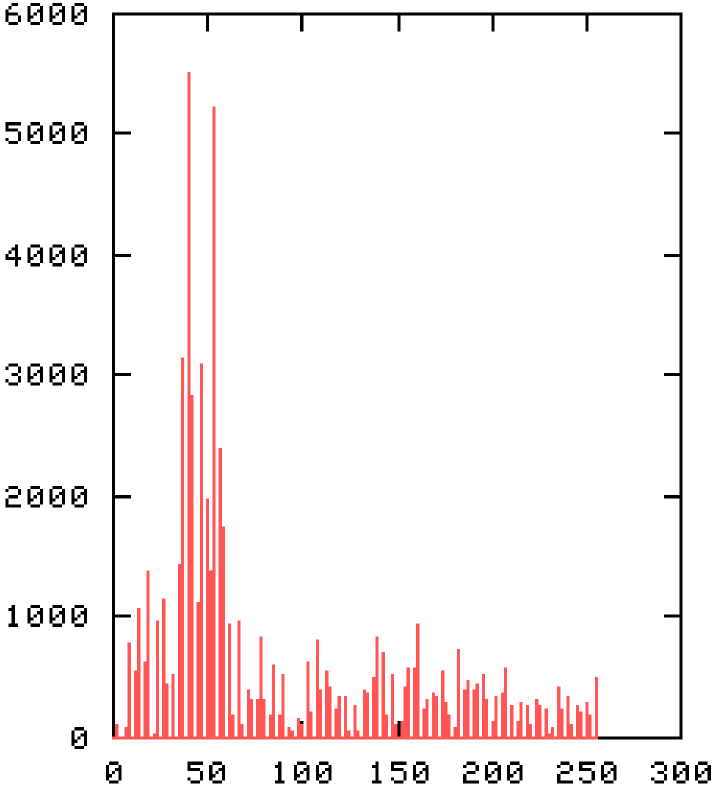
# Bag-of-words models

- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

# Some more intuition

- The information retrieval community had invented the “bag of words” model for text documents where we ignore the order of words and just consider their counts.  It turns out that this is quite an effective feature vector – medical documents  will use quite different words from real estate documents.
- An example with letters: How many different words can you think of that contain a, b, e, l, t?

# histogram



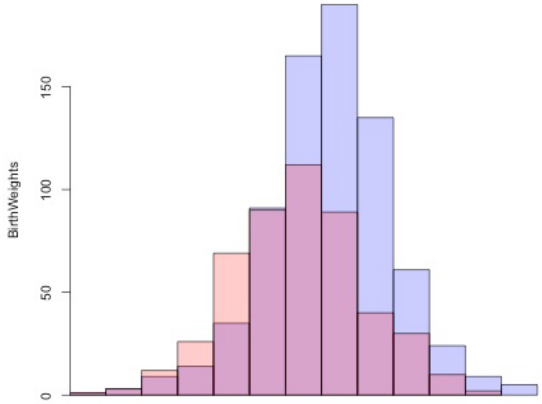
# One way to compare histograms is with the Intersection Kernel

Histogram Intersection kernel between histograms *a*, *b*

$$K(a,b) = \sum_{i=1}^{n} min(a_i, b_i)$$

$$a_i \geq 0$$
$$b_i \geq 0$$

*K* small -> *a*, *b* are different
*K* large -> *a*, *b* are similar

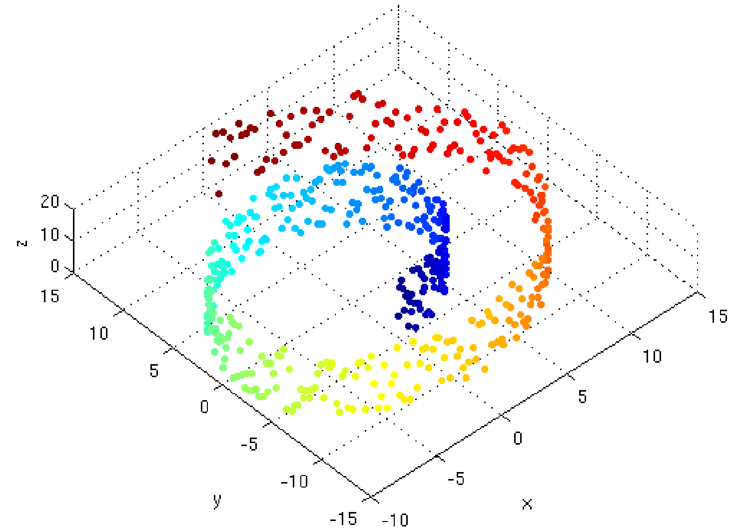



Odone et al 2005 proved positive definiteness.
Can be used directly as a kernel for an SVM.

# 3. Better distance

# Good News

- Two forms of dimensions:
  - Extrinsic (dimension of ambient space d)
  - Intrinsic
    - E.g. line in 3d
    - Swiss-roll



# How dense is the space of images?

Number of images on my hard drive: $10^4$

Number of images seen during my first 10 years: $10^8$

(3 images/second * 60 * 60 * 16 * 365 * 10 = 630720000)

Number of images seen by all humanity: $10^{20}$

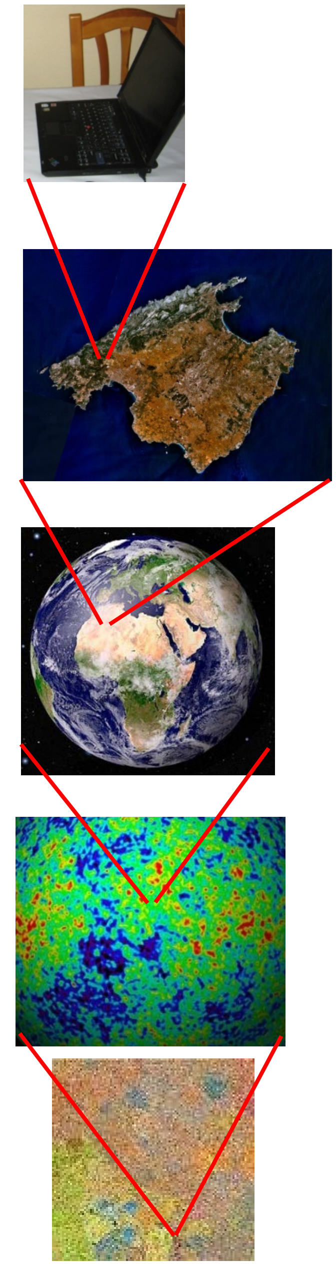
106,456,367,669 humans[1] * 60 years * 3 images/second * 60 * 60 * 16 * 365 =

1 from http://www.prb.org/Articles/2002/HowManyPeopleHaveEverLivedonEarth.aspx

Number of photons in the universe: $10^{88}$

Number of all 32x32 images: $10^{7373}$

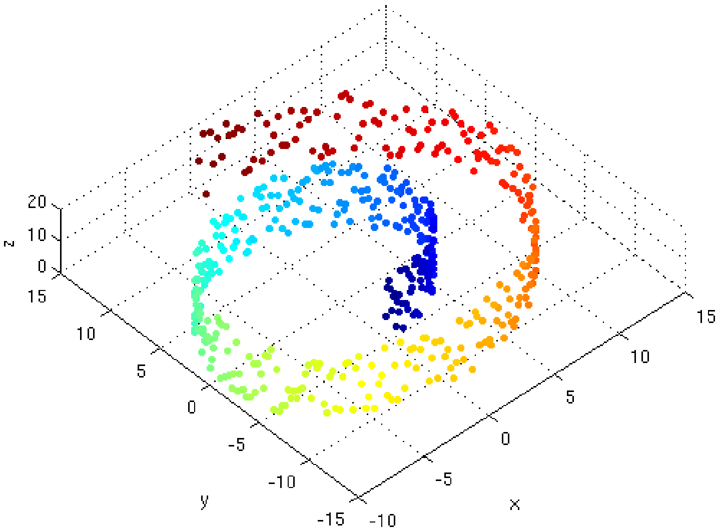
$256^{32*32*3} \sim 10^{7373}$



Slide by Antonio Torralba

# Good News

- Two forms of dimensions:
  - Extrinsic (dimension of ambient space d)
  - Intrinsic
    - E.g. line in 3d
    - Swiss-roll



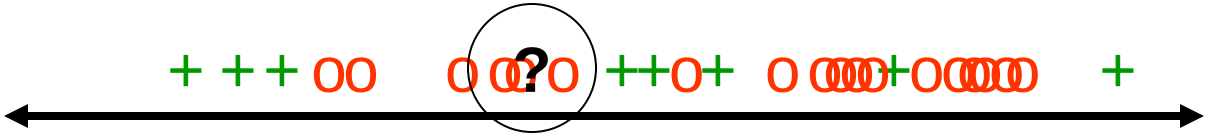

This is what can save us:

  - Terminology: k-surface in d-dimensional space, k<<d
  - (manifold)

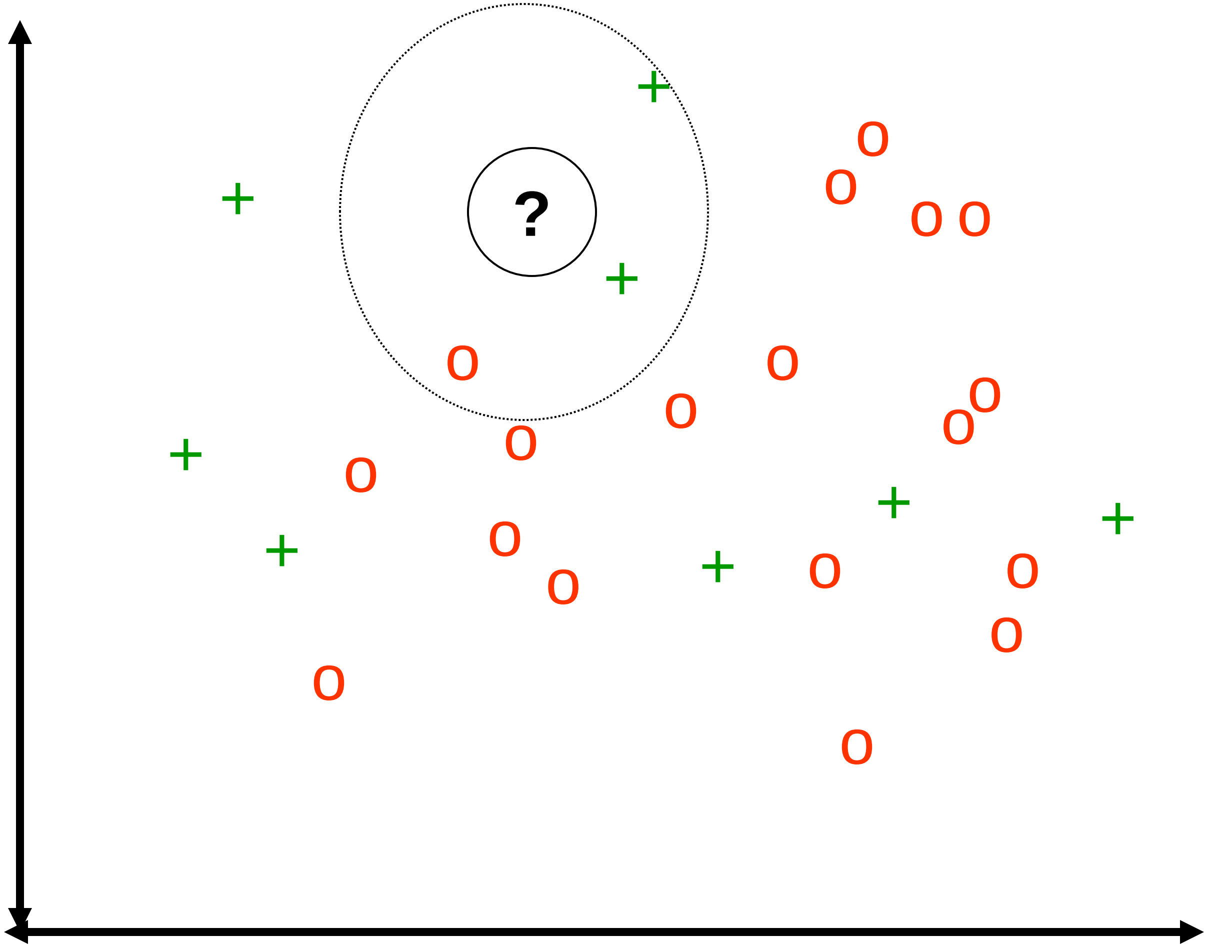
# Working in implicit dimension

1. Explicit
   - Feature selection
   - Dimensionality reduction (e.g. PCA)
2. Implicit
   - Regularization
   - better distance metric

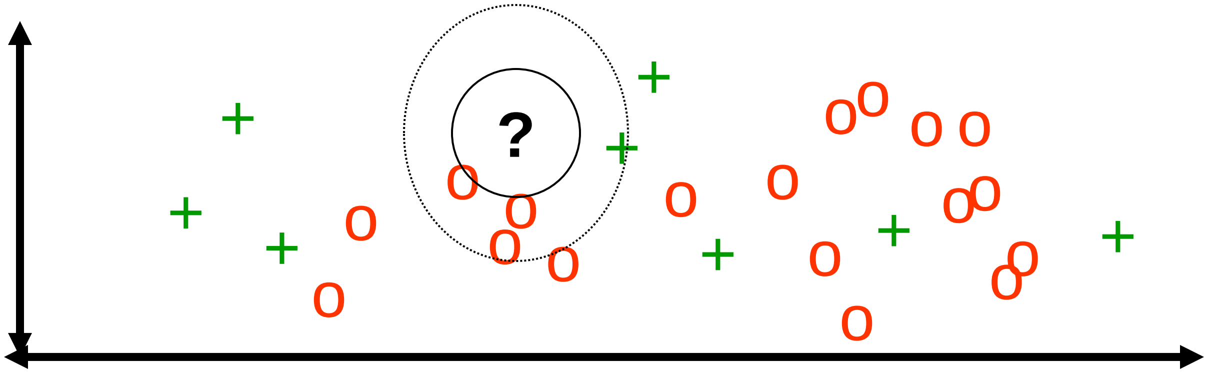
# K-NN and irrelevant features



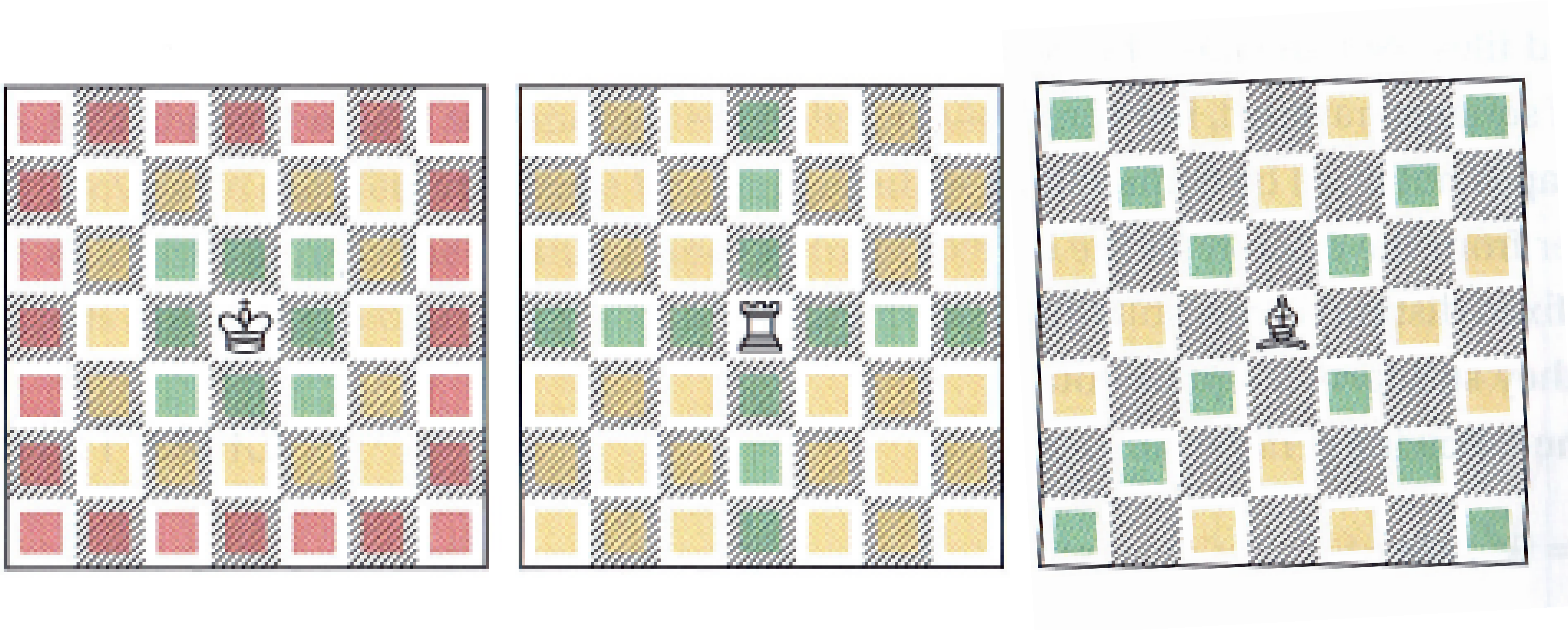
# K-NN and irrelevant features



# K-NN and irrelevant features



# Measuring distance



# Distance Metrics

If $\mathscr{X} = \mathbb{R}^d$, the *Minkowski distance* of order $p > 0$ is defined as

$$\mathrm{Dis}_p(\mathbf{x}, \mathbf{y}) = \left( \sum_{j=1}^{d} |x_j - y_j|^p \right)^{1/p} = ||\mathbf{x} - \mathbf{y}||_p$$

where $||\mathbf{z}||_p = \left( \sum_{j=1}^{d} |z_j|^p \right)^{1/p}$ is the *$p$-norm* (sometimes denoted $L_p$ norm) of the vector $\mathbf{z}$. We will often refer to $\mathrm{Dis}_p$ simply as the $p$-norm.

☞ The 1-*norm* denotes *Manhattan distance*, also called *cityblock distance*:

$$\mathrm{Dis}_1(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{d} |x_j - y_j|$$

This is the distance if we can only travel along coordinate axes.

☞ You will sometimes see references to the *0-norm* (or $L_0$ norm) which counts the number of non-zero elements in a vector. The corresponding distance then counts the number of positions in which vectors $\mathbf{x}$ and $\mathbf{y}$ differ. This is not strictly a Minkowski distance; however, we can define it as

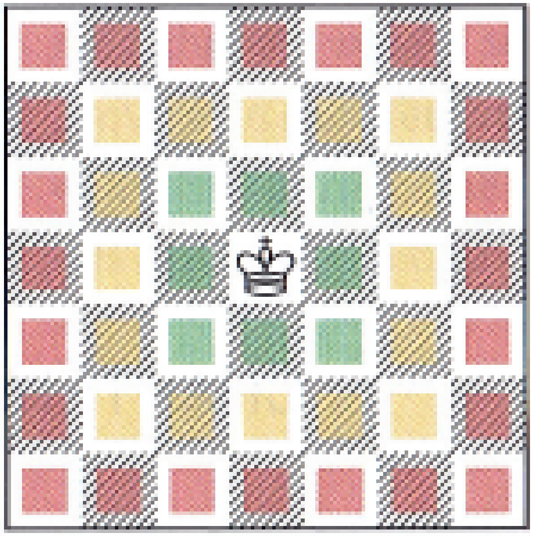
$$\mathrm{Dis}_0(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{d} (x_j - y_j)^0 = \sum_{j=1}^{d} I[x_j = y_j]$$

under the understanding that $x^0 = 0$ for $x = 0$ and $1$ otherwise.

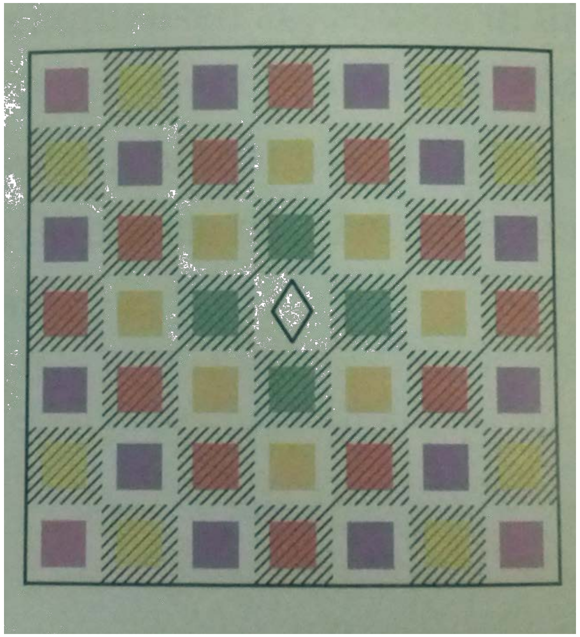
☞ If $\mathbf{x}$ and $\mathbf{y}$ are binary strings, this is also called the *Hamming distance*. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change $\mathbf{x}$ into $\mathbf{y}$.

☞ For non-binary strings of unequal length this can be generalised to the notion of *edit distance* or *Levenshtein distance*.

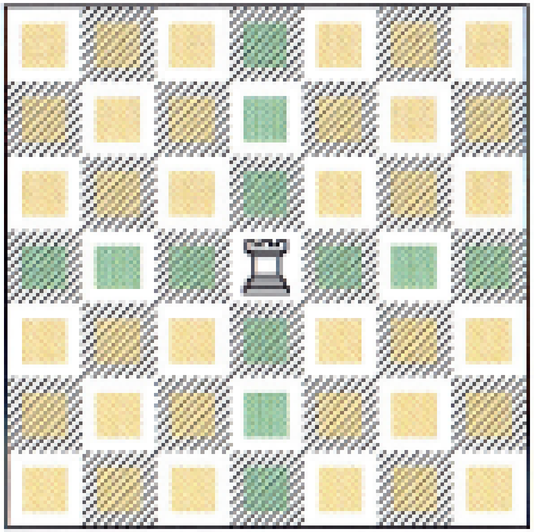
# Quiz: name that *distance*!



inf-norm

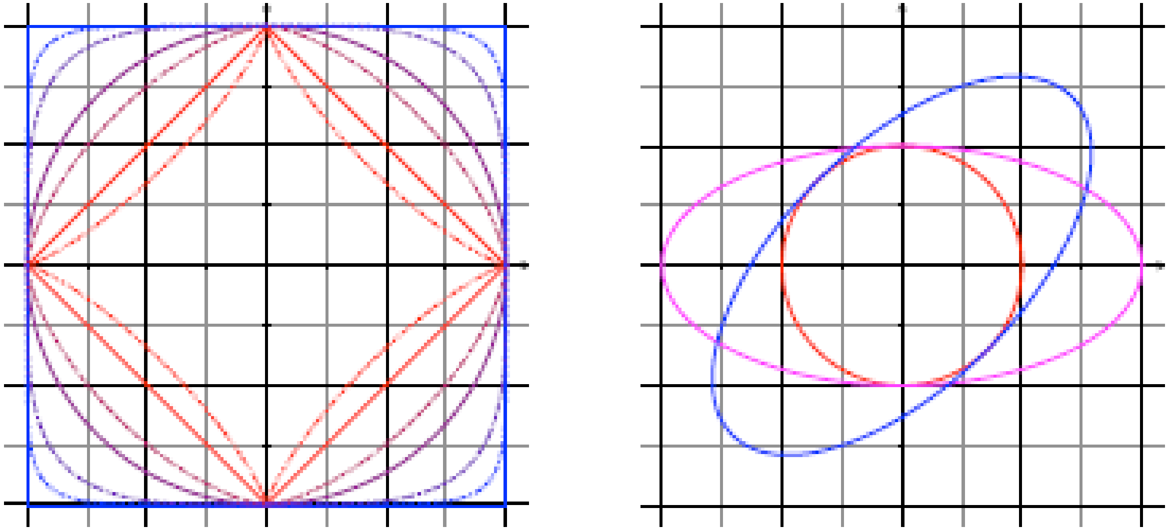


1-norm



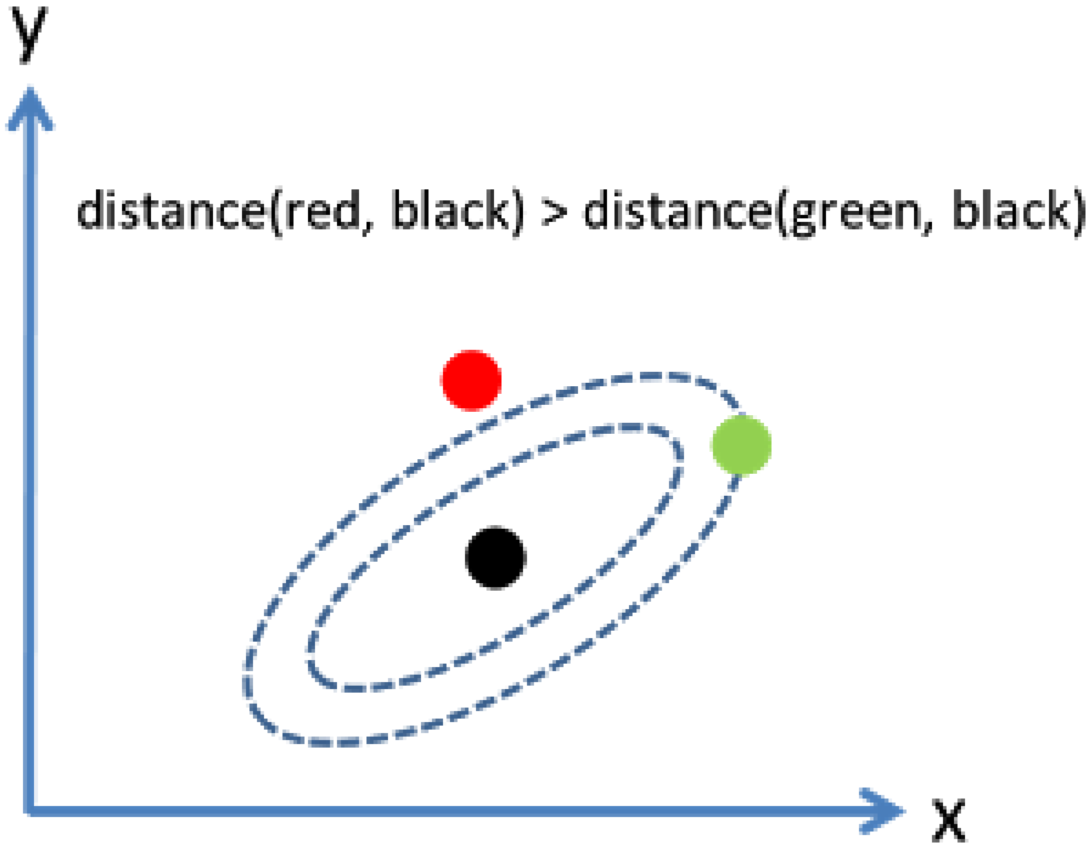
0-norm

# Visualizing distances



**(left)** Lines connecting points at order-$p$ Minkowski distance 1 from the origin for (from inside) $p = 0.8$; $p = 1$ (Manhattan distance, the rotated square in red); $p = 1.5$; $p = 2$ (Euclidean distance, the violet circle); $p = 4$; $p = 8$; and $p = \infty$ (Chebyshev distance, the blue rectangle). Notice that for points on the coordinate axes all distances agree. For the other points, our reach increases with $p$; however, if we require a rotation-invariant distance metric then Euclidean distance is our only choice. **(right)** The rotated ellipse $\mathbf{x}^T\mathbf{R}^T\mathbf{S}^2\mathbf{R}\mathbf{x} = 1/4$; the axis-parallel ellipse $\mathbf{x}^T\mathbf{S}^2\mathbf{x} = 1/4$; and the circle $\mathbf{x}^T\mathbf{x} = 1/4$.

# Adjusting to the dataset…



# Mahalanobis Distance

Often, the shape of the ellipse is estimated from data as the inverse of the covariance matrix: $\mathbf{M} = \mathbf{\Sigma}^{-1}$. This leads to the definition of the *Mahalanobis distance*

$$\mathrm{Dis}_M(\mathbf{x}, \mathbf{y}|\mathbf{\Sigma}) = \sqrt{(\mathbf{x} - \mathbf{y})^{\mathrm{T}} \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{y})}$$

Using the covariance matrix in this way has the effect of decorrelating and normalising the features.

Clearly, Euclidean distance is a special case of Mahalanobis distance with the identity matrix $\mathbf{I}$ as covariance matrix: $\mathrm{Dis}_2(\mathbf{x}, \mathbf{y}) = \mathrm{Dis}_M(\mathbf{x}, \mathbf{y}|\mathbf{I})$.

# Distance metric

Given an instance space $\mathcal{X}$, a *distance metric* is a function $\mathrm{Dis}: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that for any $x, y, z \in \mathcal{X}$:

- distances between a point and itself are zero: $\mathrm{Dis}(x,x) = 0$;
- all other distances are larger than zero: if $x \neq y$ then $\mathrm{Dis}(x,y) > 0$;
- distances are symmetric: $\mathrm{Dis}(y,x) = \mathrm{Dis}(x,y)$;
- detours can not shorten the distance: $\mathrm{Dis}(x,z) \leq \mathrm{Dis}(x,y) + \mathrm{Dis}(y,z)$.

# Cosine similarity

. Since the dot product can be written as $||\mathbf{x}|| \cdot ||\mathbf{y}|| \cos\theta$, where $\theta$ is the angle between the vectors $\mathbf{x}$ and $\mathbf{y}$, we define the *cosine similarity* as

$$\cos\theta = \frac{\mathbf{x}\cdot\mathbf{y}}{||\mathbf{x}|| \cdot ||\mathbf{y}||} = \frac{\mathbf{x}\cdot\mathbf{y}}{\sqrt{(\mathbf{x}\cdot\mathbf{x})(\mathbf{y}\cdot\mathbf{y})}}$$

Cosine similarity differs from Euclidean distance in that it doesn't depend on the length of the vectors $\mathbf{x}$ and $\mathbf{y}$.

On the other hand, it is not translation-independent, but assigns special status to the origin: one way to think of it is as a projection onto a unit sphere around the origin, and measuring distance on that sphere. Cosine similarity is usually turned into a distance metric by taking $1-\cos\theta$.

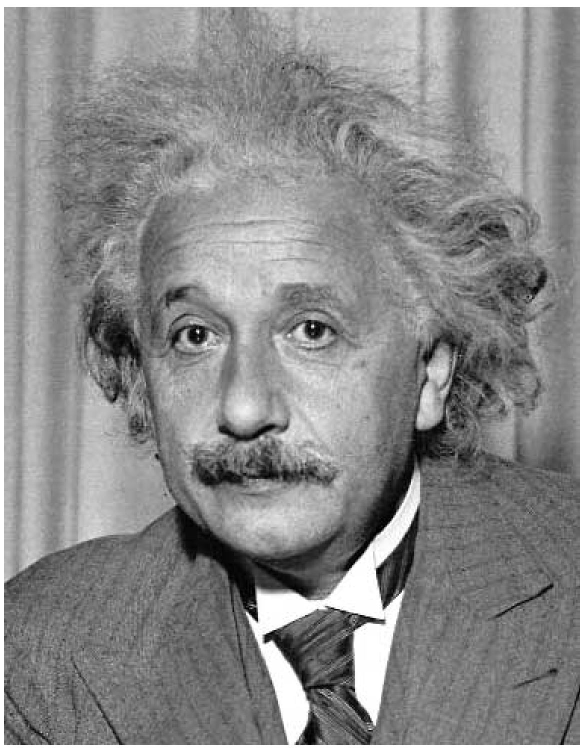
# Similarities (1-distance)

| Function | Equation | Symmetric? | Output range | Invariant to shift in input? | Pithy explanation in terms of something else |
|---|---|---|---|---|---|
| Inner(x,y) | $\langle x, y \rangle$ | Yes | $\mathbb{R}$ | No | |
| CosSim(x,y) | $\frac{\langle x, y \rangle}{\|\|x\|\| \ \|\|y\|\|}$ | Yes | [-1,1] or [0,1] if inputs non-neg | No | normalized inner product |
| Corr(x,y) | $\frac{\langle x - \bar{x},\ y - \bar{y} \rangle}{\|\|x - \bar{x}\|\| \ \|\|y - \bar{y}\|\|}$ | Yes | [-1,1] | Yes | centered cosine; *or* normalized covariance |

# Template matching

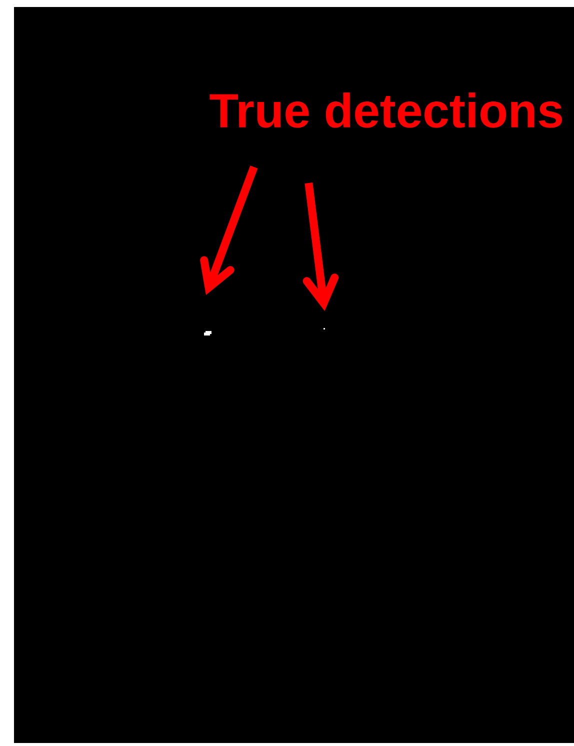
- Goal: find  in image
- Method 1: L2

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$



Input

1- sqrt(SSD)



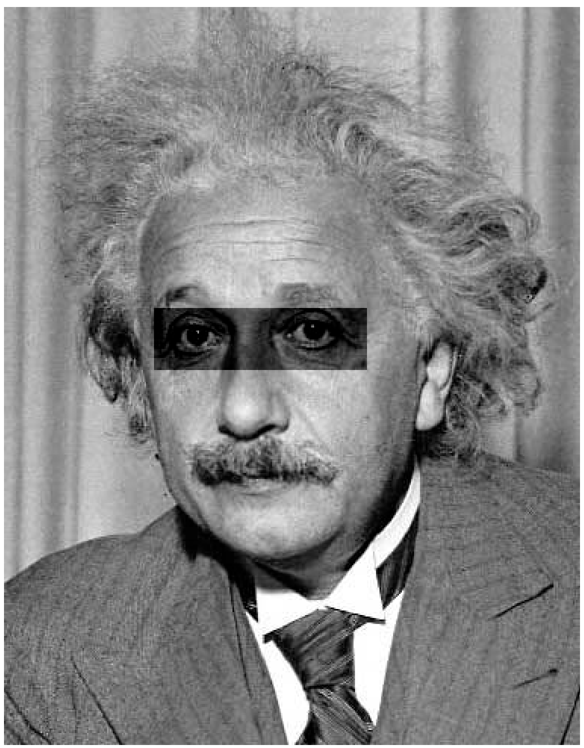

Thresholded Image

# Template Matching

- Goal: find  in image
- Method 1: L2

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$

**What's the potential downside of L2?**

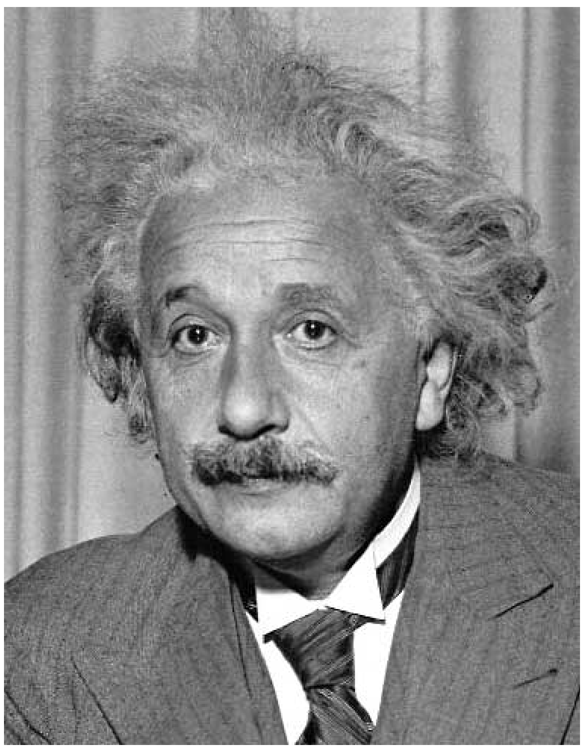


Input

1- sqrt(SSD)

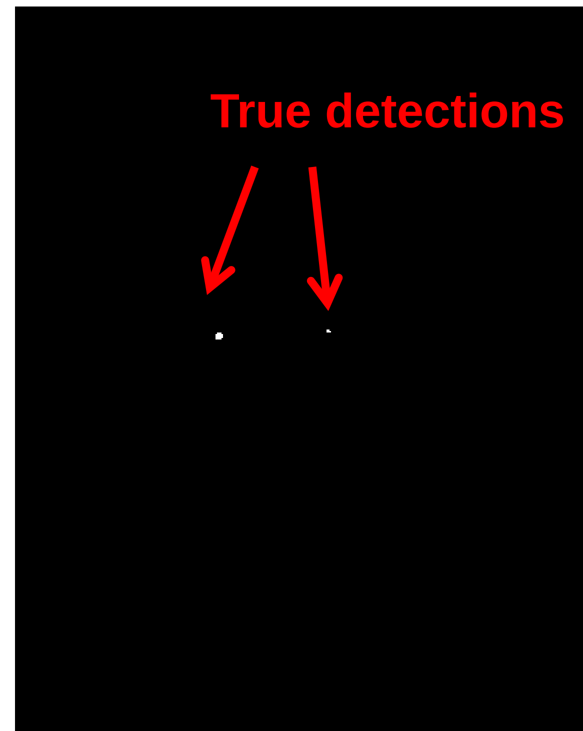
# Matching with filters

- Goal: find  in image
- Method 2: Normalized cross-correlation



Input

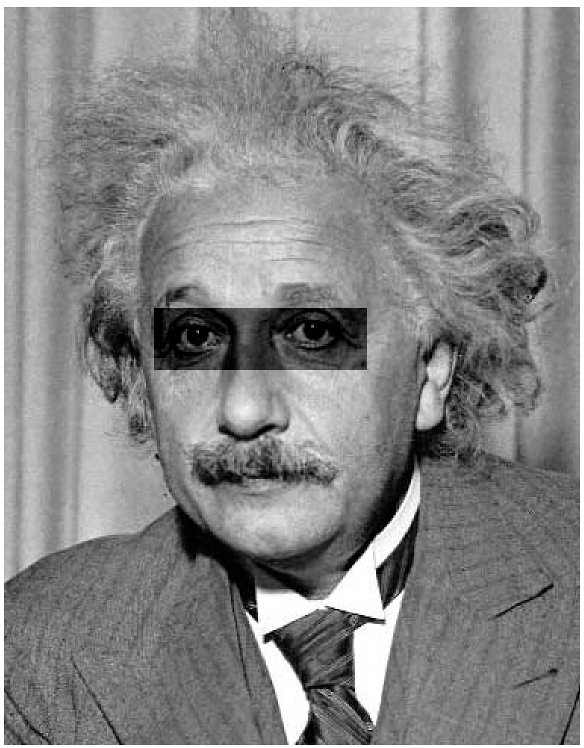
Normalized X-Correlation




Thresholded Image

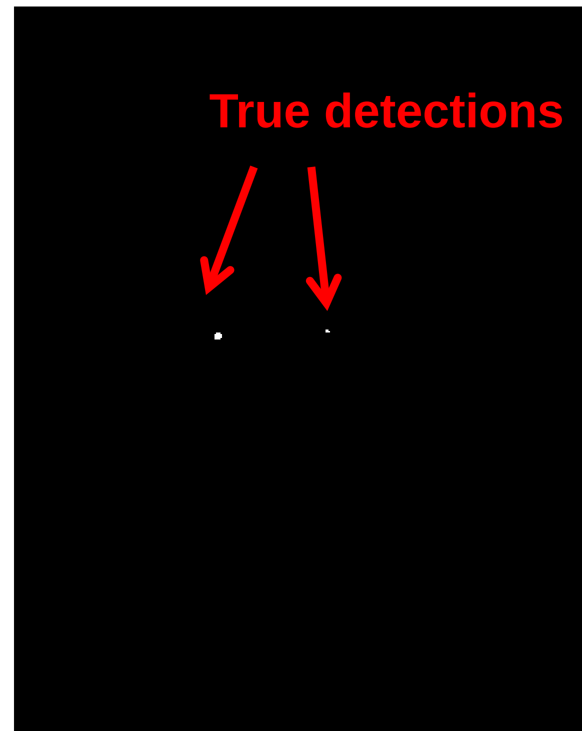
# Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation



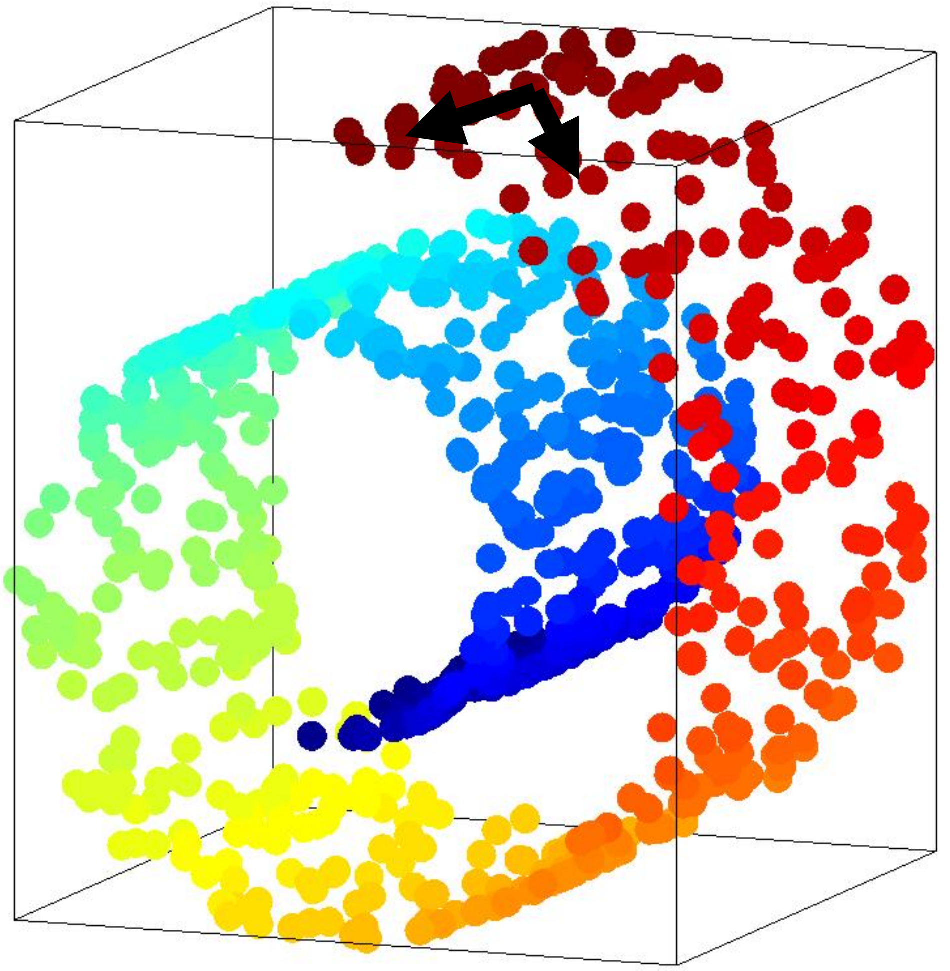
Input

Normalized X-Correlation

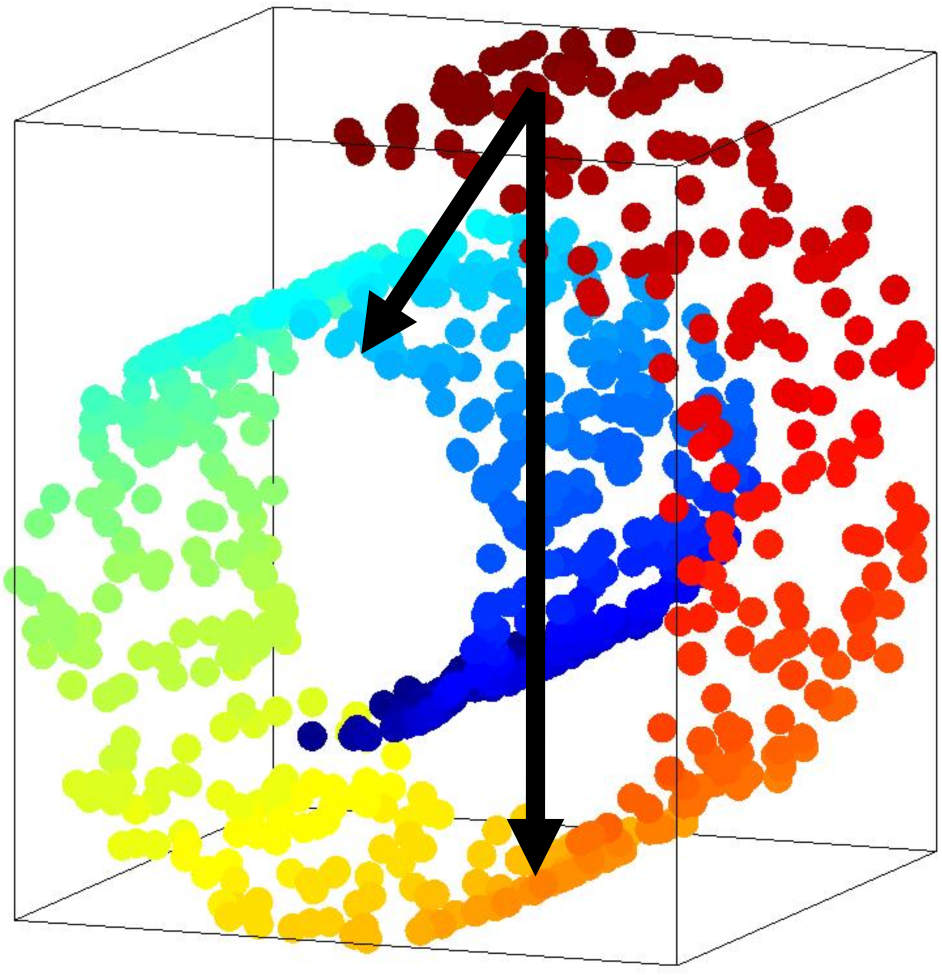



Thresholded Image

# Non-linear manifolds



# Non-linear manifolds

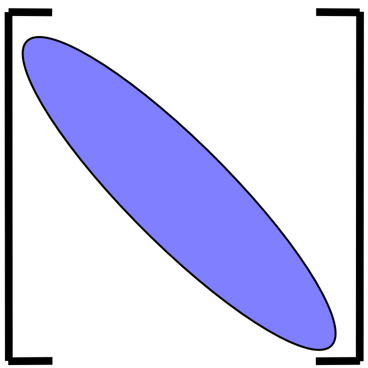


# Isomap for images

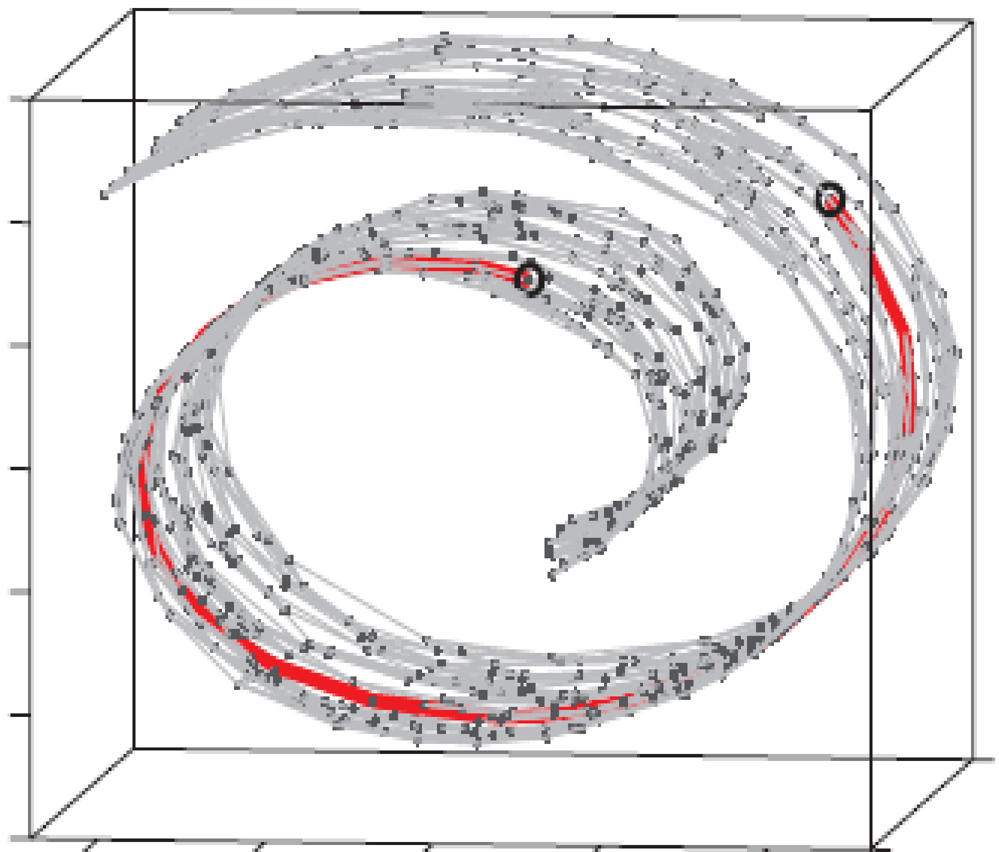
- Build a data graph G.
- Vertices: images
- (u,v) is an edge iff dist(u,v) is small
- For any two data points, we approximate the distance between them with the “shortest path” on G

# Isomap

1. Build a sparse graph with K-nearest neighbors

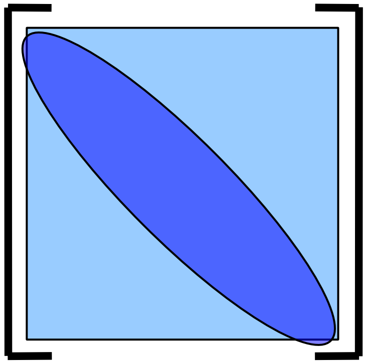
$D_g =$ 

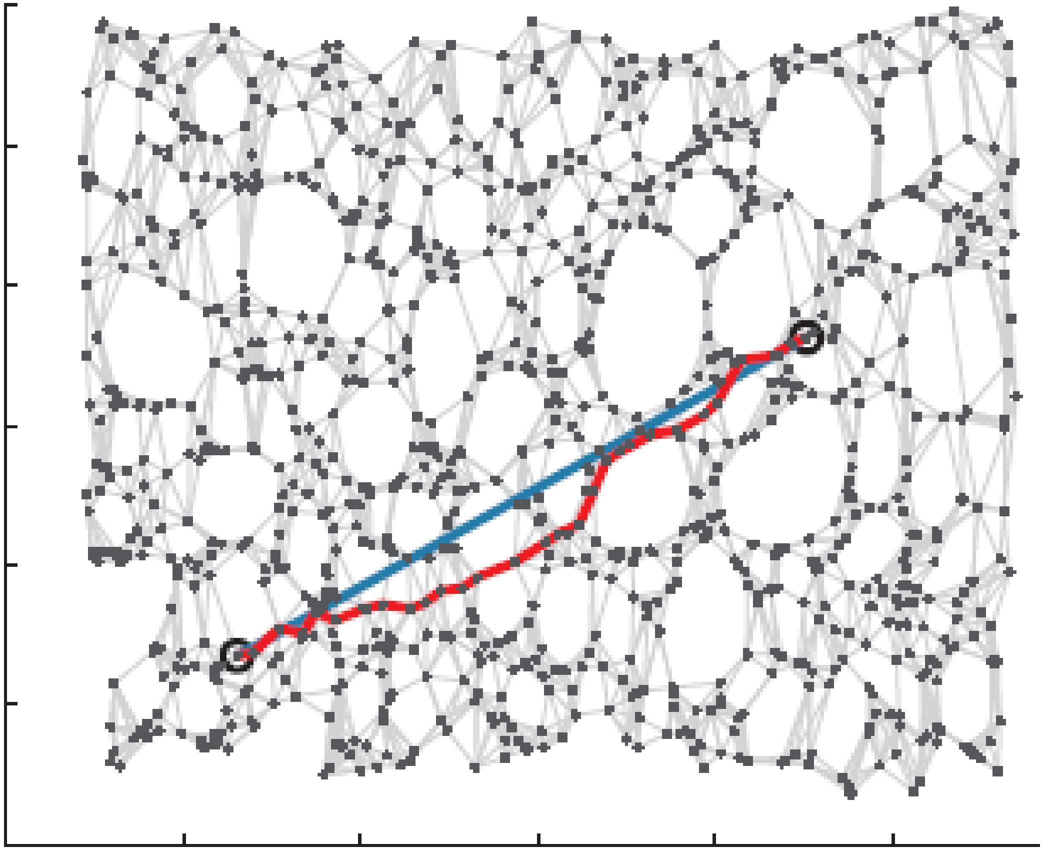
(distance matrix is sparse)



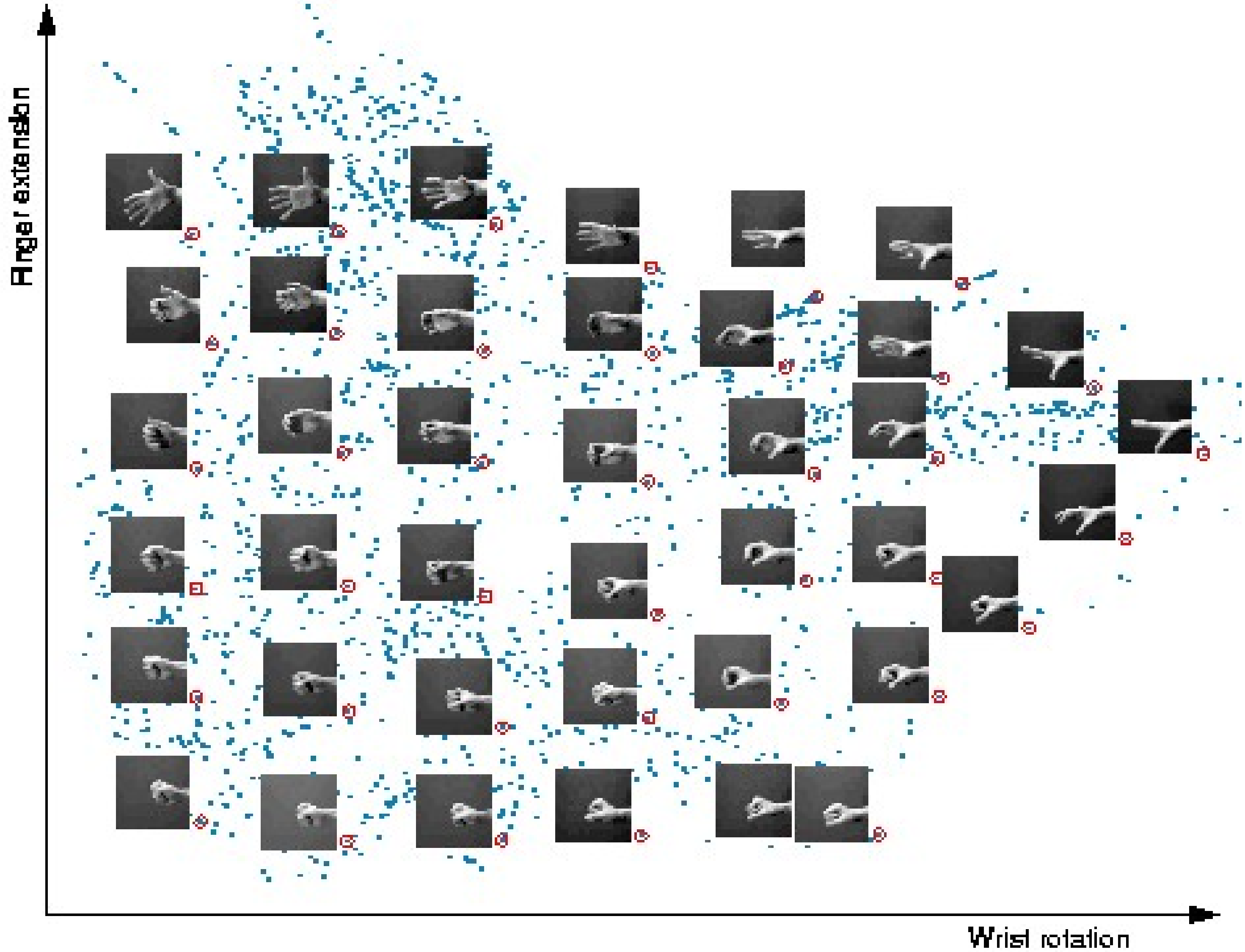
# Isomap

2. Infer other interpoint distances by finding shortest paths on the graph (Dijkstra's algorithm).

$D_g =$ 



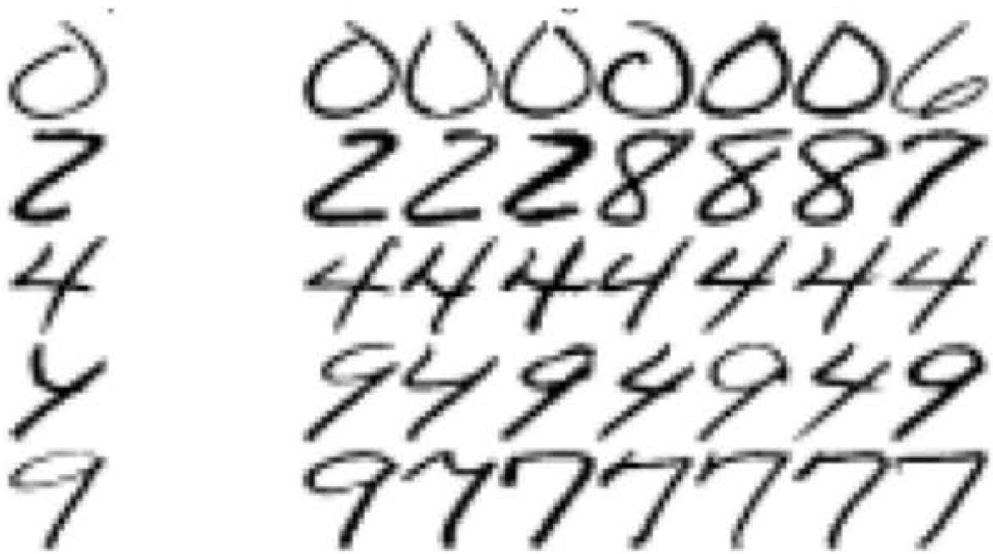
# Isomap results: hands



# Examples

# Example: Digits

- Examples:



# Nearest Neighbor Classification
# Why Euclidean distance is not very good

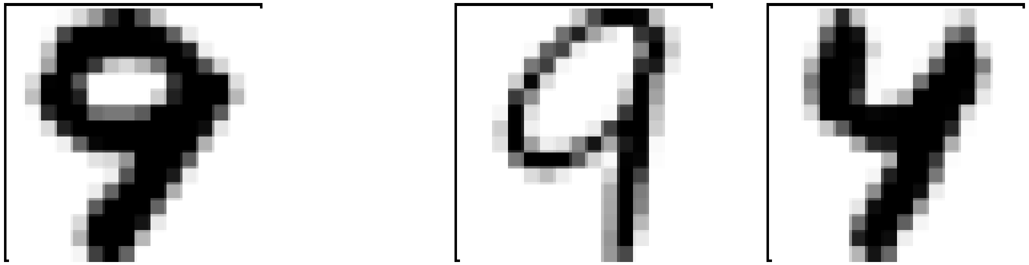
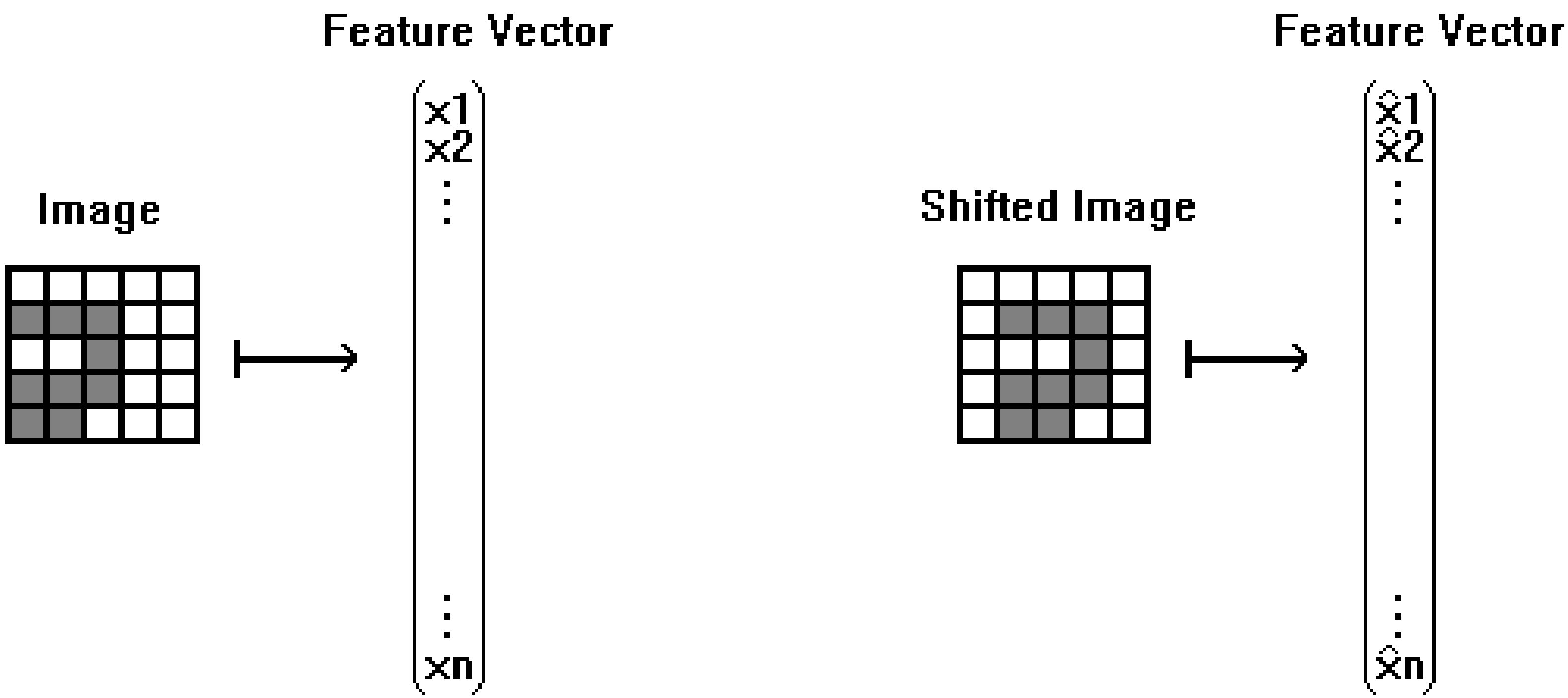


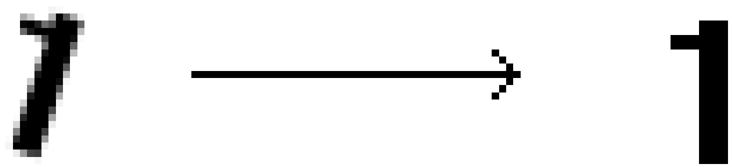
**Fig. 1.** According to the Euclidean distance the pattern to be classified is more similar to prototype B. A better distance measure would find that prototype A is closer because it differs mainly by a rotation and a thickness transformation, two transformations which should leave the classification invariant.

# Why is this a problem?



# Transformation invariance

- We want to recognize objects in spite of various transformations-scaling, translation, rotations, small deformations…



of course,  sometimes we don’t want full invariance – a 6 vs. a 9

# How do we build in transformational invariance?

- Augment the dataset (make N bigger)
  - Include in it various transformed copies of the digit, and hope that the classifier will figure out a decision boundary that works
- Build in invariance into the feature vector
  - Orientation histograms do this for several common transformations and this is why they are so popular for building feature vectors in computer vision
- Build in invariance into the classification strategy
  - E.g. Tangent Distance, convolutional neural networks, etc

# Orientation histograms

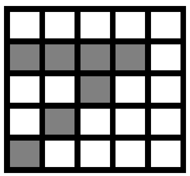


Image 1

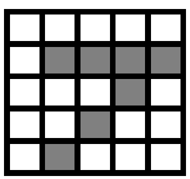
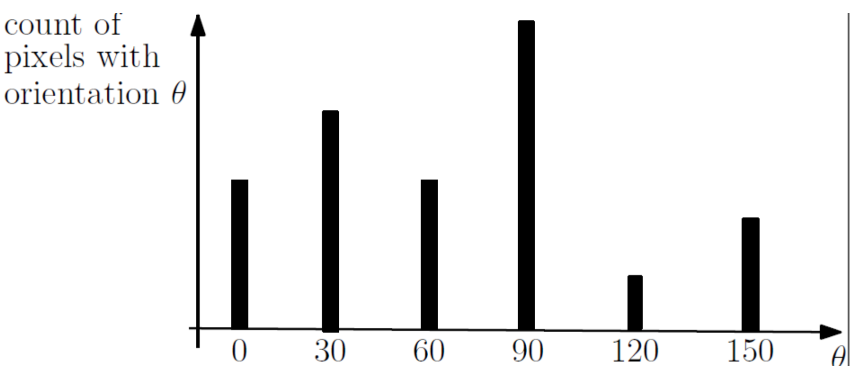
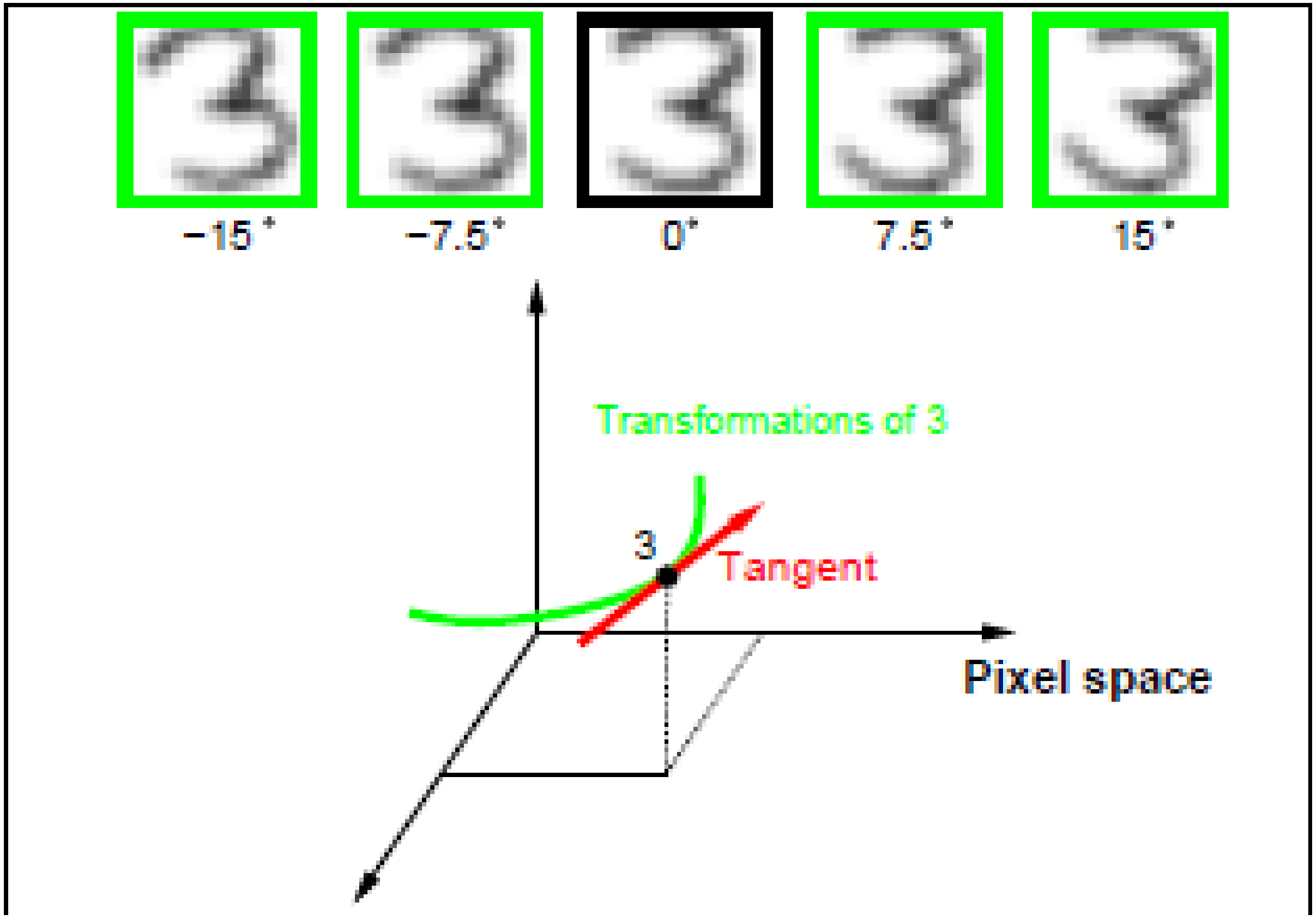


Image 2



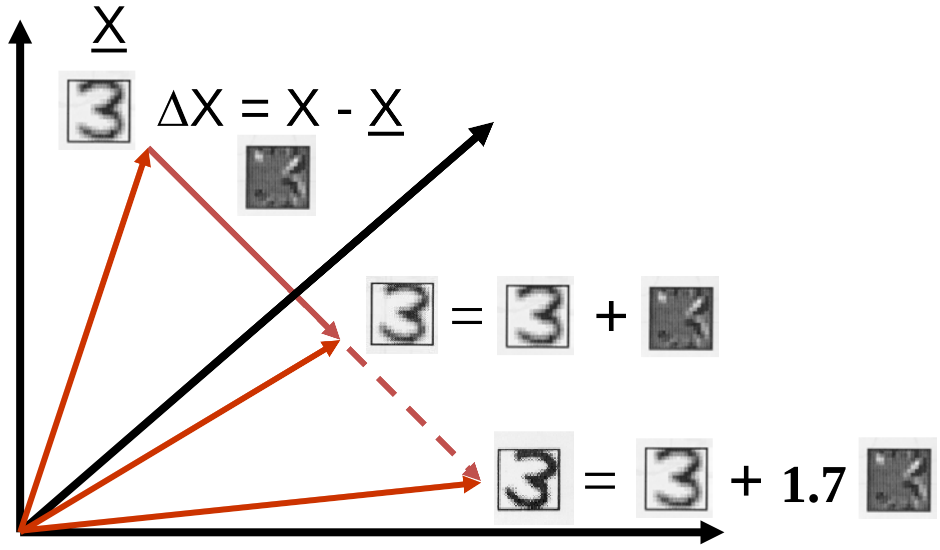

- Orientation histograms can be computed on blocks of pixels, so we can obtain tolerance to small shifts of a part of the object.
- For gray-scale images of 3d objects, the process of computing orientations, gives partial invariance to illumination changes.
- Small deformations when the orientation of a part changes only by a little causes no change in the histogram, because we bin orientations

# Invariance Manifolds




- Can we do nearest neighbors on the whole invariance manifold?

# Linear approximation (tangent)



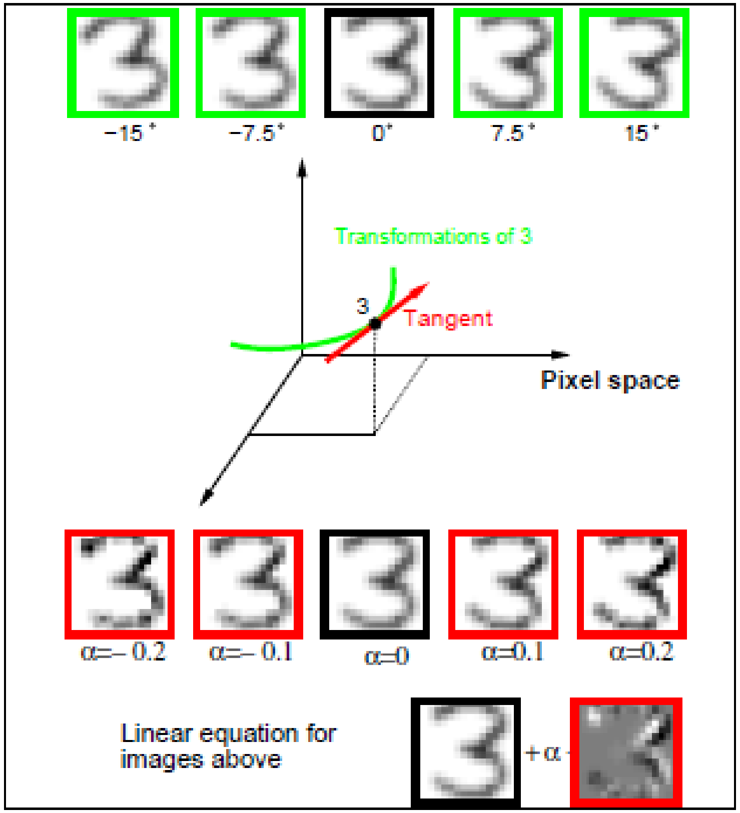



**FIGURE 13.10.** *The top row shows a "3" in its original orientation (middle) and rotated versions of it. The green curve in the middle of the figure depicts this set of rotated "3" in 256-dimensional space. The red line is the tangent line to the curve at the original image, with some "3"s on this tangent line, and its equation shown at the bottom of the figure.*

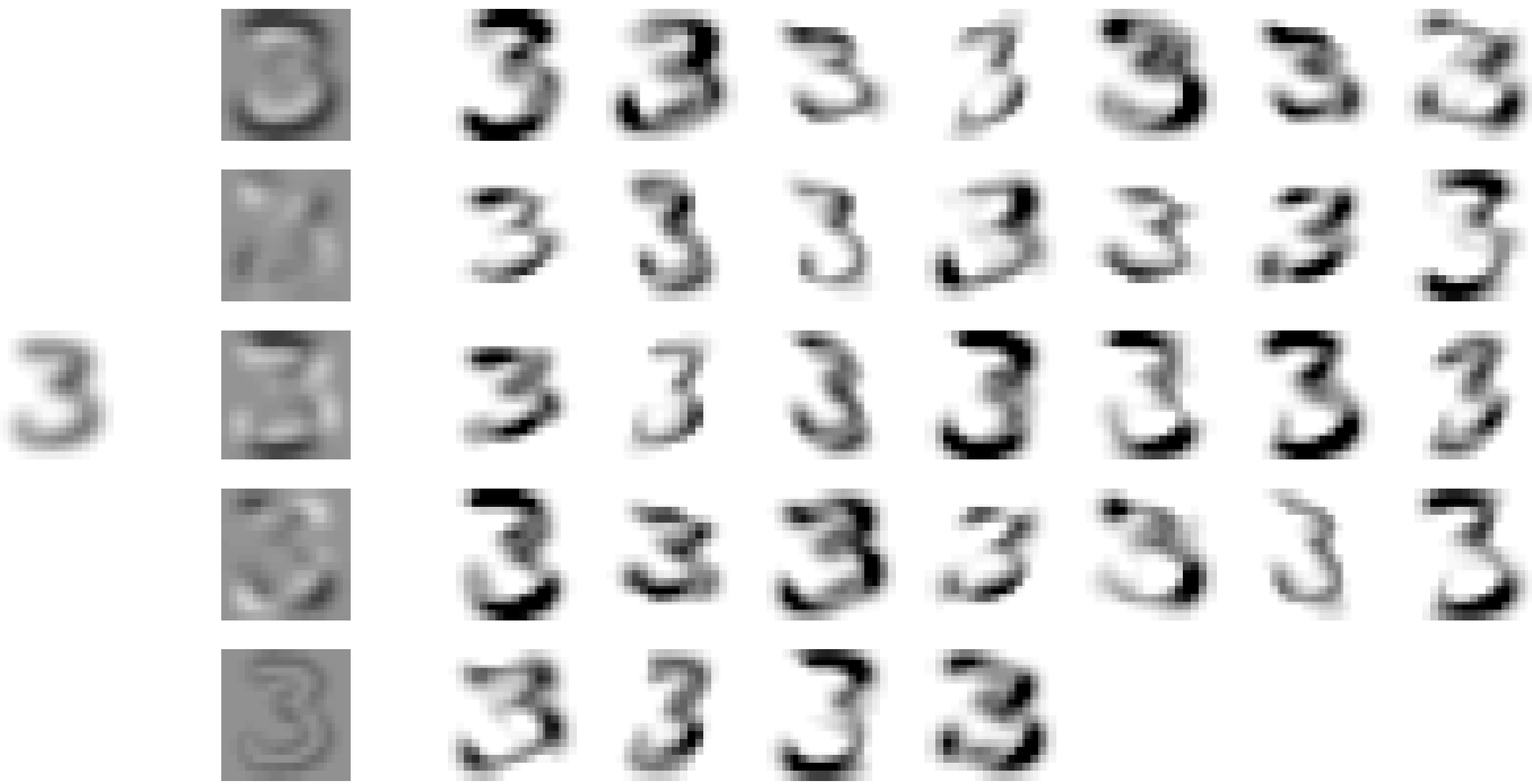


**Fig. 6.** Left: Original image. Middle: 5 tangent vectors corresponding respectively to the 5 transformations: scaling, rotation, expansion of the X axis while compressing the Y axis, expansion of the first diagonal while compressing the second diagonal and thickening. Right: 32 points in the tangent space generated by adding or subtracting each of the 5 tangent vectors.

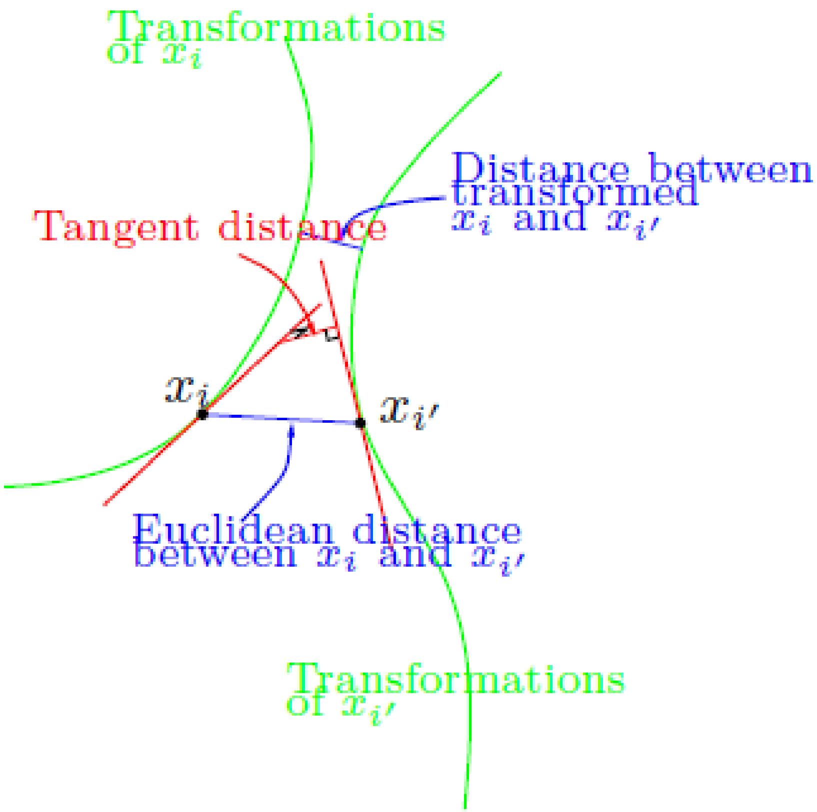



**FIGURE 13.11.** *Tangent distance computation for two images $x_i$ and $x_{i'}$. Rather than using the Euclidean distance between $x_i$ and $x_{i'}$, or the shortest distance between the two curves, we use the shortest distance between the two tangent lines.*

Details in Simard et al, "Tangent Distance" paper

# Results on Digits

| Feature | Classifier | Error Rate |
|---|---|---|
| Raw Pixels | SVM (linear) | 11.3% |
| Raw Pixels | SVM (intersection) | 8.7% |
| Raw Pixels | SVM (poly, $d = 3$) [7] | 4.0% |
| Raw Pixels | VSV (poly, $d = 3$) [7] | 3.2% |
| PHOG | SVM (linear) | 3.4% |
| PHOG | SVM (intersection) | 3.4% |
| PHOG | SVM (poly, $d = 5$) | 3.2% |
| PHOG | SVM (rbf, $\gamma = 0.1$) | 2.7% |
| Raw Pixels | Tangent Distance [23]* | 2.6% |
| Raw Pixels | Boosted Neural Nets [8]* | 2.6% |
| | Human Error Rate [3] | 2.5% |

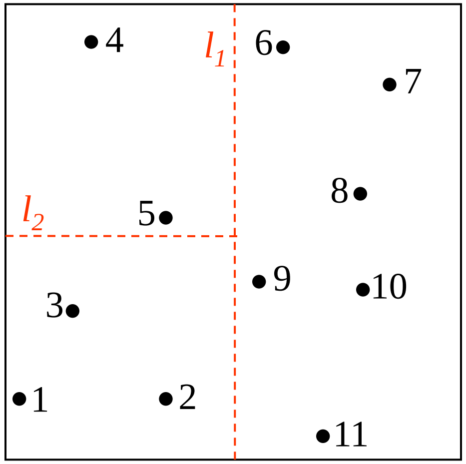
Table 5: Summary of various results on the USPS dataset. Both the linear and the intersection kernel SVMs outperform the existing numbers using SVMs which is at 4%. The VSV method which jitters the Support Vectors to create additional training examples, and retrains a SVM, leads to an improved accuracy of 3.2%. Using polynomial and rbf kernel SVMs on PHOG features reduces the error rate even further to 3.2% and 2.7% respectively. Some of the results shown in * use a different training dataset which has been enhanced by adding machine-printed characters. Note that our numbers are the best in the unmodified version of the dataset.

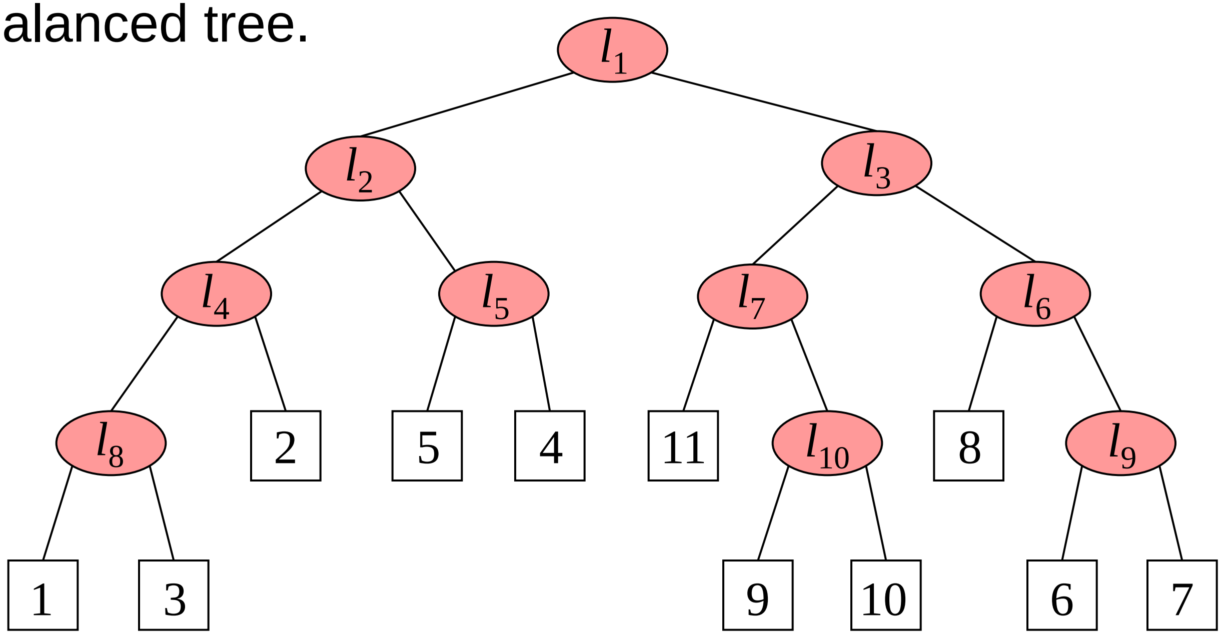
# Reducing Computational Cost

- Nearest-neighbors has O(N) complexity
  - Infeasible for large datasets
- Can we speed it up?
  - Think of guessing a number between 1 and 10…

# K-d tree

- K-d tree is a binary tree data structure for organizing a set of points in a K-dimensional space.
- Each internal node is associated with an axis aligned hyper-plane splitting its associated points into two sub-trees.
- Dimensions with high variance are chosen first.
- Position of the splitting hyper-plane is chosen as the mean/median of the projected points – balanced tree.

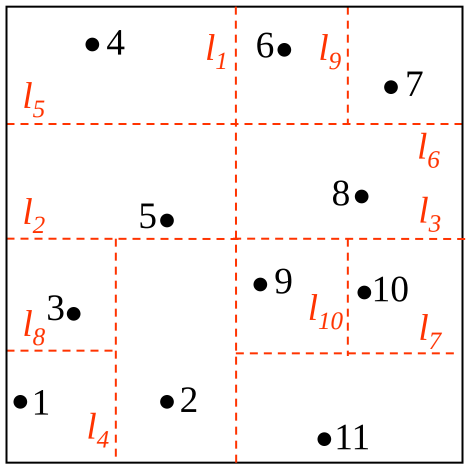


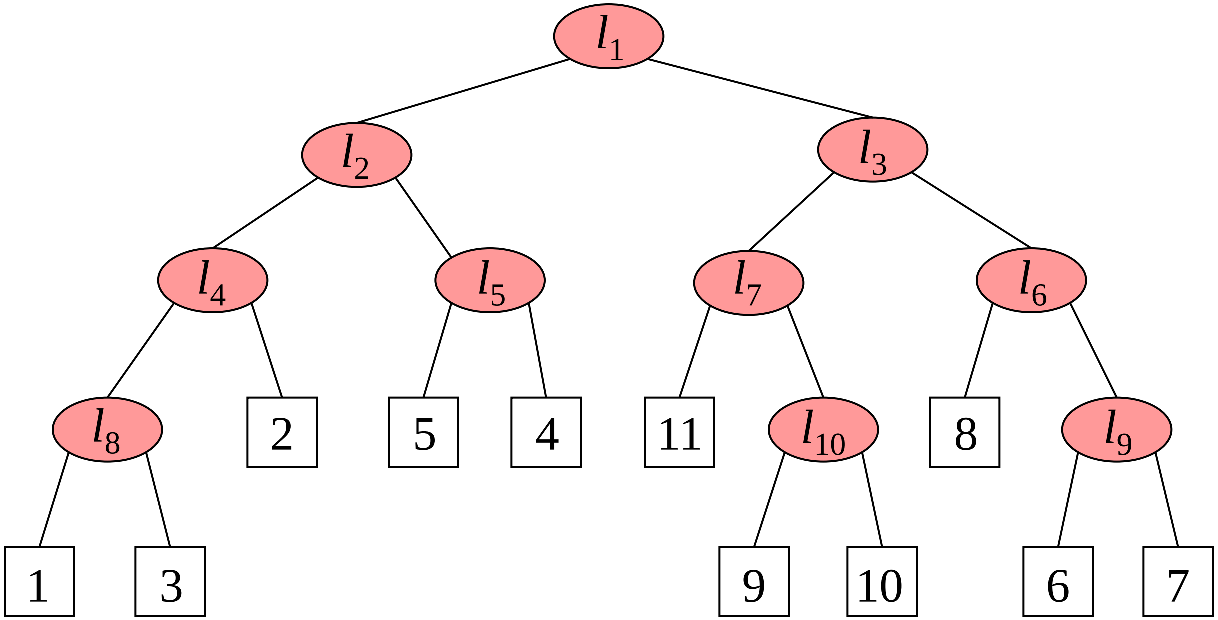




Images: Anna Atramentov

# K-d tree construction

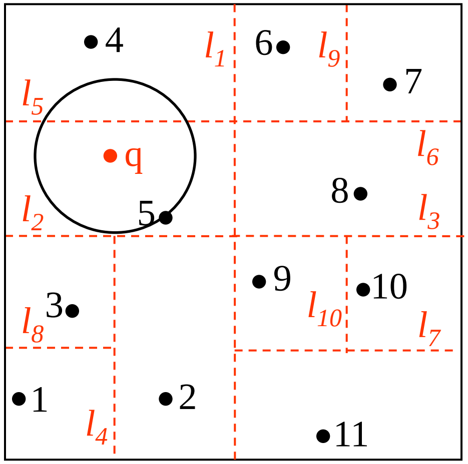
Simple 2D example

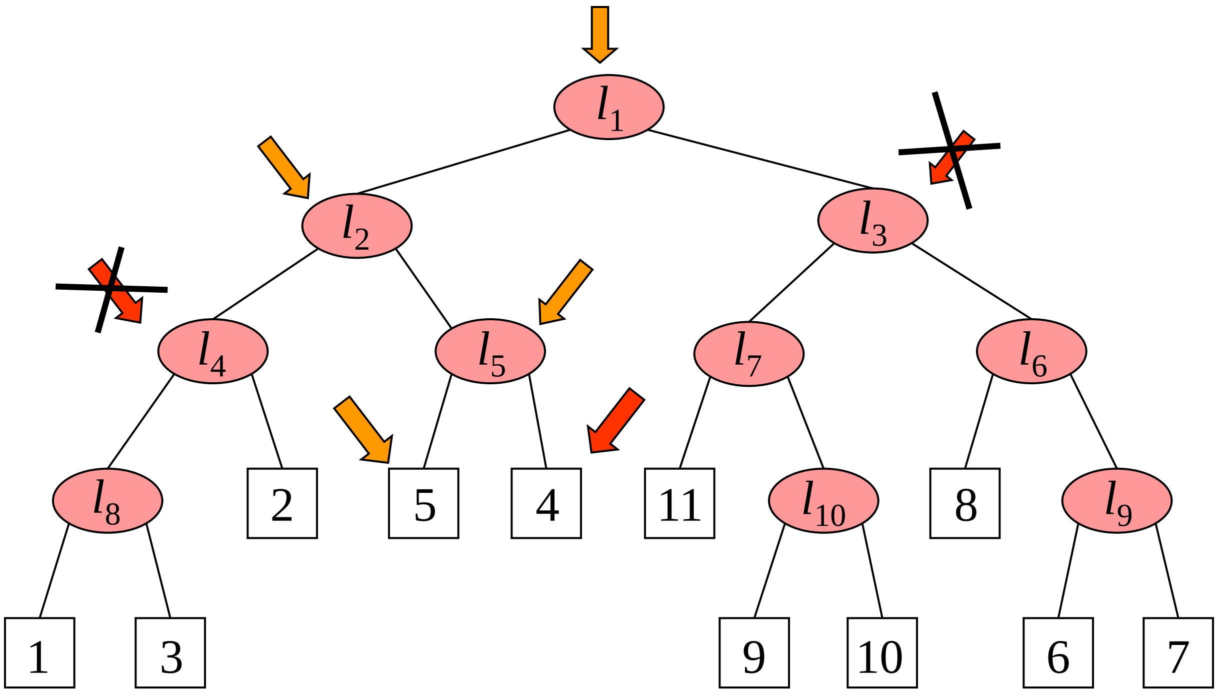



Slide credit: Anna Atramentov

# K-d tree query




Slide credit: Anna Atramentov

# K-d tree: Backtracking

Backtracking is necessary as the true nearest neighbor may not lie in the query cell.

But in some cases, almost all cells need to be inspected.

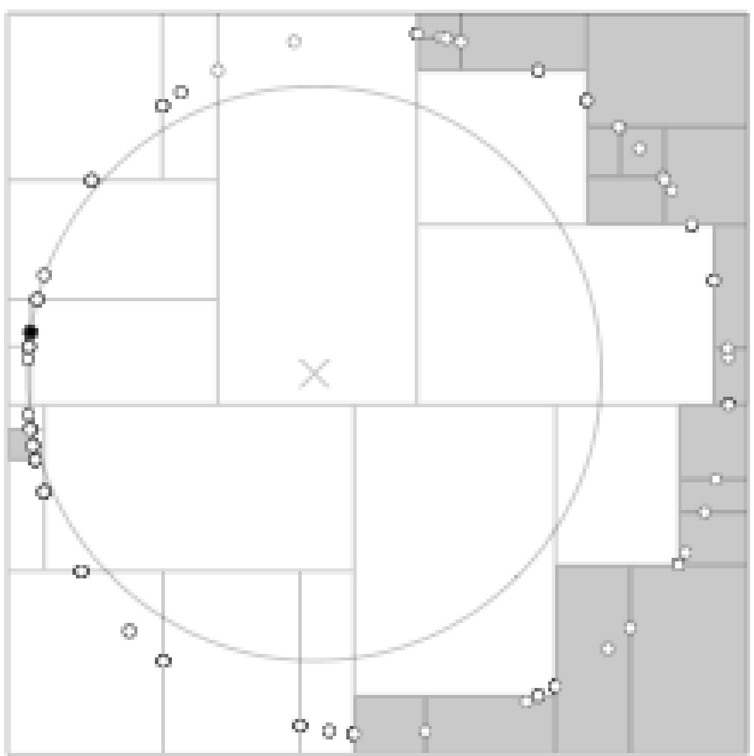


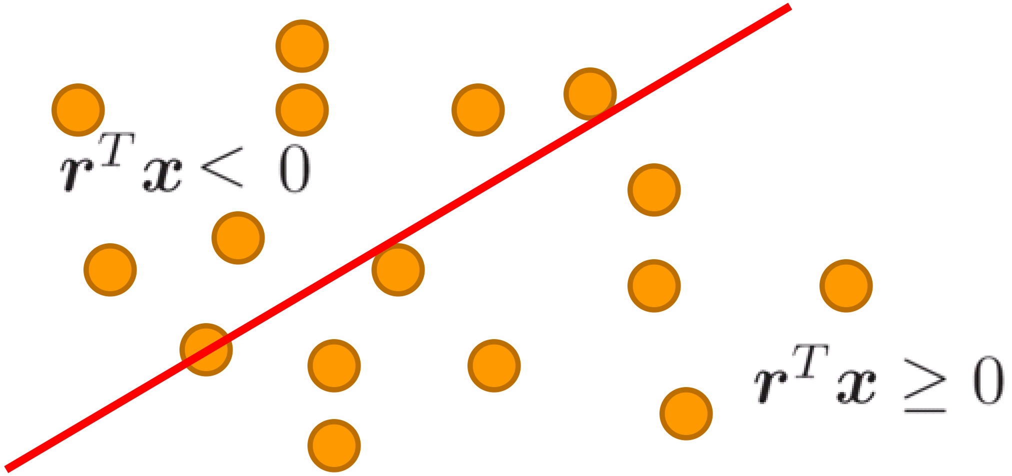
Figure 6.6

A bad distribution which forces almost all nodes to be inspected.

Figure: A. Moore

# Do we need axis-aligned hyperplanes?

Normal unit vector *r* defines is a hyperplane separating the space

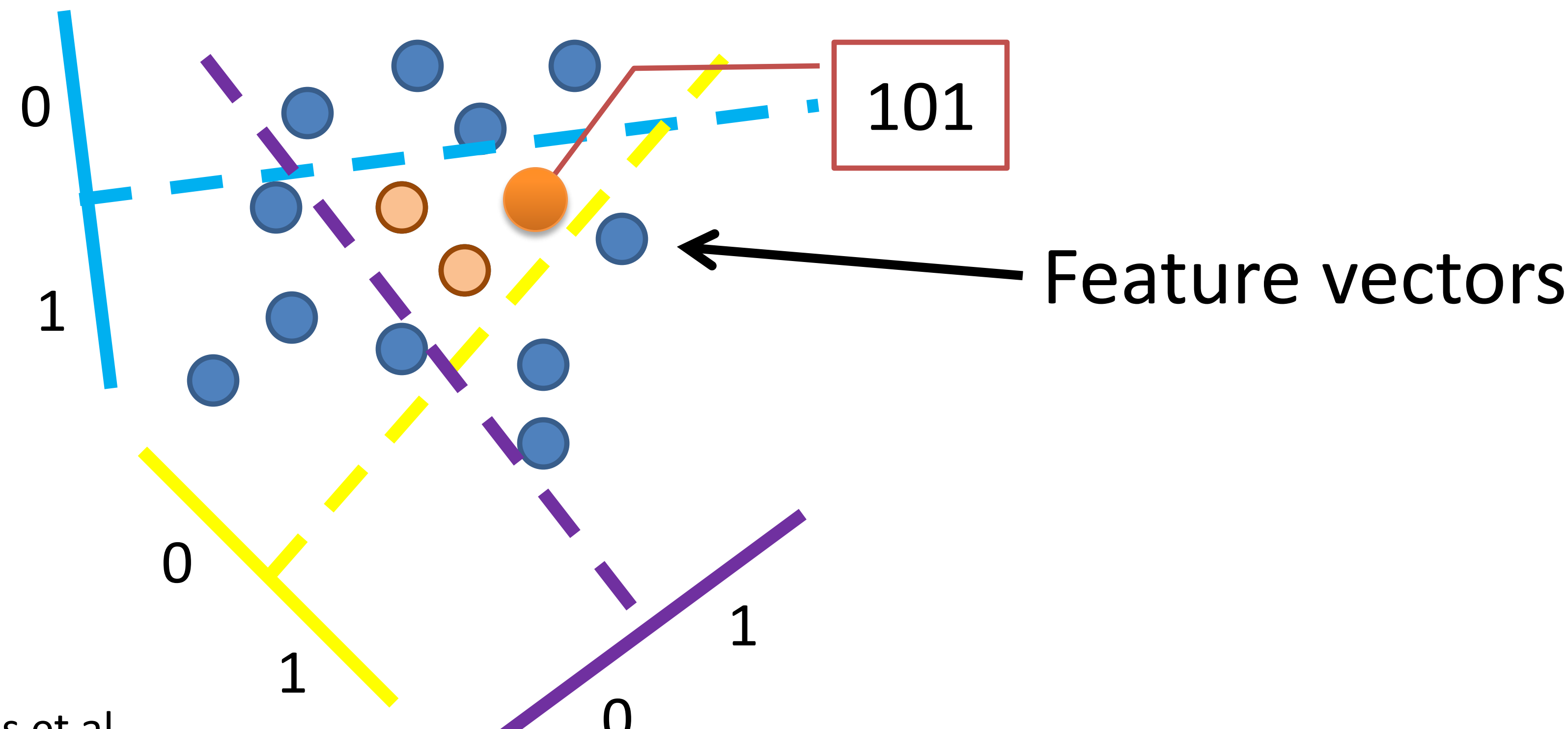



For any point x, define:

$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

# Hashing by Random Projections

- Take random projections of data $r^T x$
- Quantize each projection with few bits




Fergus et al

# Locality Sensitive Hashing

- The basic idea behind LSH is to project the data into a low-dimensional binary (Hamming) space; that is, each data point is mapped to a b-bit vector, called the *hash key.*
- Unlike normal hashing, here we want our hashes to cluster – create collisions
- Each hash function h must satisfy the locality sensitive hashing property:

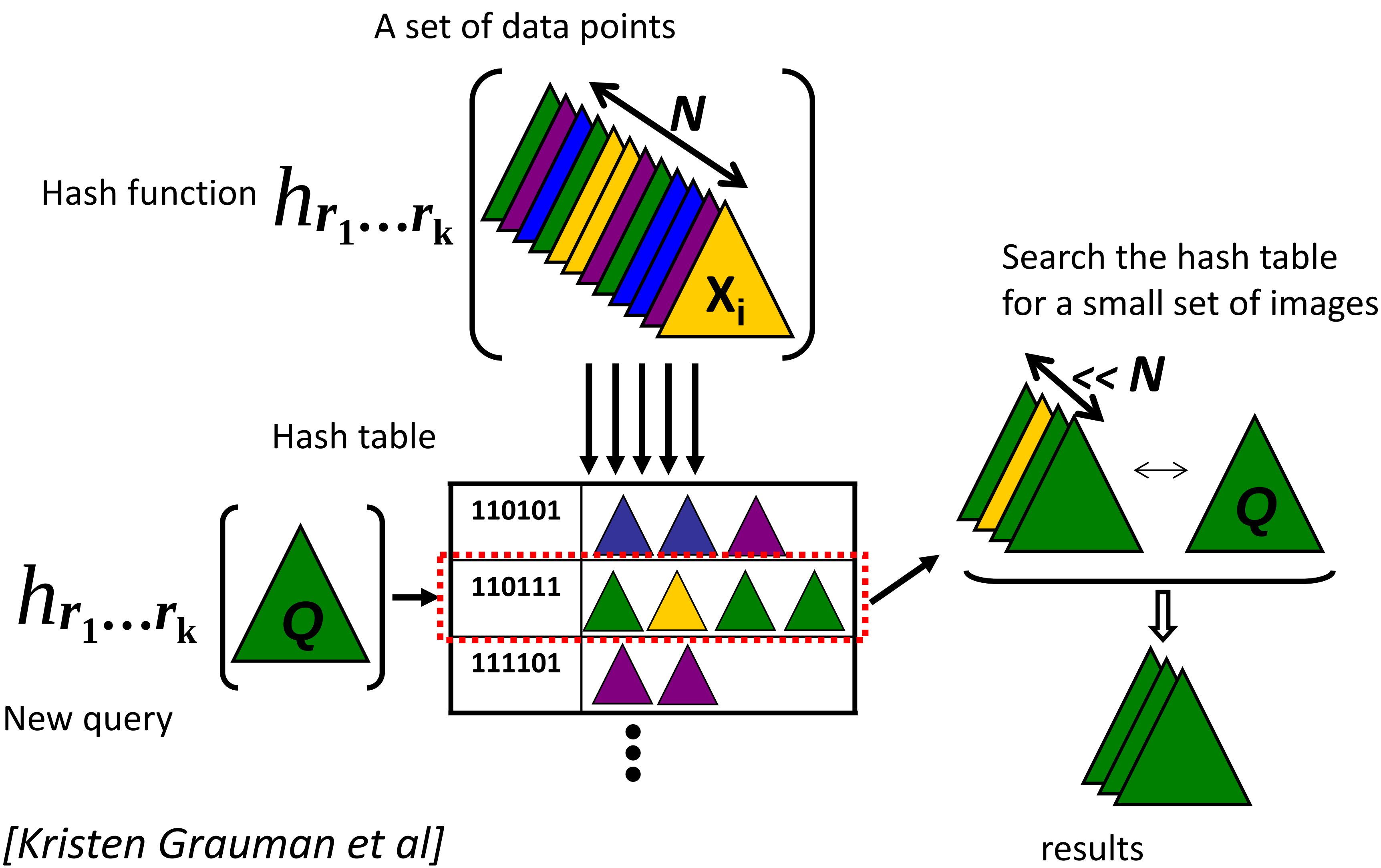
$$\Pr[h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)] = \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

  – Where $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in [0, 1]$ is the similarity function. In our case: $Pr[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$

Kristen Grauman et al

Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *SOCG, 2004.*

# Approximate Nearest-Neighbor Search




[Kristen Grauman et al]