Handwritten digit recognition

Jitendra Malik

Handwritten digit recognition (MNIST,USPS)

- LeCun's Convolutional Neural Networks variations (0.8%, 0.6% and 0.4% on MNIST)
- Tangent Distance(Simard, LeCun & Denker: 2.5% on USPS)
- Randomized Decision Trees (Amit, Geman & Wilder, 0.8%)
- K-NN based Shape context/TPS matching (Belongie, Malik & Puzicha: 0.6% on MNIST)
- SVM on orientation histograms(Maji & Malik, 0.8%)

The MNIST DATABASE of handwritten digits

yann.lecun.com/exdb/mnist/

Yann LeCun & Corinna Cortes

- Has a training set of 60 K examples (6K examples for each digit), and a test set of 10K examples.
- Each digit is a 28 x 28 pixel grey level image. The digit itself occupies the central 20 x 20 pixels, and the center of mass lies at the center of the box.
- *“It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.”*

The machine learning approach to object recognition

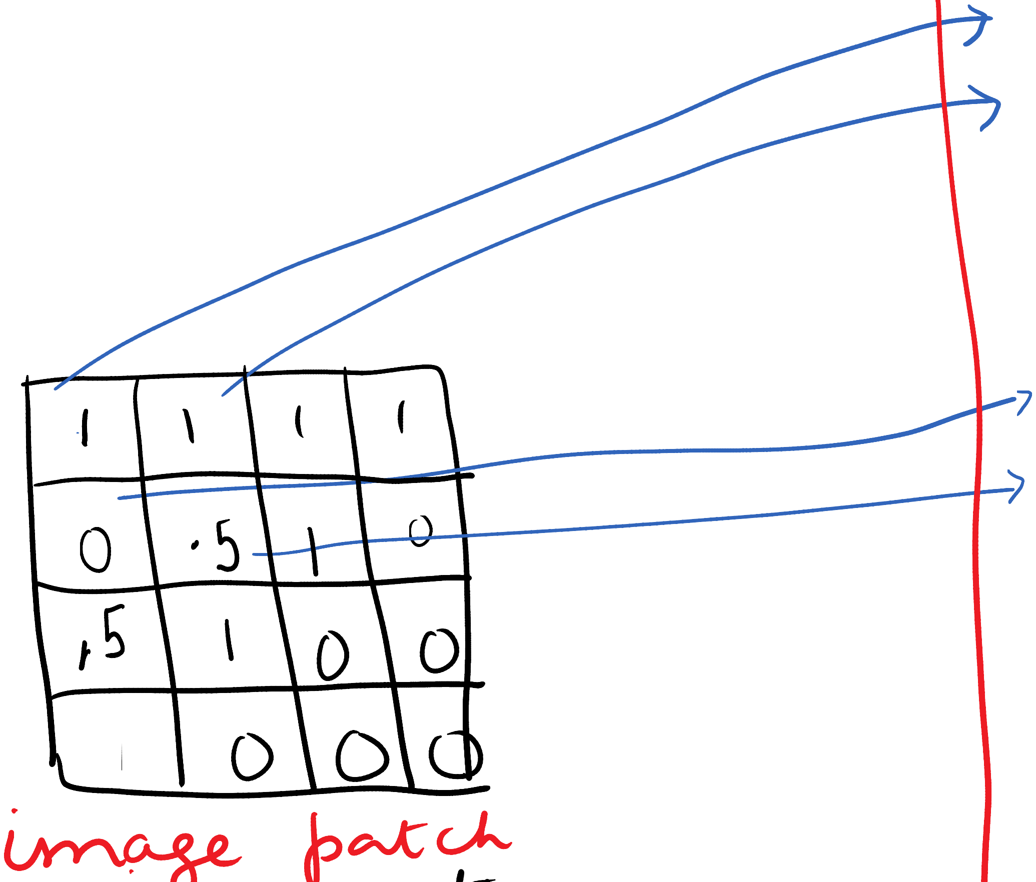
- Training time
  - Compute feature vectors for positive and negative examples of image patches
  - Train a classifier
- Test Time
  - Compute feature vector on image patch
  - Evaluate classifier

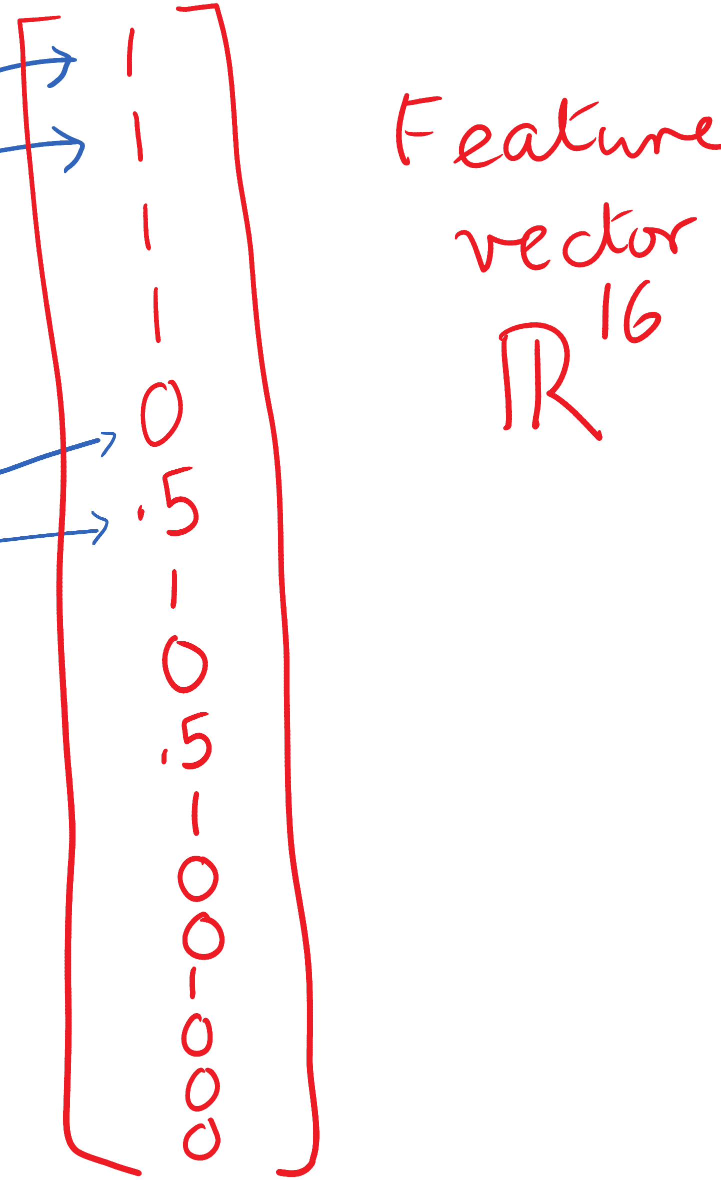
Let us take an example...

	1	1	
0	.5	1	0
.5	1	0	0
1	0	0	0

image patch

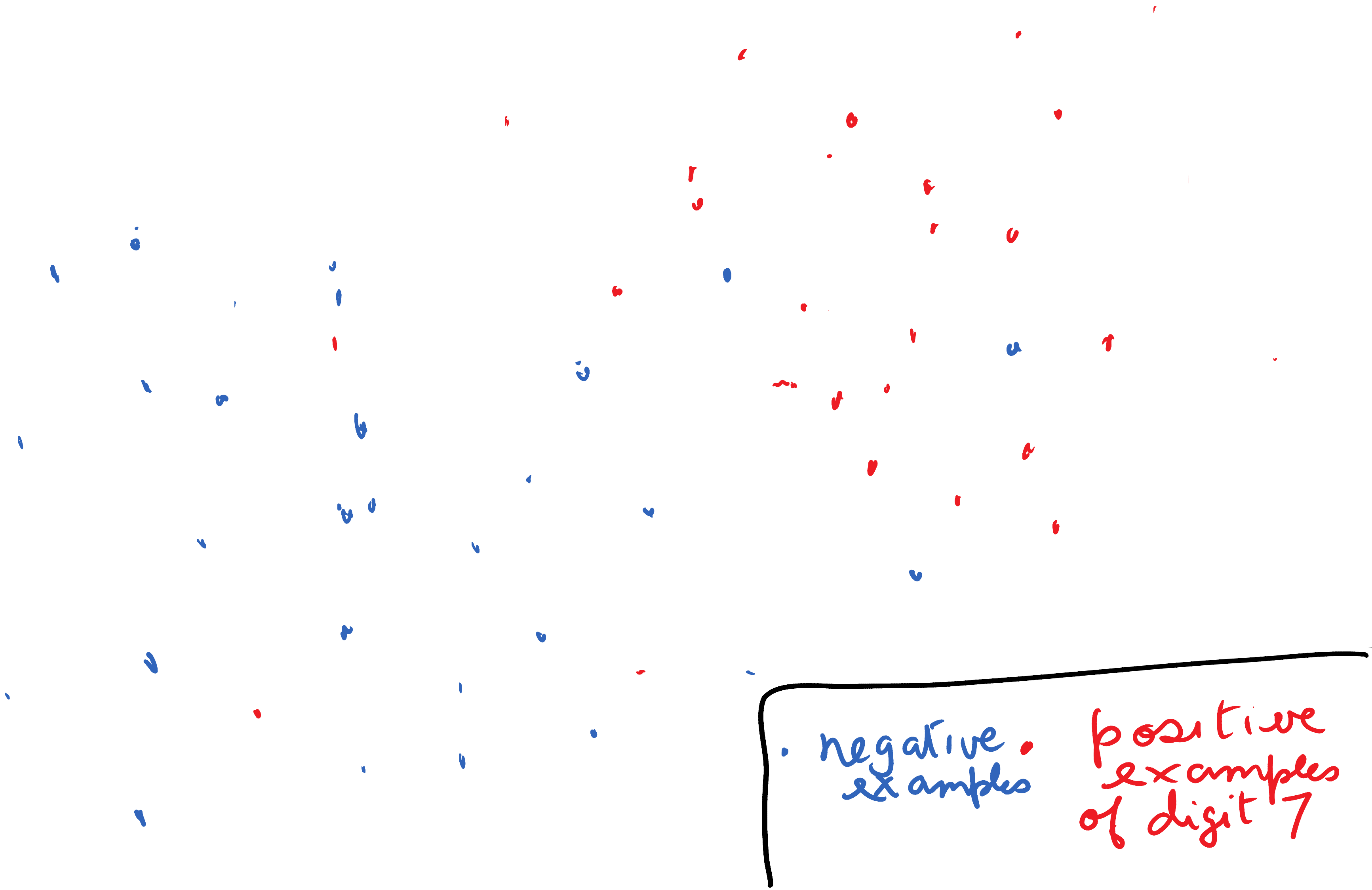
Let us take an example...



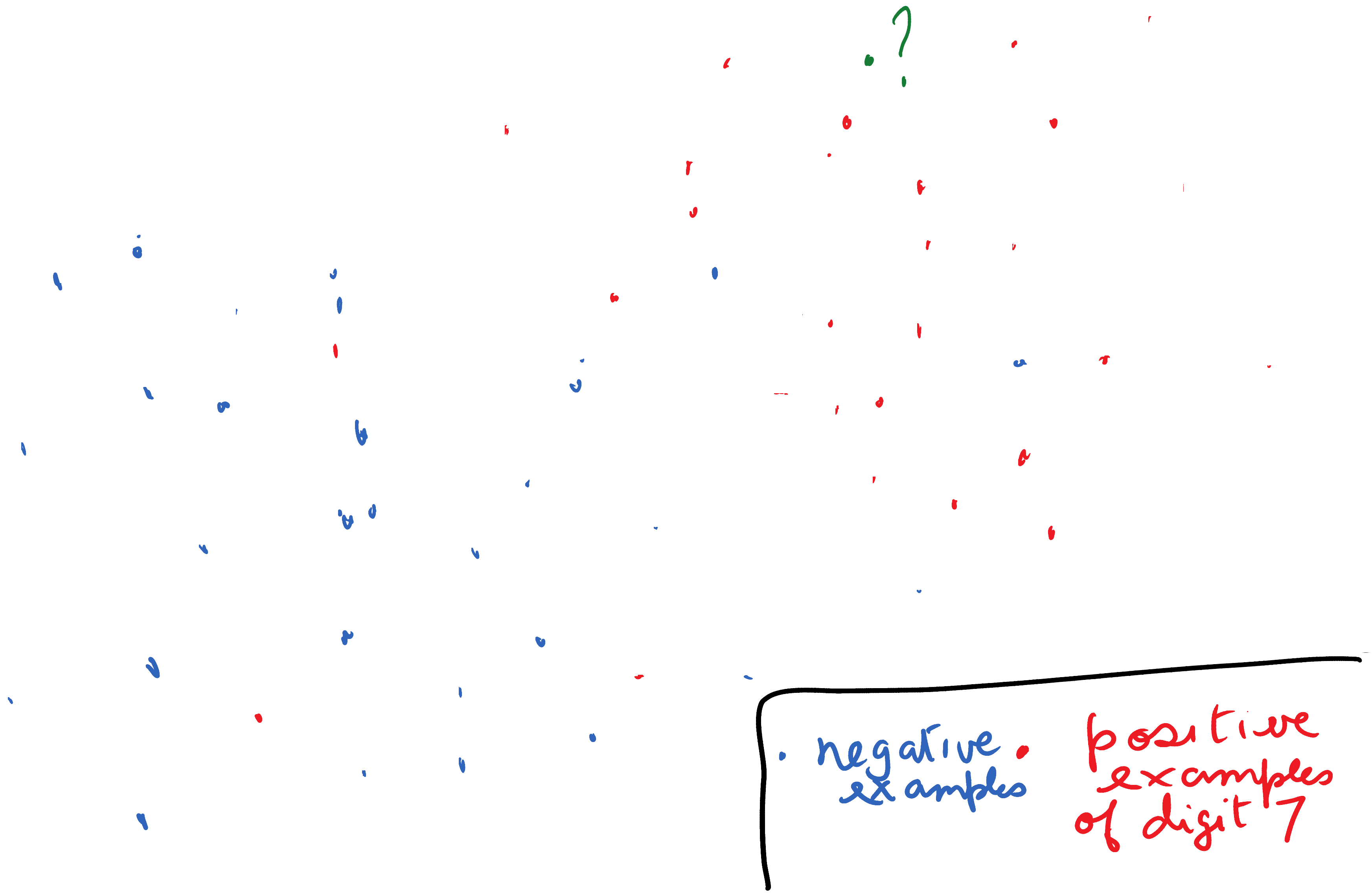


Note that there are several ways to construct a feature vector. This is one example..

In feature space, positive and negative examples are just points…

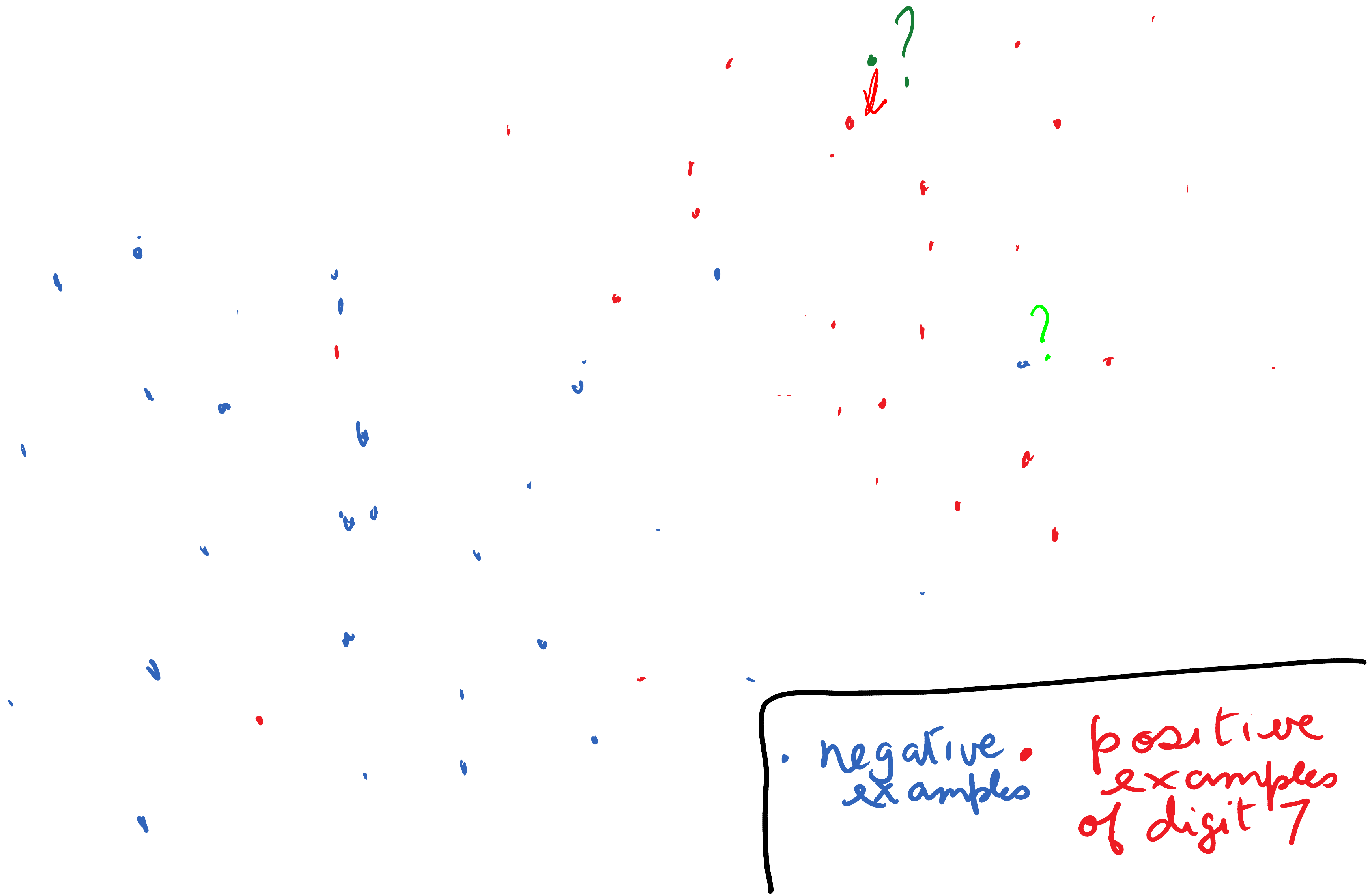


How do we classify a new point?



Nearest neighbor rule
“transfer label of nearest example”

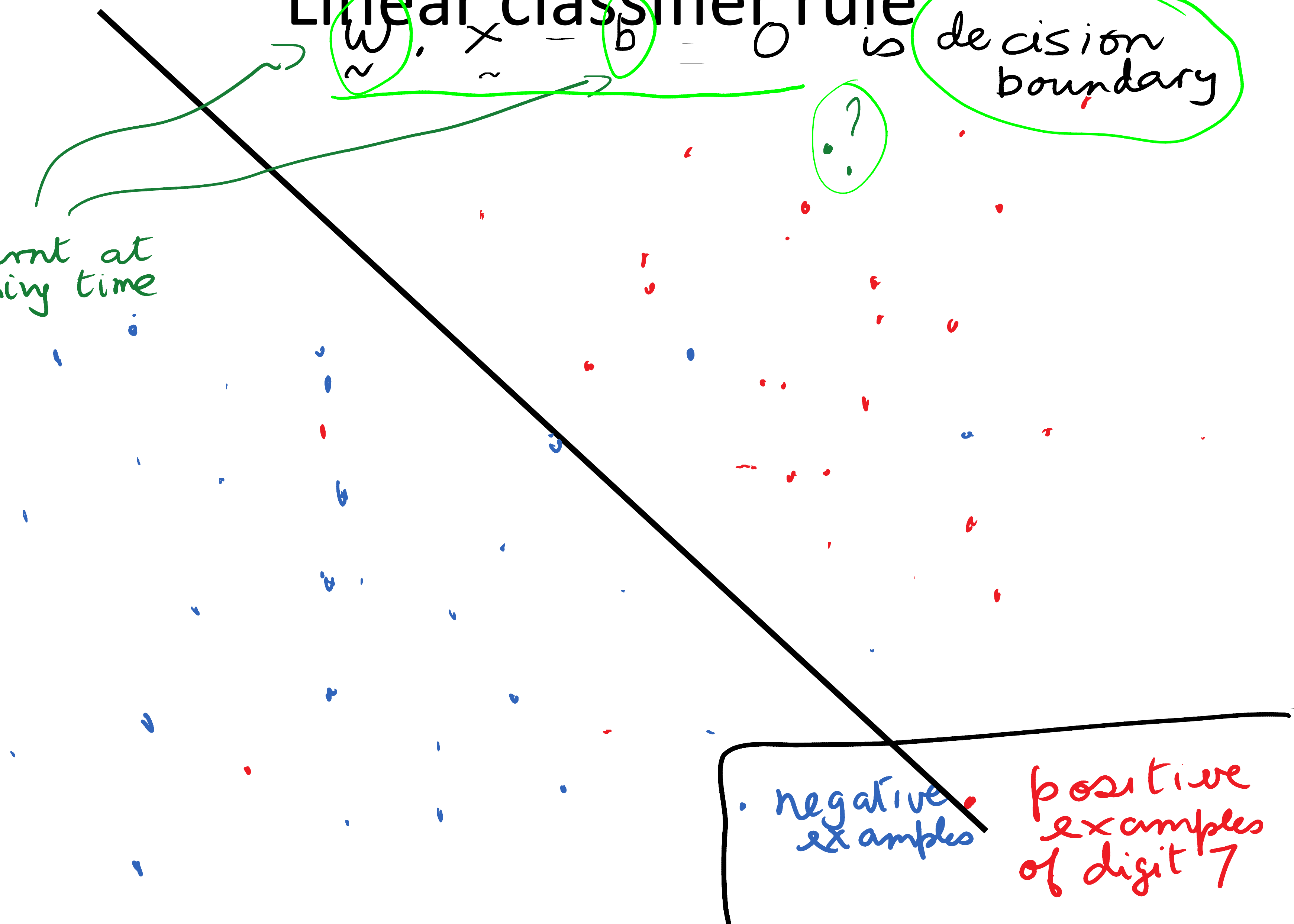
1-NN
3-NN



Linear classifier rule

$\underset{\sim}{w} \cdot \underset{\sim}{x} - b = 0$ is decision boundary

learnt at training time

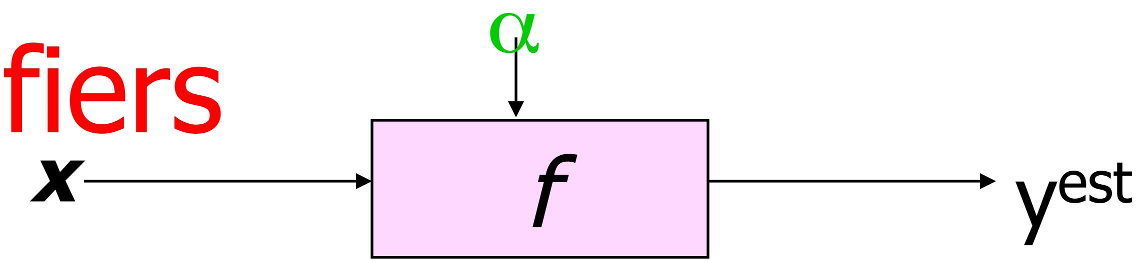


Different approaches to training classifiers

- Nearest neighbor methods
- Neural networks (multi-layer perceptrons)
- Support vector machines
- Randomized decision trees (random forests)
- …

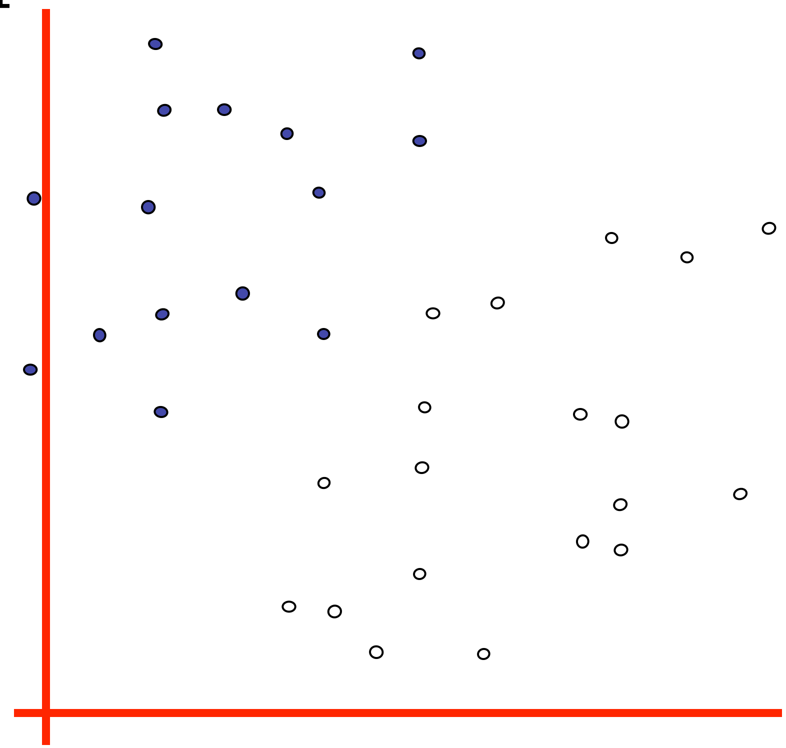
A machine learning course will go into more details on these (and other) approaches.
I will just cover two (SVMs, decision trees)

Linear Classifiers



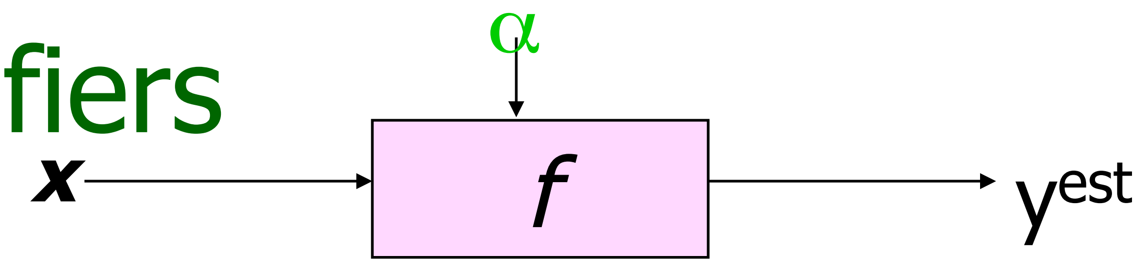
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$

- denotes +1
- ○ denotes -1



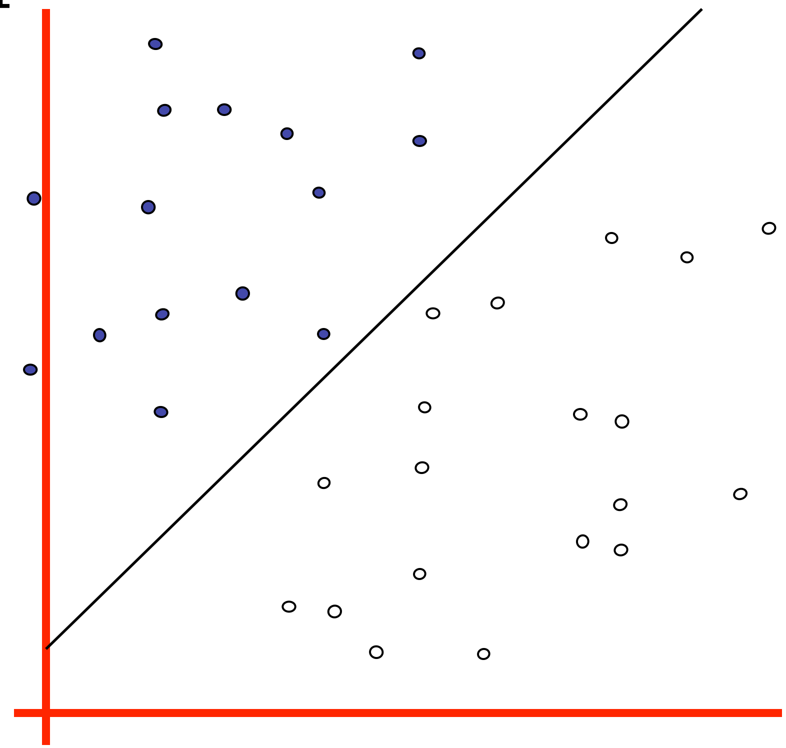
How would you classify this data?

Linear Classifiers



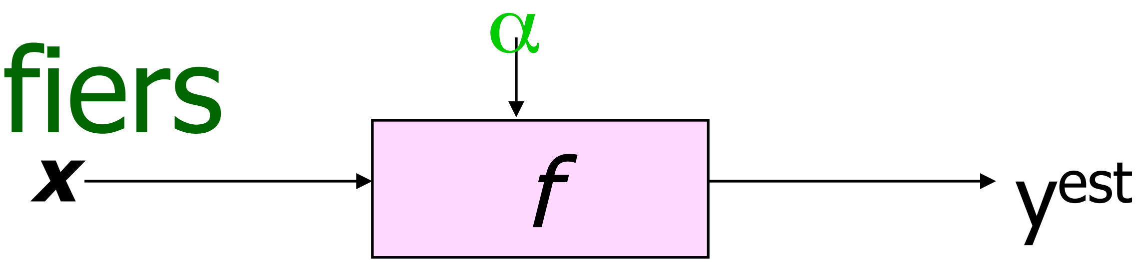
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$

- denotes +1
- denotes -1



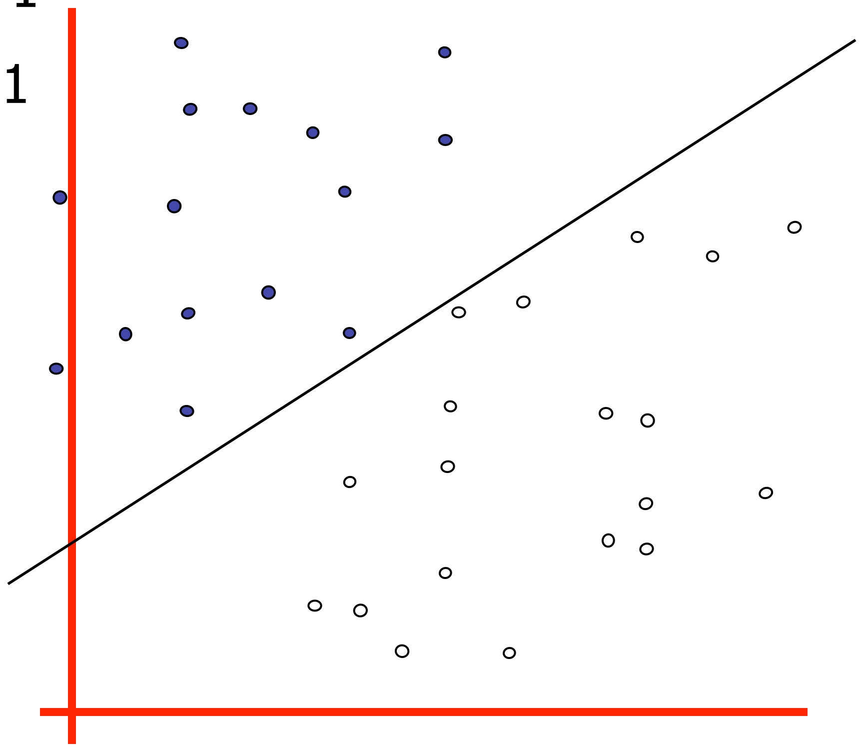
How would you classify this data?

Linear Classifiers



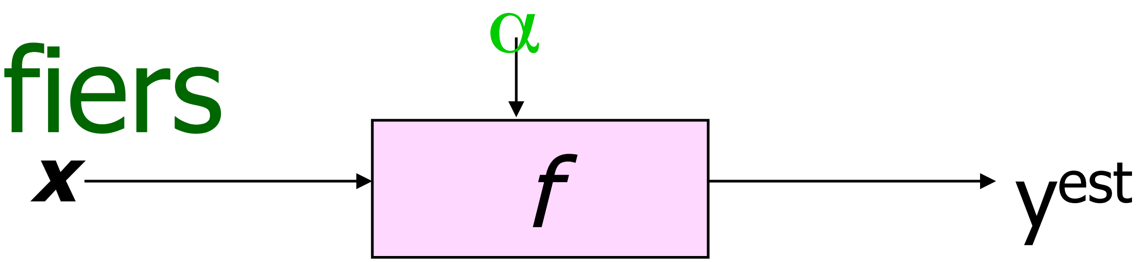
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- • denotes +1
- ○ denotes -1



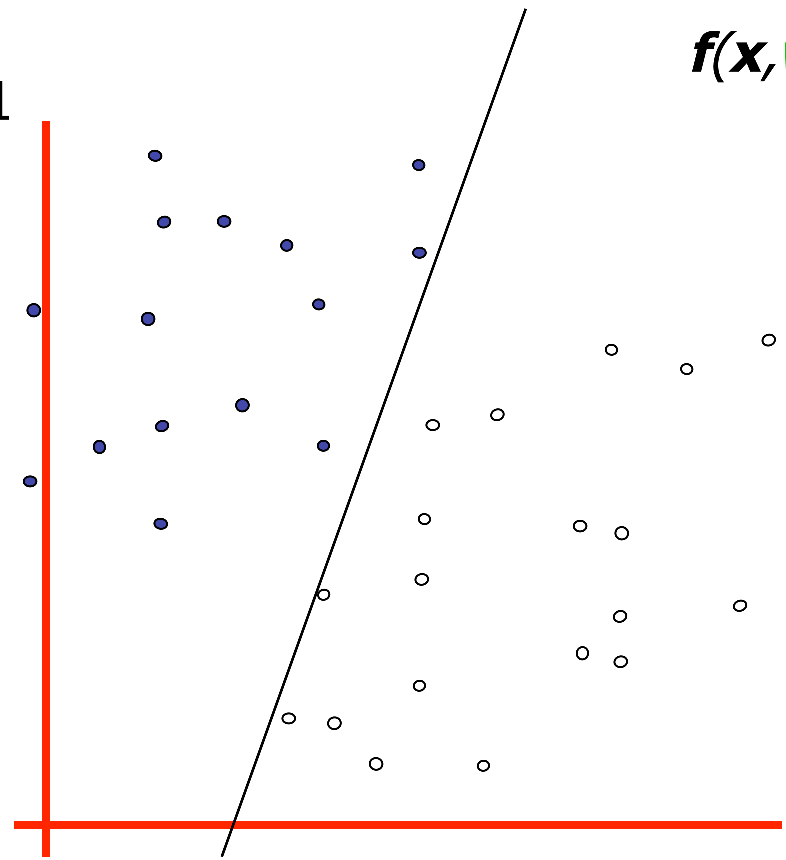
How would you classify this data?

Linear Classifiers



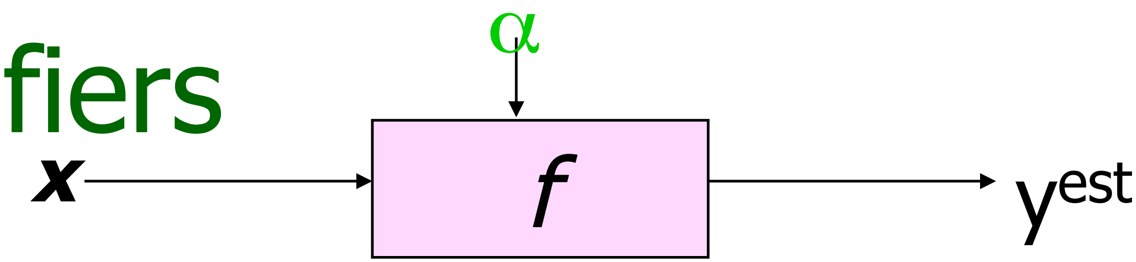
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$

- ● denotes +1
- ○ denotes -1



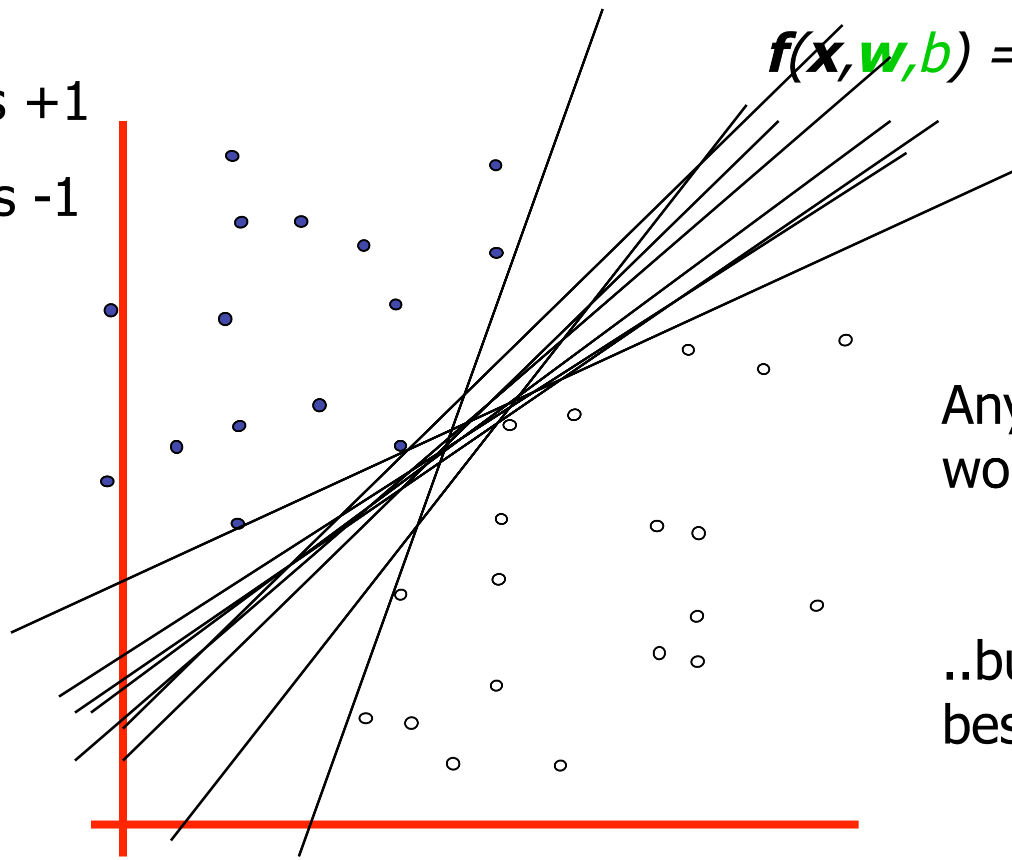
How would you classify this data?

Linear Classifiers



$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$

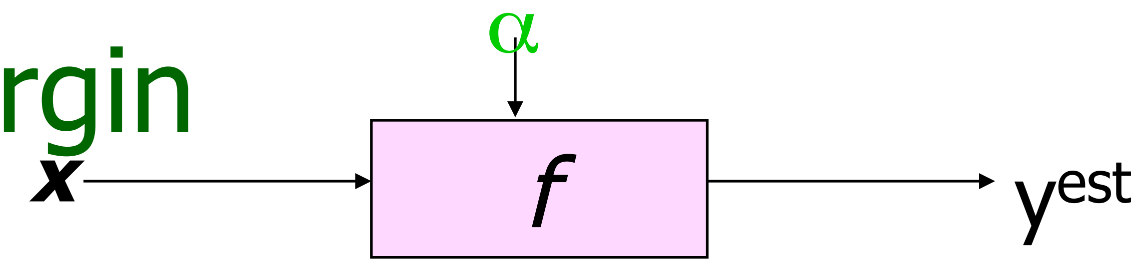
- ● denotes +1
- ○ denotes -1



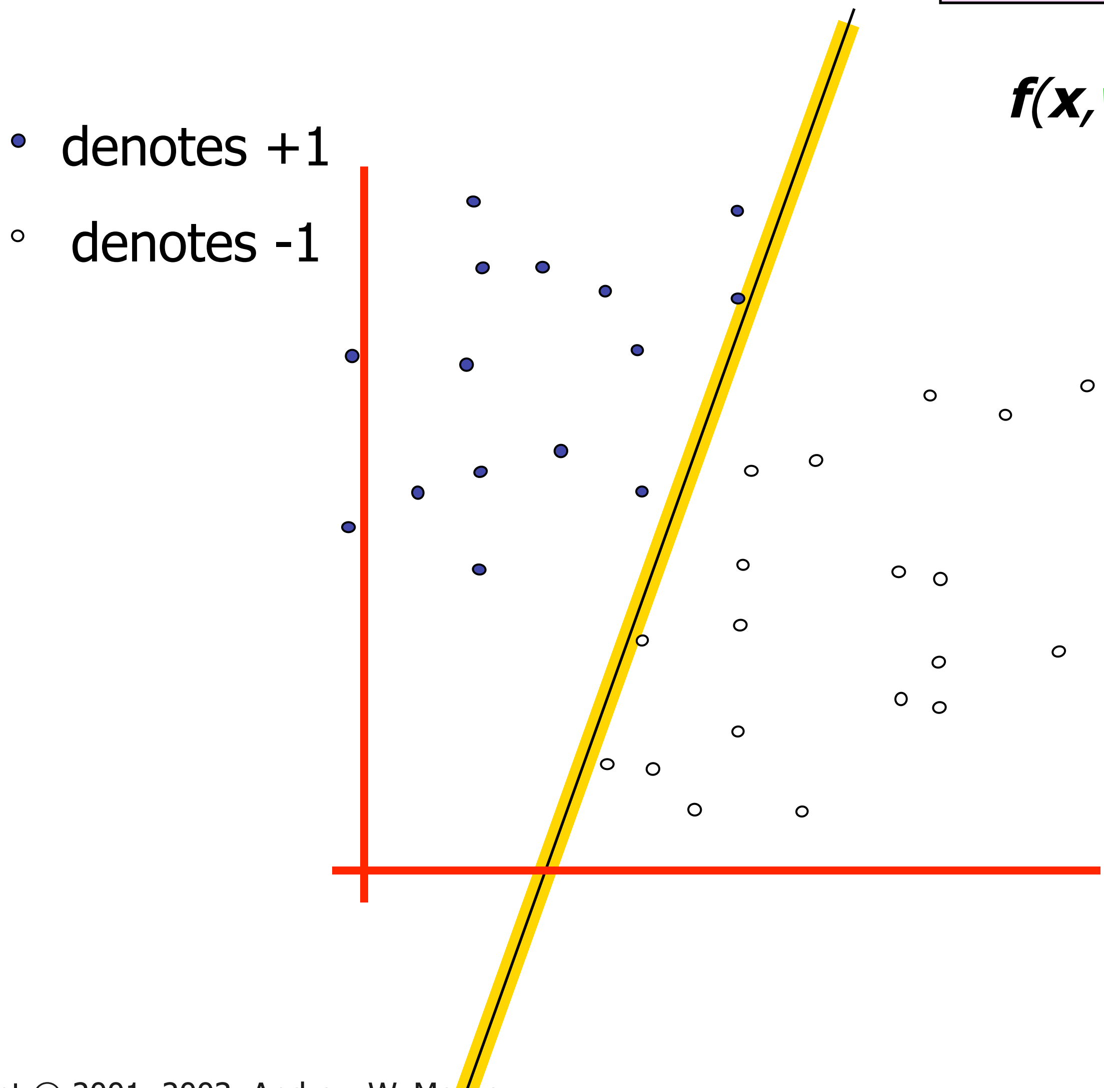
Any of these would be fine..

..but which is best?

Classifier Margin

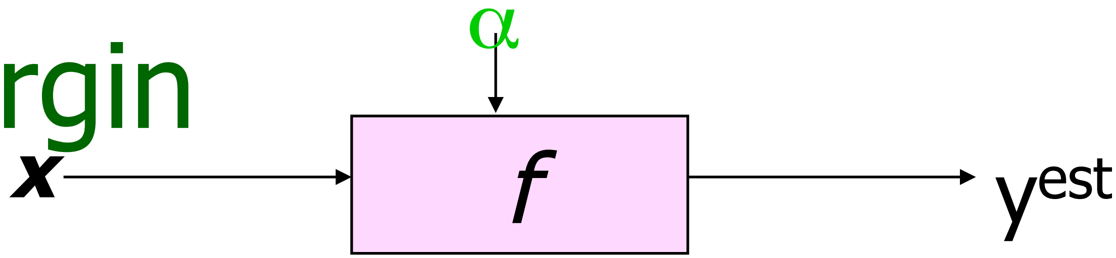


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$

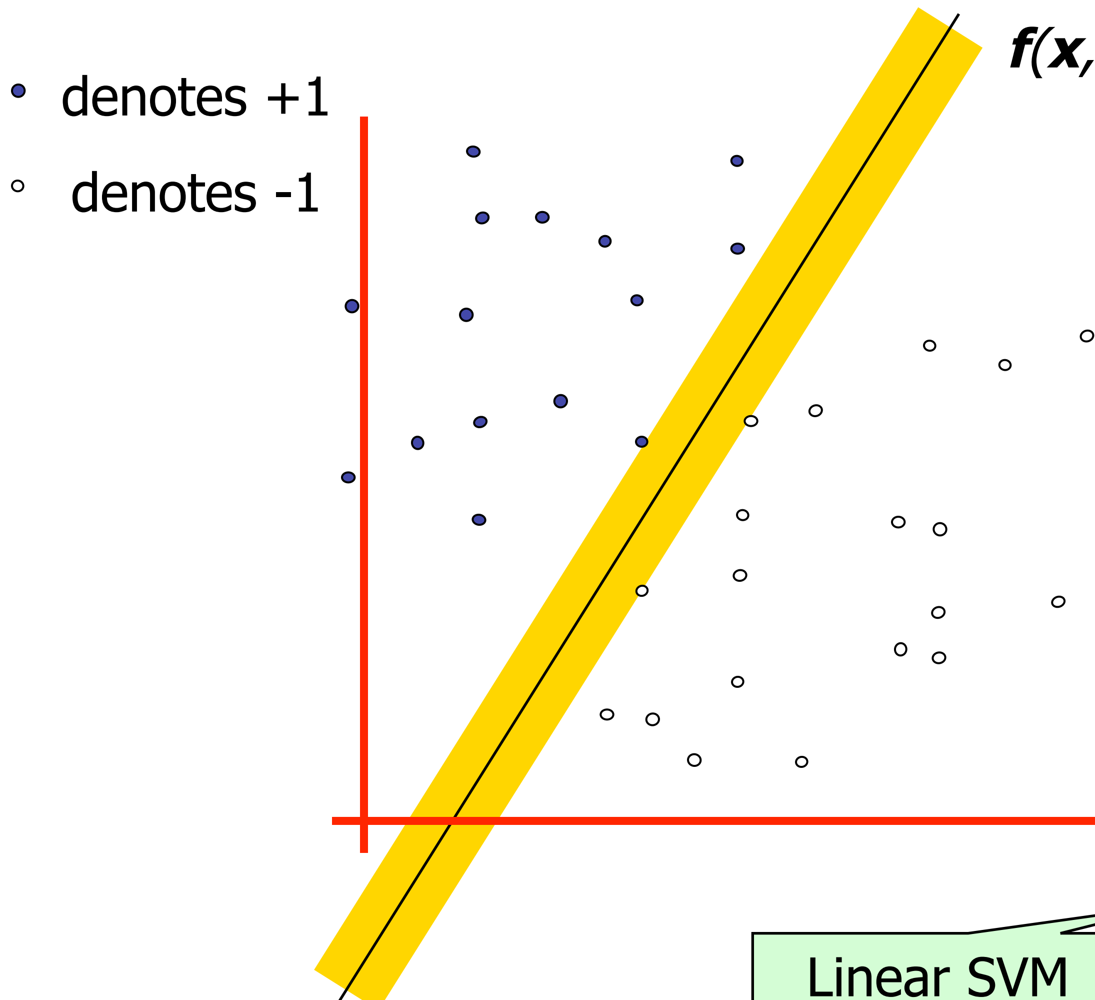


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

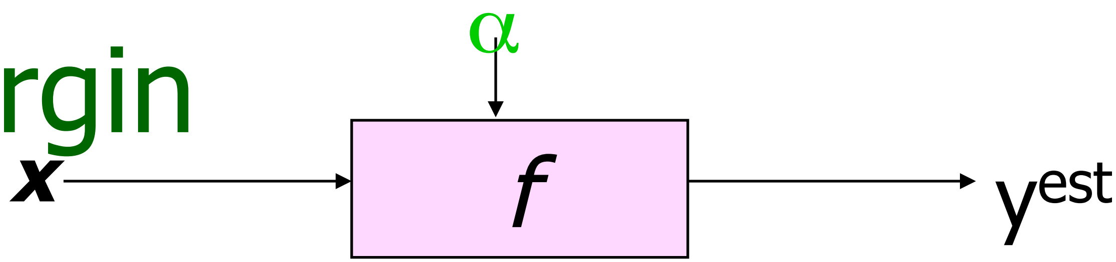


The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

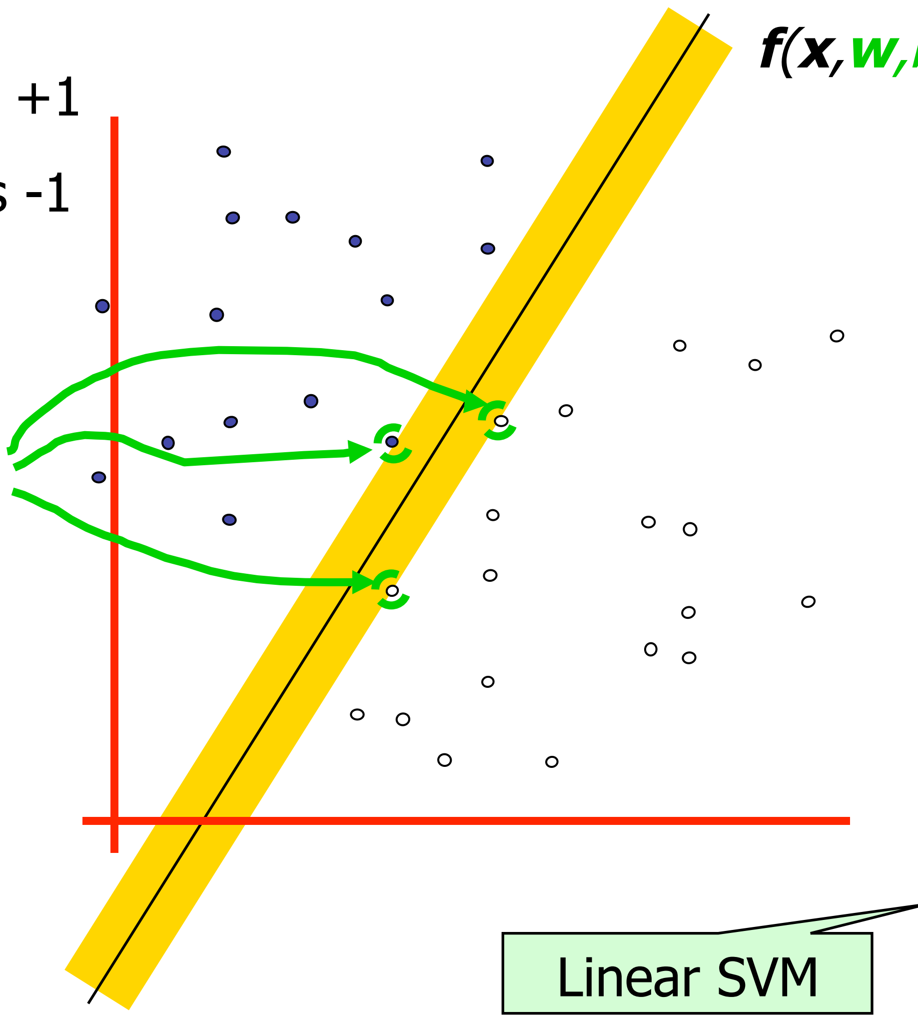
Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

• denotes +1

° denotes -1

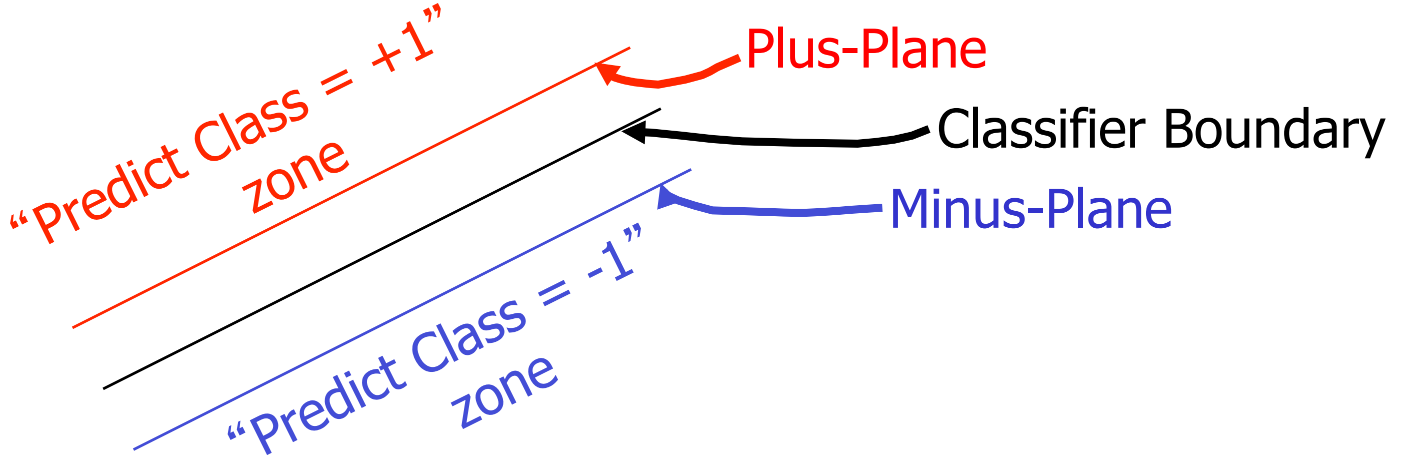


Support Vectors are those datapoints that the margin pushes up against => just a few

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

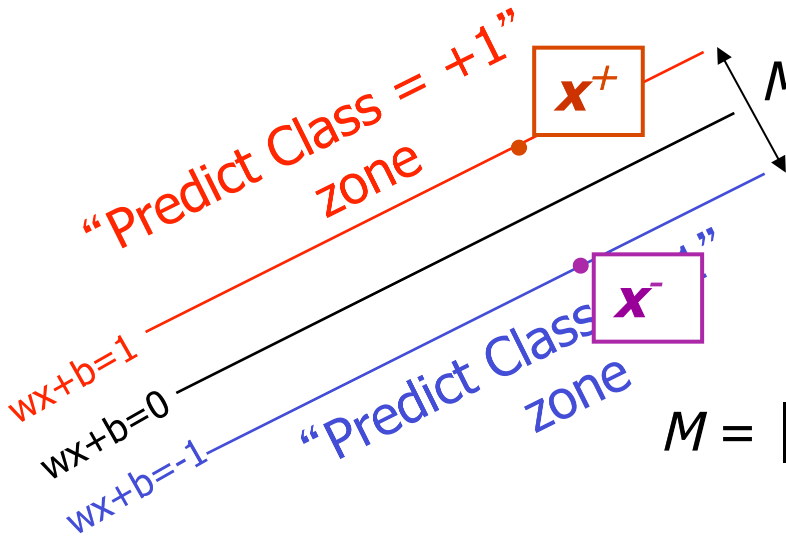
This is the simplest kind of SVM (Called an LSVM)

Specifying a line and margin



- How do we represent this mathematically?
- ...in *m* input dimensions?

Computing the margin width



M = Margin Width = $\dfrac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

What we know:

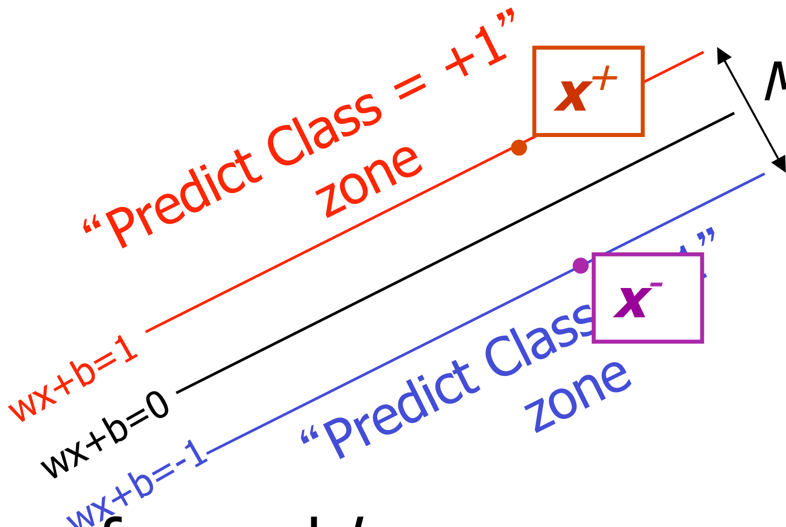
- $\boldsymbol{w} \,.\, \boldsymbol{x}^{+} + b = +1$
- $\boldsymbol{w} \,.\, \boldsymbol{x}^{-} + b = -1$
- $\boldsymbol{x}^{+} = \boldsymbol{x}^{-} + \lambda\, \boldsymbol{w}$
- $|\boldsymbol{x}^{+} - \boldsymbol{x}^{-}| = M$
- $$\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$$

$$M = |\boldsymbol{x}^{+} - \boldsymbol{x}^{-}| = |\lambda\, \boldsymbol{w}| =$$

$$= \lambda\, |\mathbf{w}| = \lambda \sqrt{\mathbf{w}.\mathbf{w}}$$

$$= \frac{2\sqrt{\mathbf{w}.\mathbf{w}}}{\mathbf{w}.\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Learning the Maximum Margin Classifier



M = Margin Width = $\dfrac{2}{\sqrt{\mathbf{w.w}}}$

Given a guess of $\boldsymbol{w}$ and b we can

- Compute whether all data points in the correct half-planes
- Compute the width of the margin

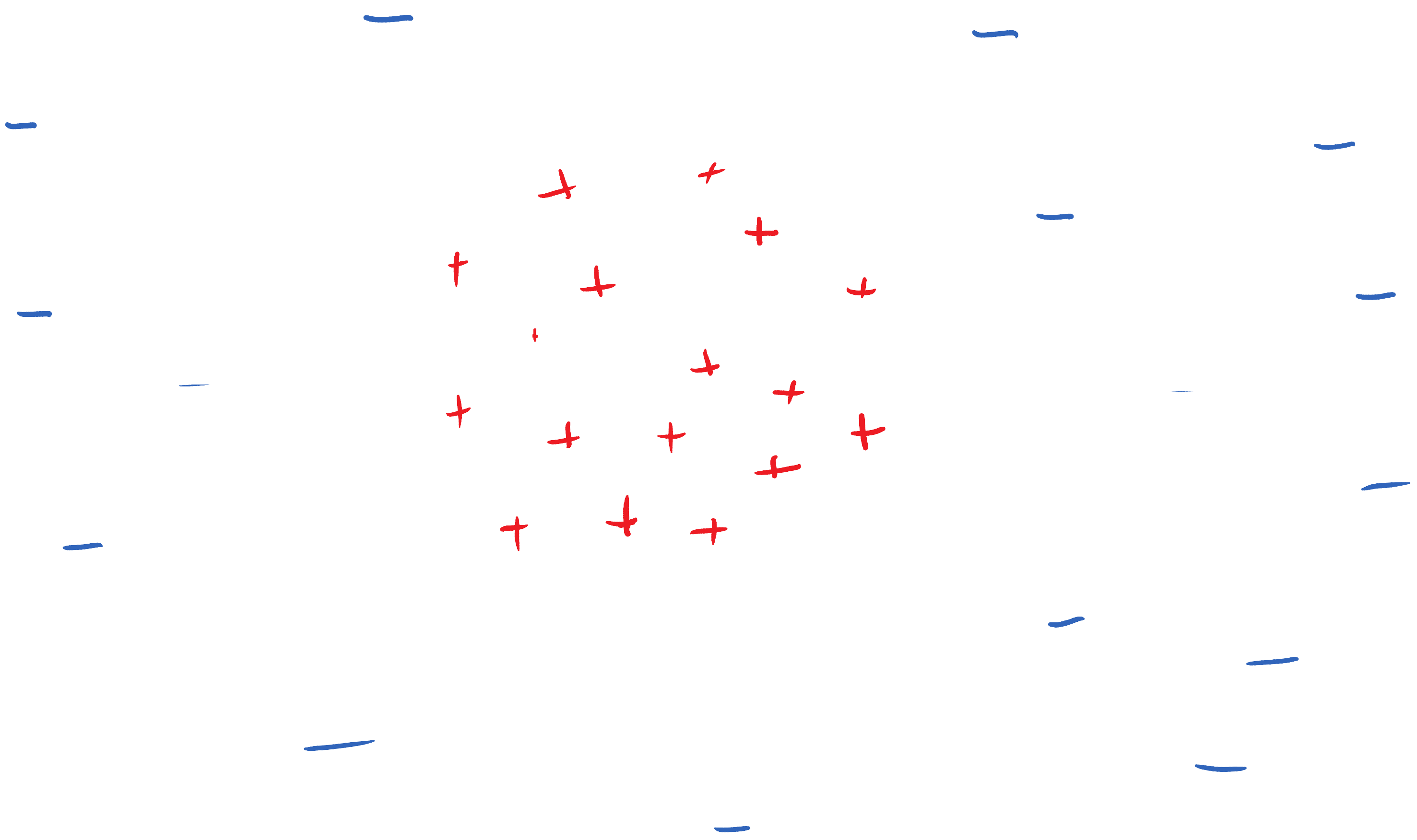
So now we just need to write a program to search the space of **w**’s and *b*’s to find the widest margin that matches all the datapoints. *How?*

Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton’s Method?

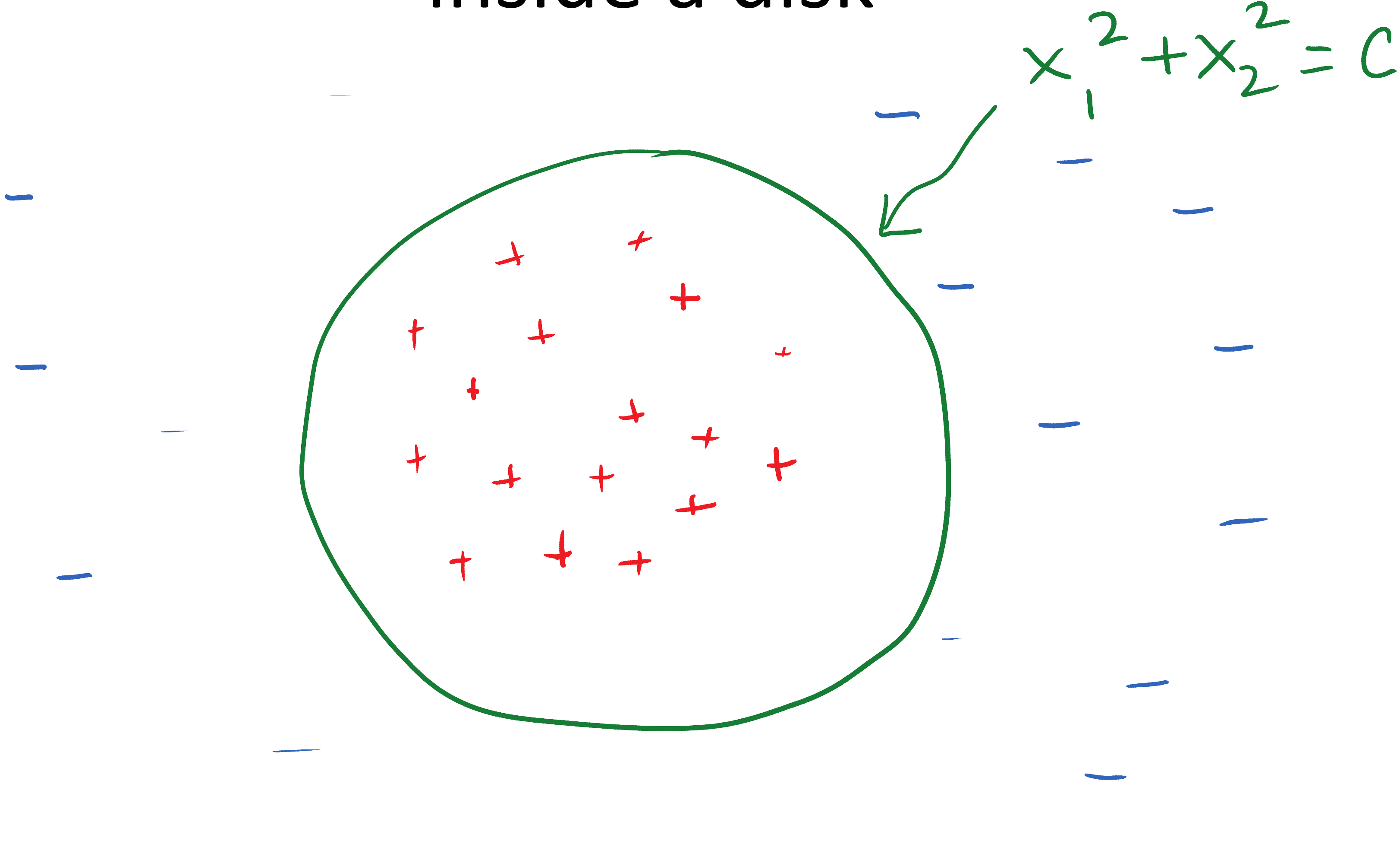
Some remarks..

- While the diagram corresponds to a linearly separable case, the idea can be generalized to a “soft margin SVM” where mistakes are allowed but penalized.
- Training an SVM is a convex optimization problem, so we are guaranteed that we can find the globally best solution. Various software packages are available such as LIBSVM, LIBLINEAR
- But what if the decision boundary is horribly non-linear? We use the “kernel trick”

Suppose the positive examples lie inside a disk



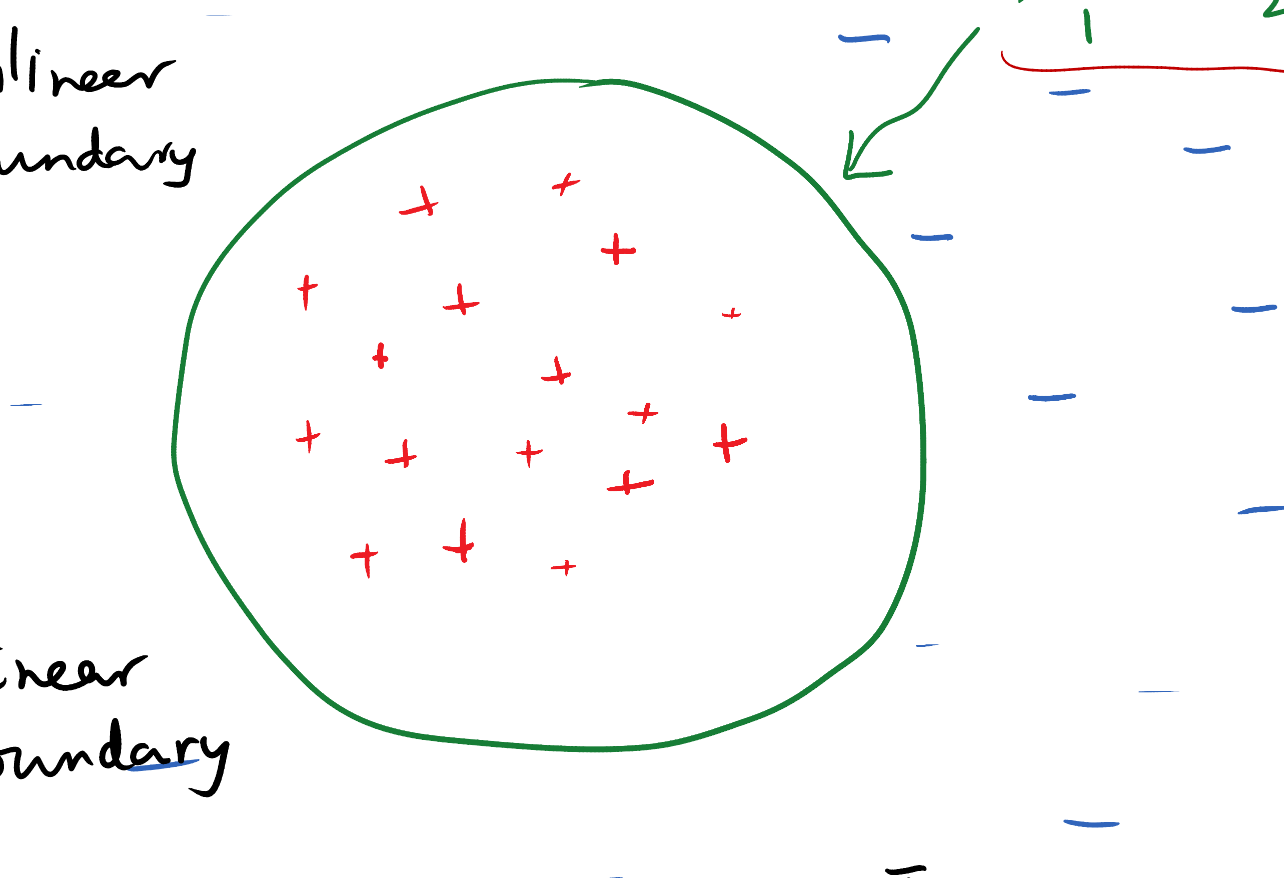
Suppose the positive examples lie inside a disk



We can construct a new higher-dimensional feature space where the boundary is linear

$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ nonlinear boundary

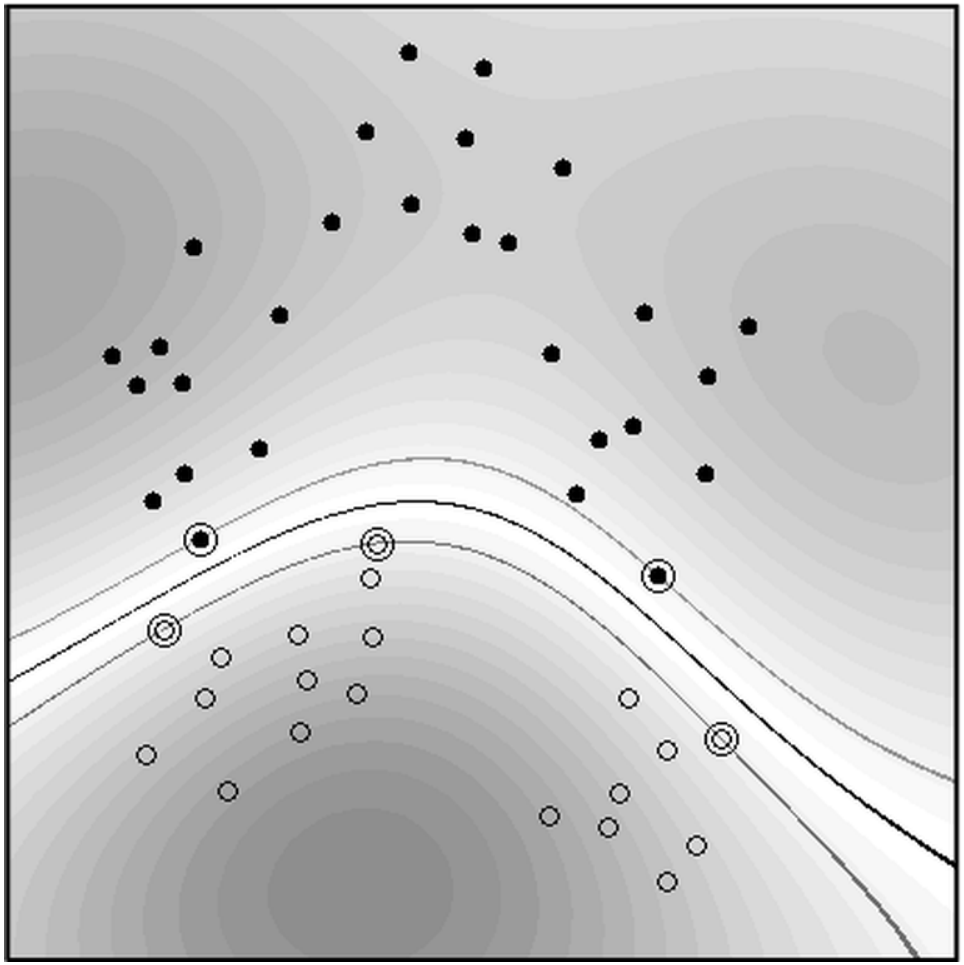
$x_1^2 + x_2^2 = C$



$\begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \end{bmatrix}$ linear boundary

$\underset{\sim}{x} \longrightarrow \Phi(\underset{\sim}{x})$

Kernel Support Vector Machines



Kernel :

- Inner Product in Hilbert Space
- Can Learn Non Linear Boundaries

Skipping math here, Sorry!

linear SVM, Kernelized SVM

Decision function is $\operatorname{sign}\left(\underline{h(x)}\right)$ where:

$\underset{\sim}{w} \cdot \underset{\sim}{x} + b = 0$

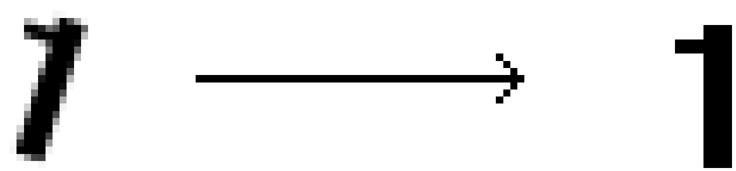
Linear: $$h(x) = w'x + b = \sum_{i=1}^{\#\mathrm{dim}} w_i x_i + b$$

Non-linear Using Kernel $$\underline{h(x)} = \underline{\sum_{j=1}^{\#\mathrm{sv}} \alpha^j \underbrace{K(x, x^j)} + b}$$

SUPPORT VECTORS

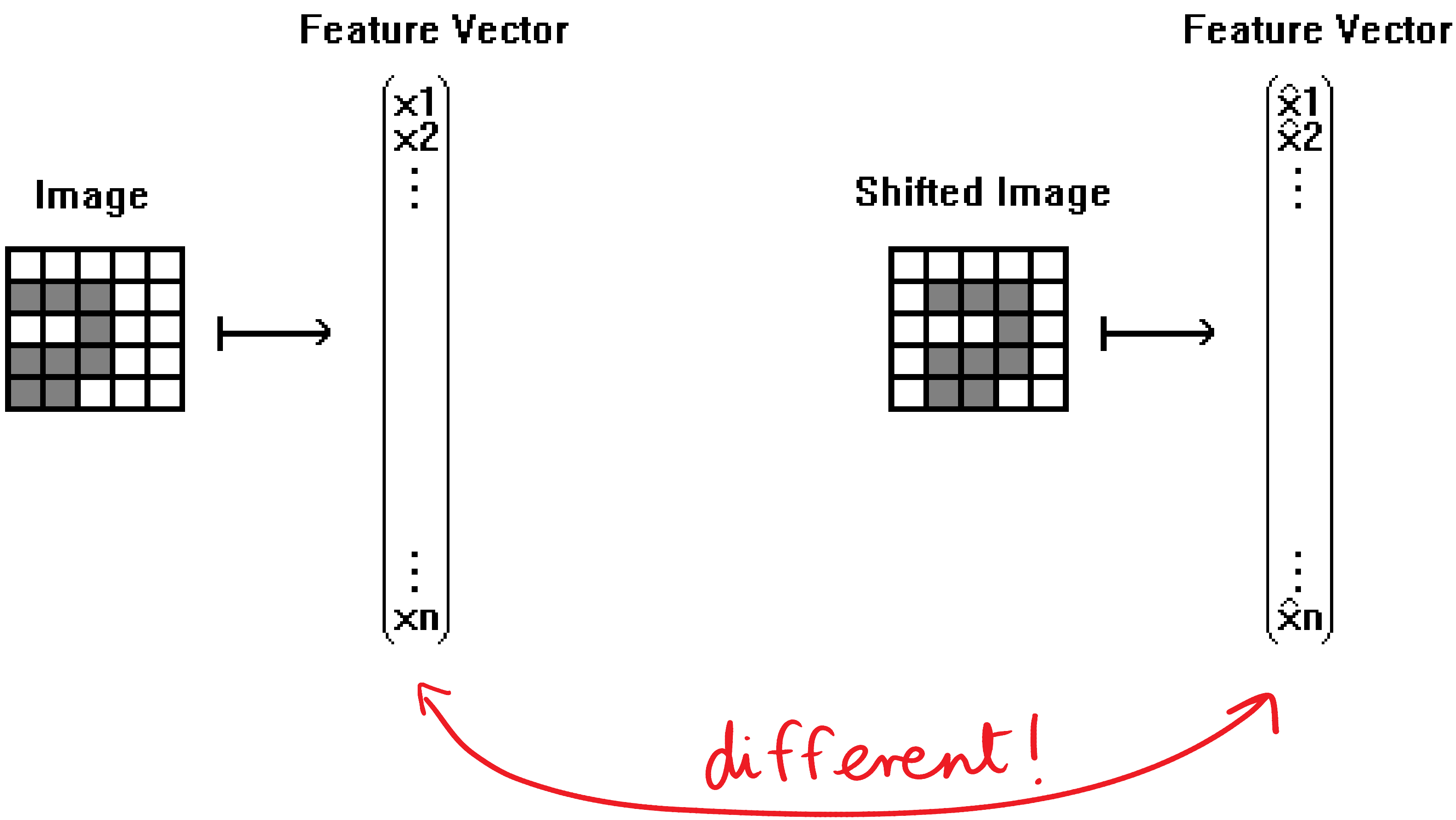
Transformation invariance
(or, why we love orientation histograms so much!)

- 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

Why is this a problem?



How do we build in transformational invariance?

- Augment the dataset
  - 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
  - Multi-scale scanning deals with scaling and translation

Orientation histograms

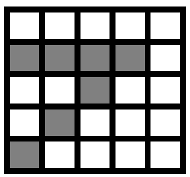


Image 1

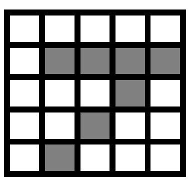
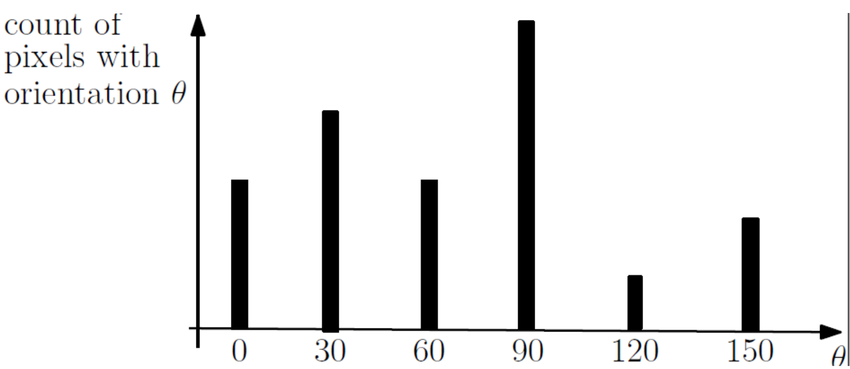


Image 2



- 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

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?
- Throwing away the spatial arrangement in the process of constructing an orientation histogram loses some information, but not that much.
- In addition, we can construct orientation histograms at different scales- the whole object, the object divided into quadrants, the object divided into even smaller blocks.

We compare histograms using the Intersection Kernel

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

$$K(a,b) = \sum_{i=1}^{n} min(a_i, b_i) \qquad \begin{matrix} a_i \geq 0 \\ b_i \geq 0 \end{matrix}$$

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

Intro. by Swain and Ballard 1991 to compare color histograms.
Odone et al 2005 proved positive definiteness.
Can be used directly as a kernel for an SVM.

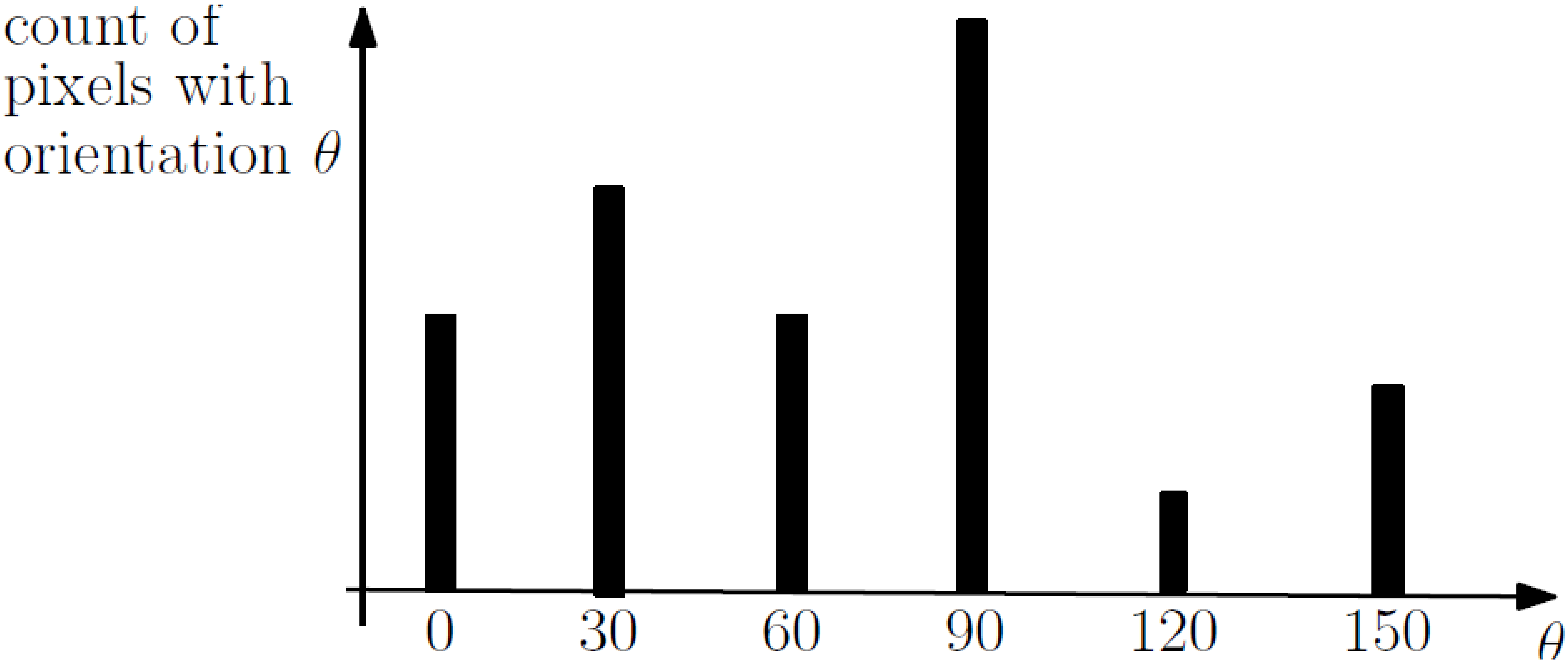
What is the Intersection Kernel?

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

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

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

Orientation histograms



What is 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

Intro. by Swain and Ballard 1991 to compare color histograms.
Odone et al 2005 proved positive definiteness.
Can be used directly as a kernel for an SVM.
Compare to $-\chi^2$

Digit Recognition using SVMS

Jitendra Malik

Lecture is based on

Maji & Malik (2009)

Digit recognition using SVMs

- What feature vectors should we use?
  - Pixel brightness values
  - Orientation histograms
- What kernel should we use for the SVM?
  - Linear
  - Intersection kernel
  - Polynomial
  - Gaussian Radial Basis Function

Some popular kernels in computer vision

x and y are two feature vectors

$$
\begin{aligned}
\text{LINEAR} \quad k_{lin}(\mathbf{x}, \mathbf{y}) &= \mathbf{x} \cdot \mathbf{y} \\
\text{HISTOGRAM} \quad k_{int}(\mathbf{x}, \mathbf{y}) &= \min(\mathbf{x}, \mathbf{y}) \\
k_{poly}(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + 1)^5 \\
k_{rbf}(\mathbf{x}, \mathbf{y}) &= \exp(-\gamma ||\mathbf{x} - \mathbf{y}||^2)
\end{aligned}
$$

GENERAL NON-LINEAR

Kernelized SVMs slow to evaluate

Decision function is $\text{sign}\left(h(x)\right)$ where:

Feature vector to evaluate

Sum over all support vectors

Kernel Evaluation

Feature corresponding to a support vector *l*

Arbitrary Kernel

$$h(x) = \sum_{j=1}^{\#\text{sv}} \alpha^j K(x, x^j) + b$$

Histogram Intersection Kernel

$$h(x) = \sum_{j=1}^{\#\text{sv}} \left(\alpha^j \sum_{i=1}^{\#\text{dim}} \min(x_i, x_i^j) \right) + b$$

Cost: # Support Vectors x Cost of kernel computation

For a linear kernel, $h(x)$ simplifies

$$\sum \alpha^j x \cdot x_j + b = x \cdot \underbrace{\left(\sum \alpha^j x_j\right)}_{\text{PRECOMPUTED}} + b$$

Complexity considerations

- Linear kernels are the fastest
- Intersection kernels are nearly as fast, using the "Fast Intersection Kernel" (Maji, Berg & Malik, 2008)
- Non-linear kernels such as the polynomial kernel or Gaussian radial basis functions are the slowest, because of the need to evaluate kernel products with each support vector. There could be thousands of support vectors!

Raw pixels do not make a good feature vector

- Each digit in the MNIST DATABASE of handwritten digits is a 28 x 28 pixel grey level image.

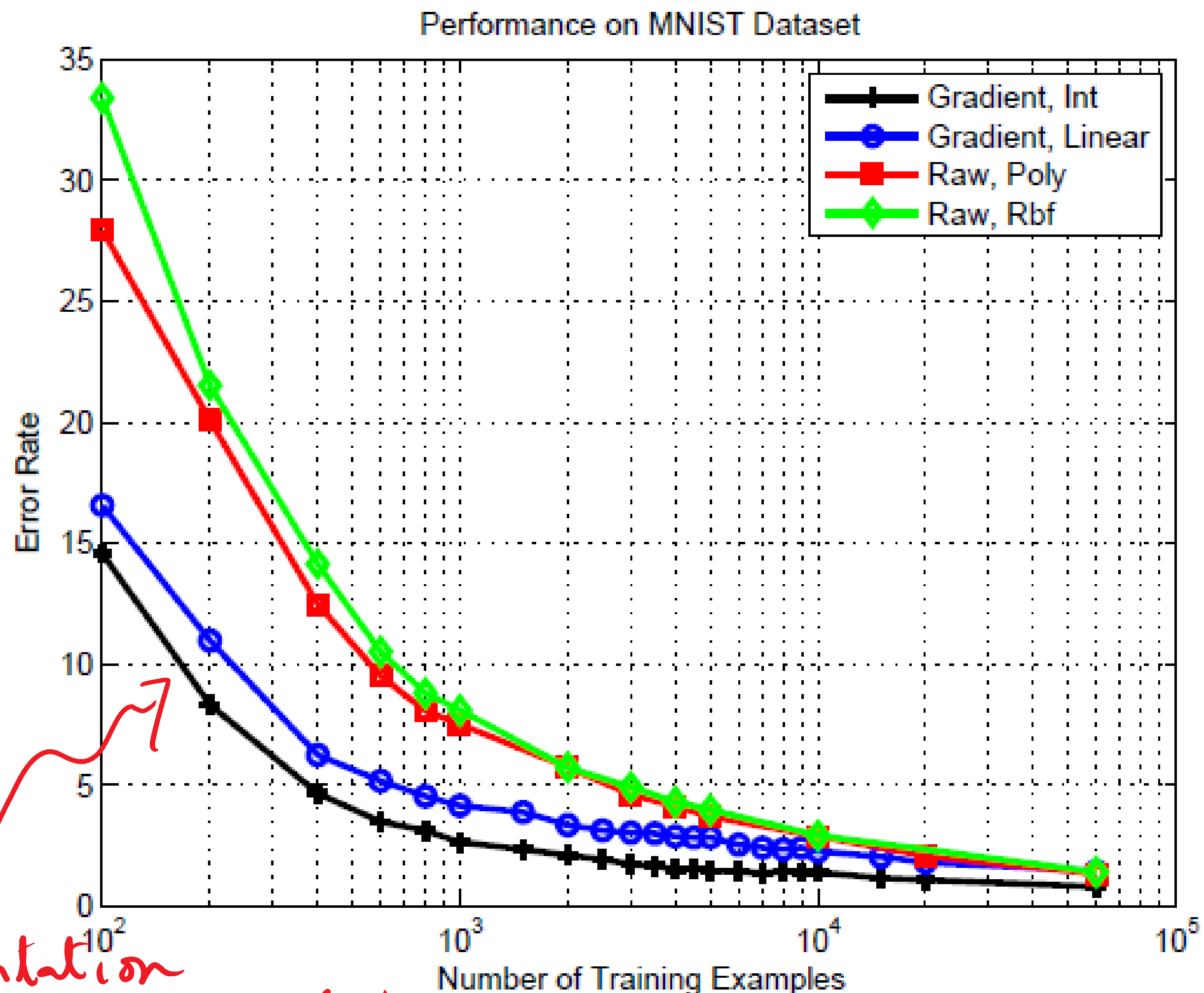
Complexity	kernel	error rate(%)
$\mathcal{O}(1)$	linear	15.38
	int	13.29
$\mathcal{O}(\#SV)$	poly	7.41
	rbf	8.10

kernel	error rate(%)
linear	14.84
int	9.02
poly	7.71
rbf	6.57

Table 1: Error rates on the MNIST dataset using raw pixels(left) and pyramid of raw pixels(right). Only the first 1000 examples were used for training.

Nonlinear kernels give smaller error rates, but at considerably higher computational expense

Error rates vs. the number of training examples



Orientation histograms rule!

Technical details on orientation computation

1. **Oriented Derivative Filter** The input grayscale image is convolved with filters which respond to horizontal and vertical gradients from which the magnitude and orientation is computed. Let $rh(p)$ and $rv(p)$ be the response in the horizontal and vertical direction at a pixel p respectively, then the magnitude $m(p)$ and the angle $a(p)$ of the pixel is given by :

$$m(p) = \sqrt{rh(p)^2 + rv(p)^2} \tag{5}$$

$$a(p) = \operatorname{atan2}\left(rh(p), rv(p)\right) \in [0, 360) \tag{6}$$

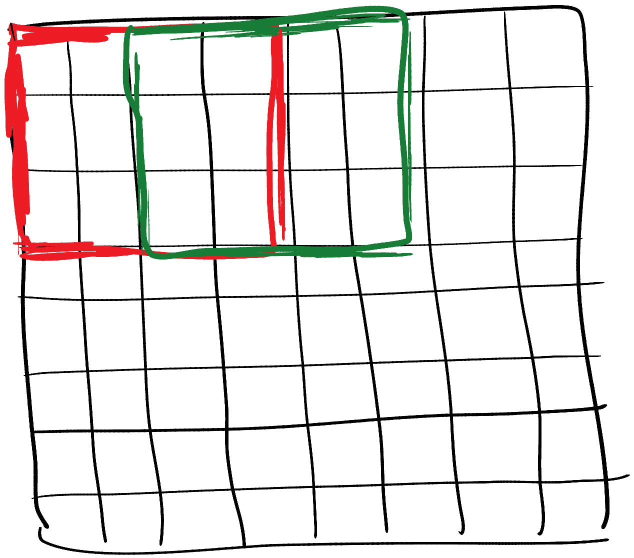
   We experiment with tap filters, Sobel and oriented Gaussian derivative (OGF) filters.

2. **Signed vs. Unsigned** The orientation could be signed $(0-360)$ or unsigned $(0-180)$. The signed gradient distinguishes between black to white and white to black transitions which might be useful for digits.

3. **Number of Orientation Bins** The orientation at each pixel is binned into a discrete set of orientations by linear interpolation between bin centers to avoid aliasing.

The best choice is determined experimentally

Details of histogram computation



- Compute histogram on a c × c block
- Shift block by c/2 and recompute histogram
- All the block histograms are stacked in a long feature vector.

Block sizes c = 14, 7, 4 were used.

Final error rate = 0.79%

The 79 Errors

Some key references on orientation histograms

- D. Lowe, ICCV 1999, SIFT
- A. Oliva & A. Torralba, IJCV 2001, GIST
- A. Berg & J. Malik, CVPR 2001, Geometric Blur
- N. Dalal & B. Triggs, CVPR 2005, HOG
- S. Lazebnik, C. Schmid & J. Ponce, CVPR 2006, Spatial Pyramid Matching

Code is available for all of these approaches. Go ahead and explore!

Kernelized SVMs slow to evaluate

Decision function is $\text{sign}\left(h(x)\right)$ where:

Feature vector to evaluate

Sum over all support vectors

Kernel Evaluation

Feature corresponding to a support vector *l*

Arbitrary Kernel

$$h(x) = \sum_{j=1}^{\#\text{sv}} \alpha^j K(x, x^j) + b$$

Histogram Intersection Kernel

$$h(x) = \sum_{j=1}^{\#\text{sv}} \left(\alpha^j \sum_{i=1}^{\#\text{dim}} \min(x_i, x_i^j) \right) + b$$

SVM with Kernel Cost:

Support Vectors x Cost of kernel comp.

IKSVM Cost:

Support Vectors x # feature dimensions

Randomized decision trees (a.k.a. Random Forests)

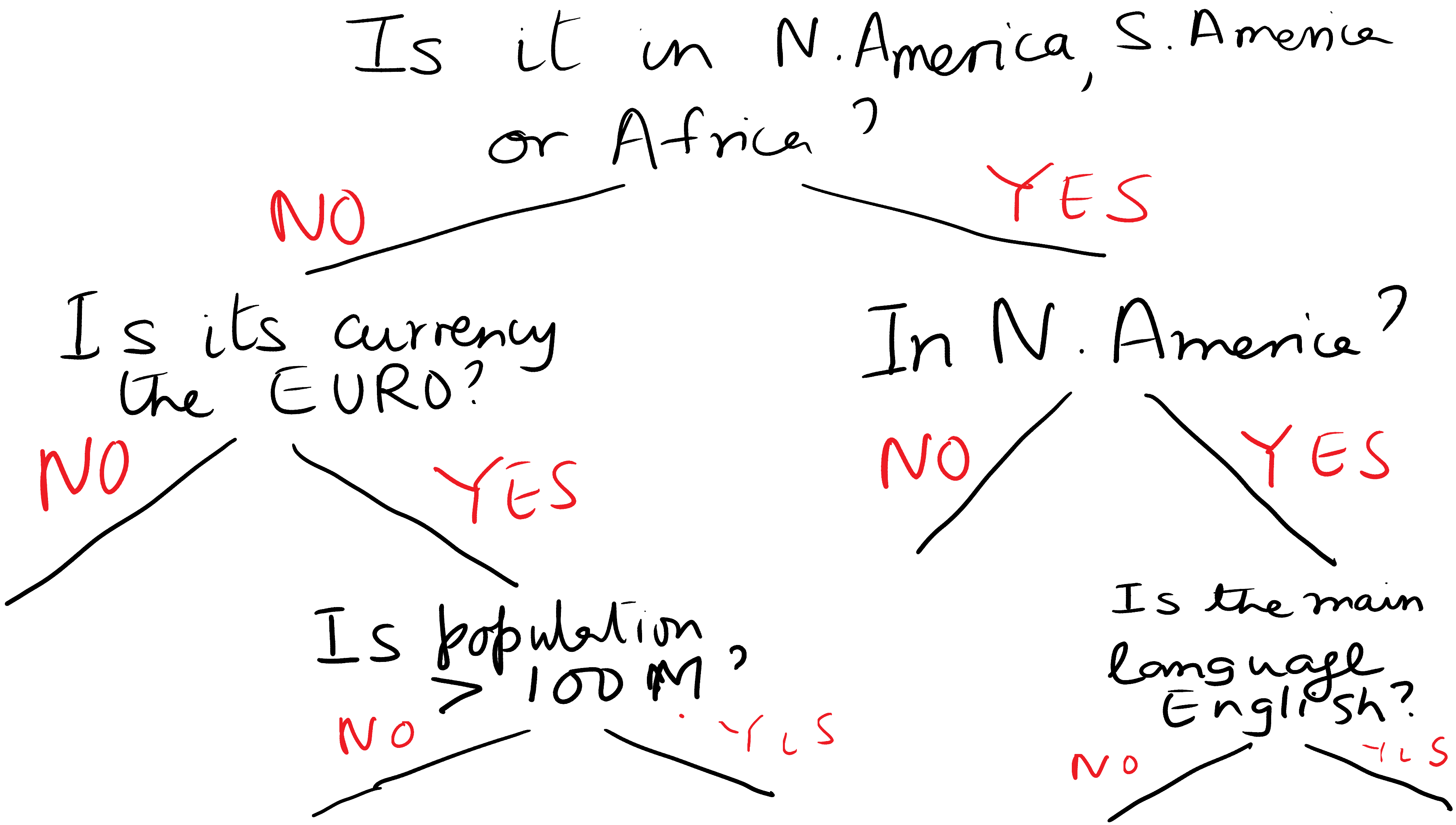
Jitendra Malik

Two papers

- Y. Amit, D. Geman & K. Wilder, Joint induction of shape features and tree classifiers, IEEE Trans. on PAMI, Nov. 1997.(digit classification)
- J. Shotton et al, Real-time Human Pose Recognition in Parts from Single Depth Images, IEEE CVPR, 2011. (describes the algorithm used in the Kinect system)

What is a decision tree?

What is a decision tree?



Decision trees for Classification

- Training time
  - Construct the tree, i.e. pick the questions at each node of the tree. Typically done so as to make each of the child nodes “purer”(lower entropy). Each leaf node will be associated with a set of training examples
- Test time
  - Evaluate the tree by sequentially evaluating questions, starting from the root node. Once a particular leaf node is reached, we predict the class to be the one with the most examples(from training set) at this node.

Training is slow, testing is fast

Amit, Geman & Wilder’s approach

- Some questions are based on whether certain “tags” are found in the image. Crudely, think of these as edges of particular orientation.
- Other questions are based on spatial relationships between pairs of tags. An example might be whether a vertical edge is found above and to the right of an horizontal edge

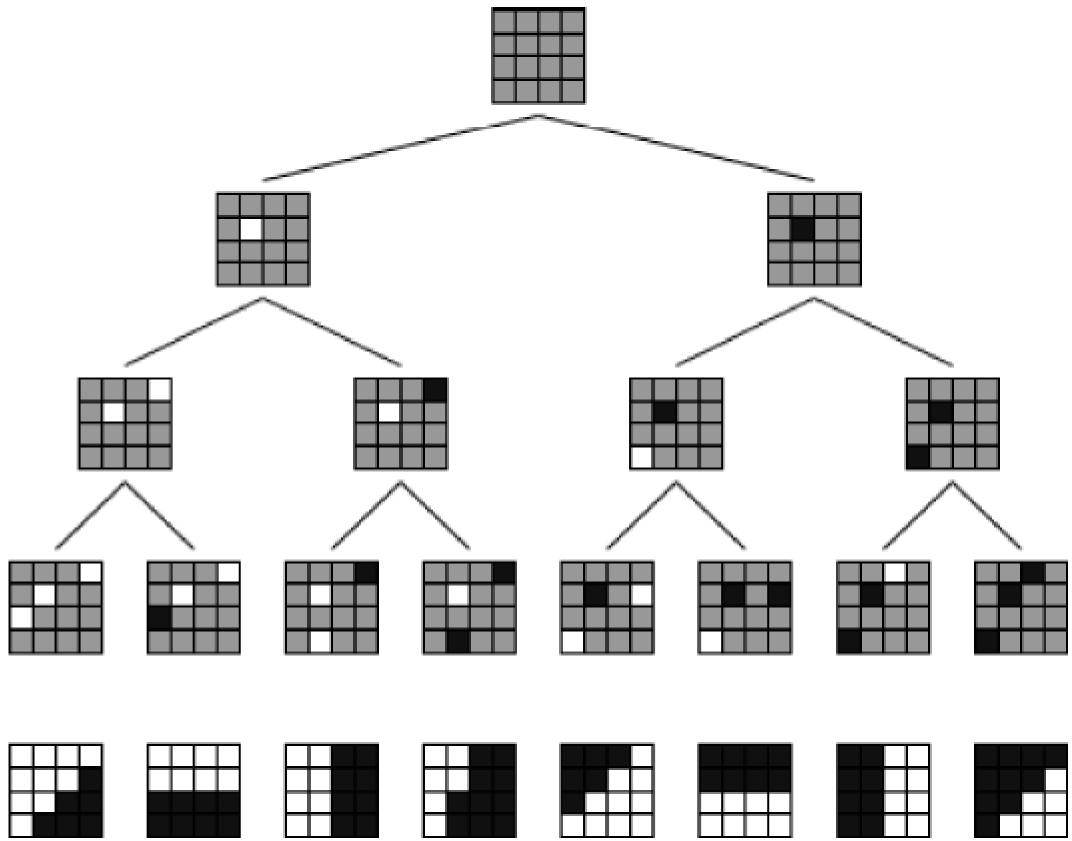


Fig. 1. First three tag levels with most common configurations.

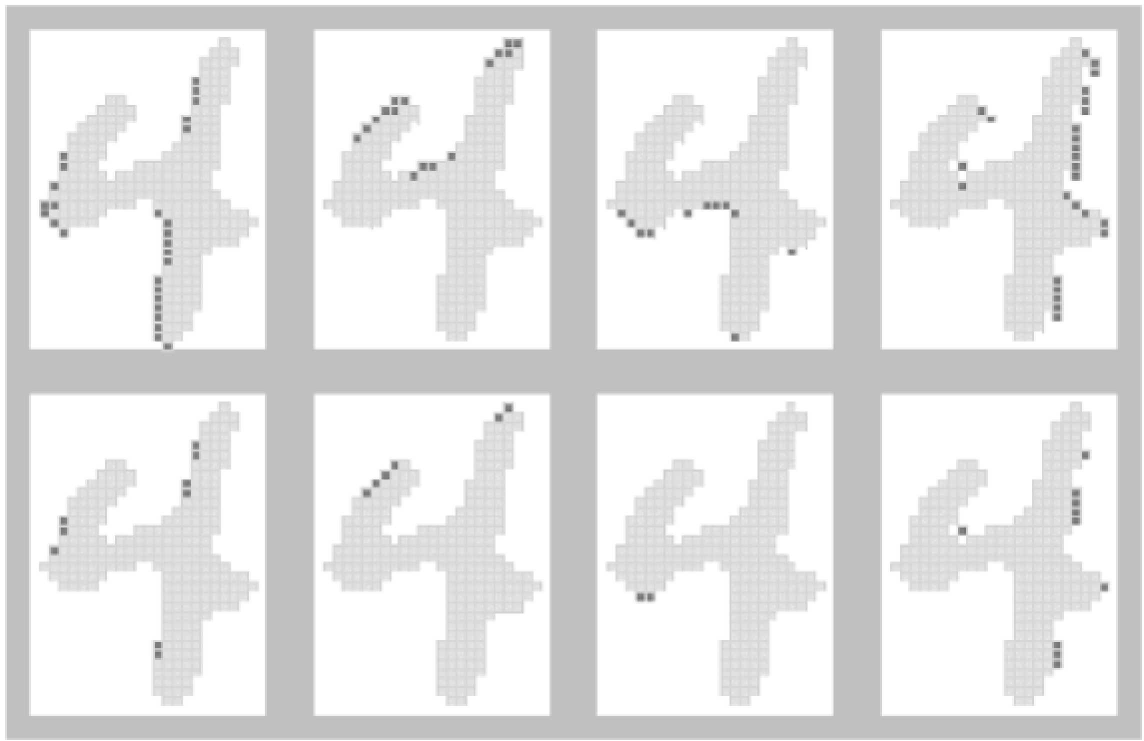


Fig. 2. Top: All instances of four depth three tags. Bottom: All instances of four depth five tags.

An example of such an arrangement

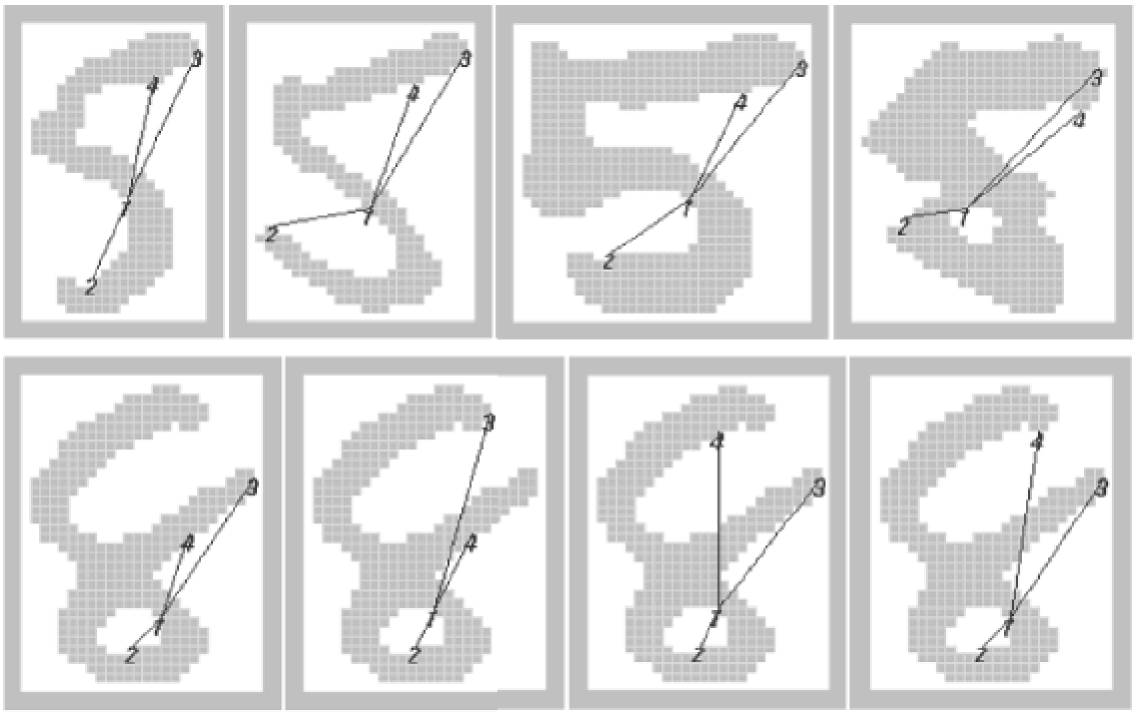


Fig. 3. Top row: Instances of a geometric arrangement in several 5s. Bottom row: Several instances of the geometric arrangement in one 8.

This arrangement does not discriminate a 5 from a 8. So we ask another question

Additional questions “grow” the arrangement

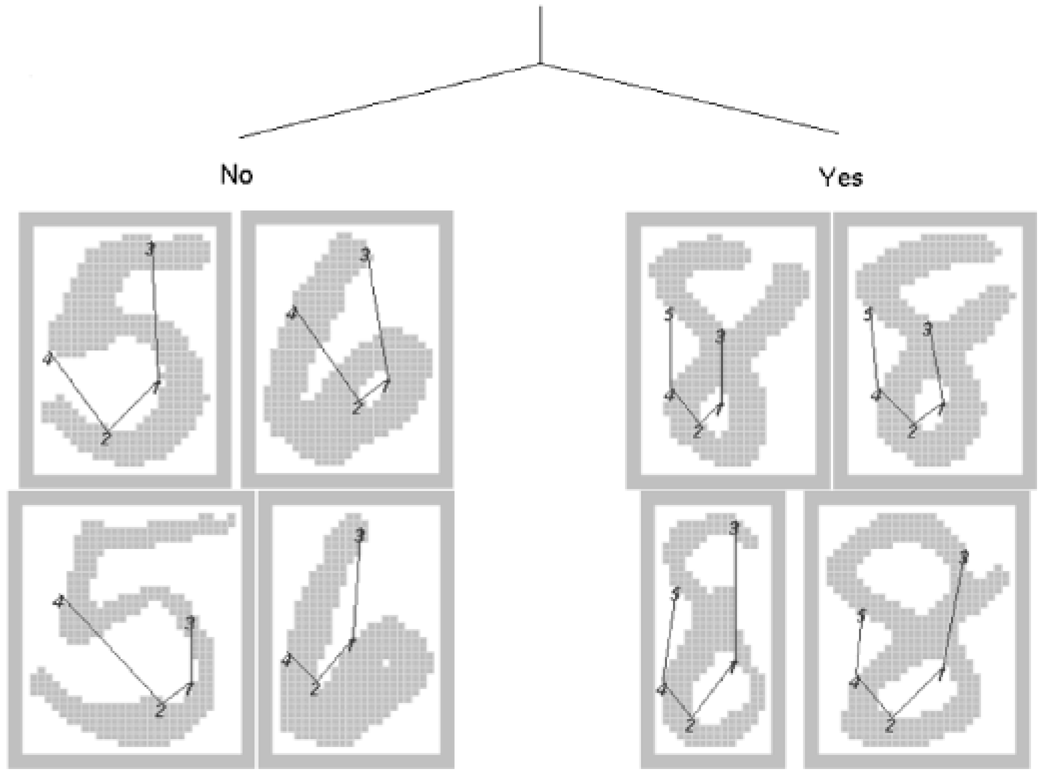


Fig. 4. Example of node-splitting in a typical digit tree; the query involves adding a fifth tag (vertex) to the pending arrangement. Specifically, the proposed arrangement adds a fifth vertex and a fourth relation to the existing graph which has four vertices and three relations.

Multiple randomized trees

- It turns out that using a single tree for classification doesn't work too well. Error rates are around 7% or so.
- But if one trains multiple trees (different questions) and averages the predicted posterior class probabilities, error rates fall below 1%
- Powerful general idea- now called "Random Forests"

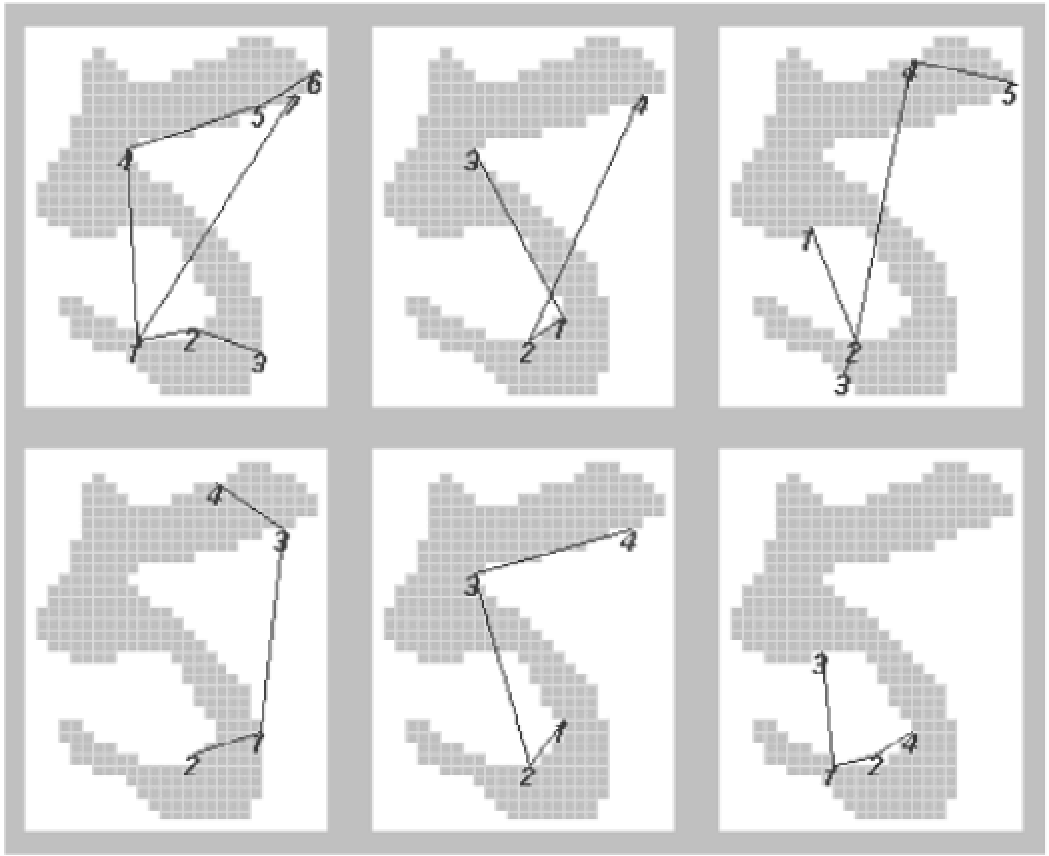


Fig. 5. Arrangements found in an image at terminal nodes of six different trees.

The Microsoft Kinect system uses a similar approach…

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. *Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.*

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

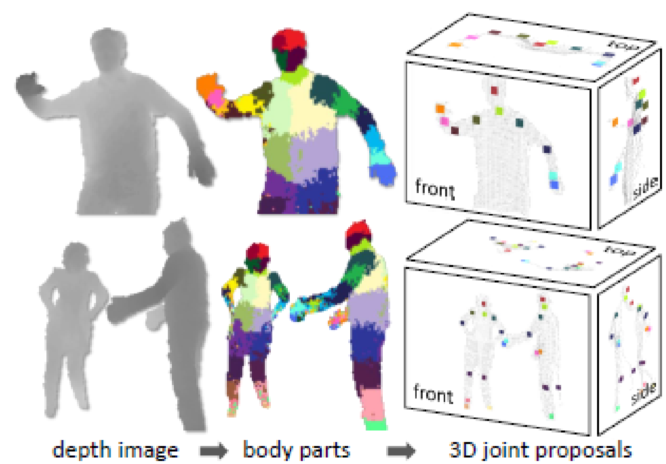


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

joints of interest. Reprojecting the inferred parts into world

The idea behind Tangent Distance (Simard et al)

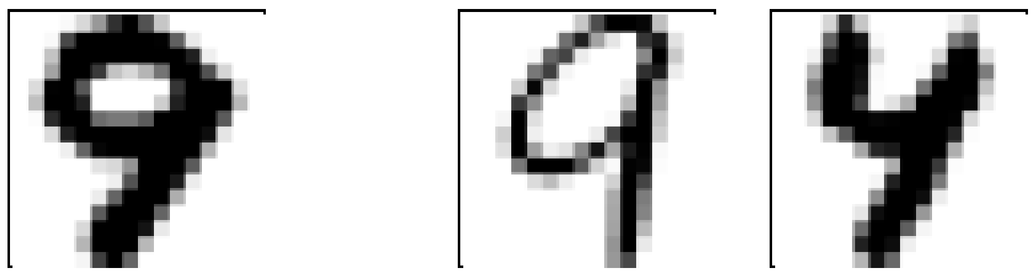


Fig. 1. According to the Euclidean distance the pattern to be classified is more similar to prototype B. A better distance measure would find that prototype A is closer because it differs mainly by a rotation and a thickness transformation, two transformations which should leave the classification invariant.

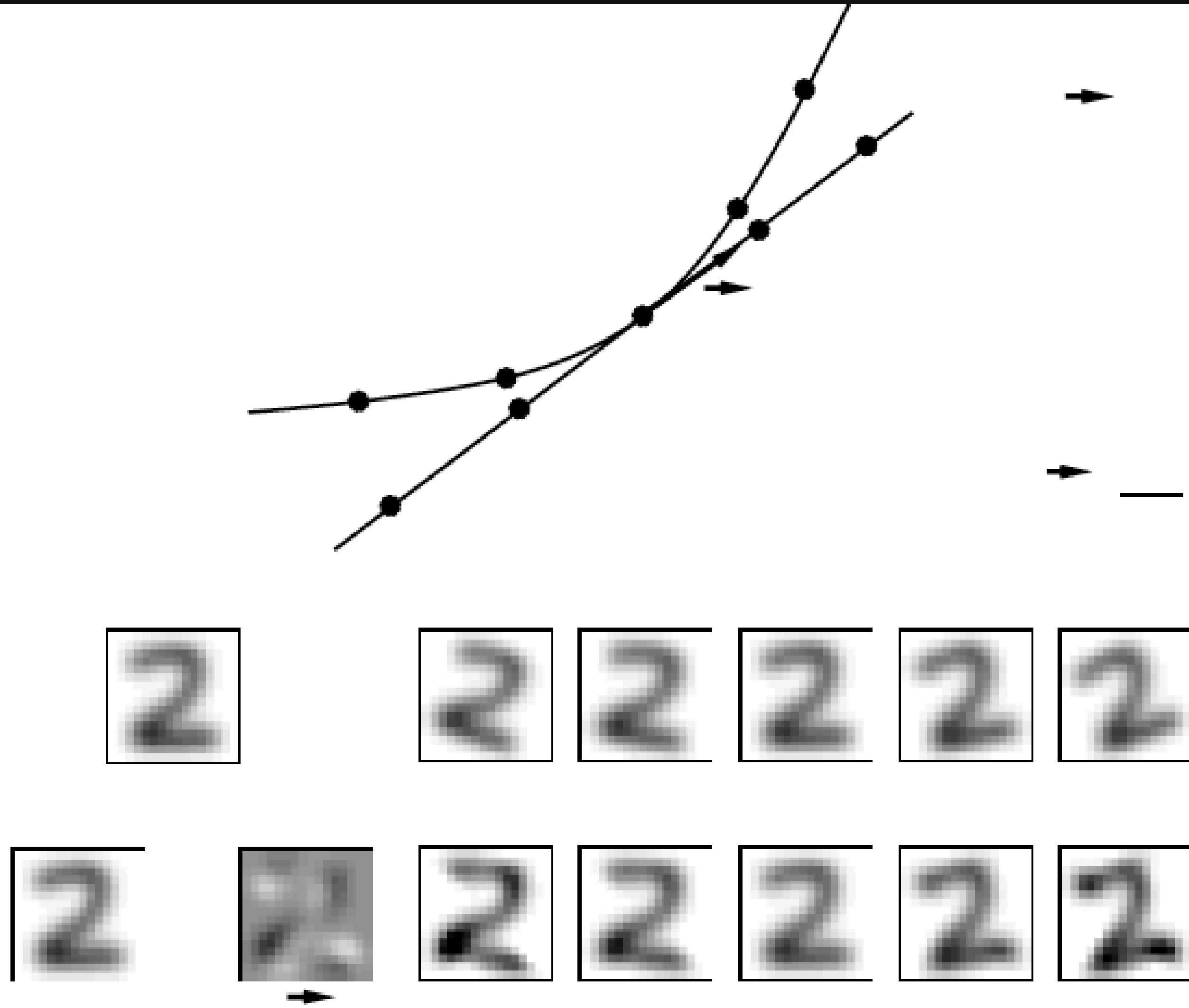


Fig. 2. Top: Representation of the effect of the rotation in pixel space. Middle: Small rotations of an original digitized image of the digit "2", for different angle values of α. Bottom: Images obtained by moving along the tangent to the transformation curve for the same original digitized image P by adding various amounts (α) of the tangent vector T.

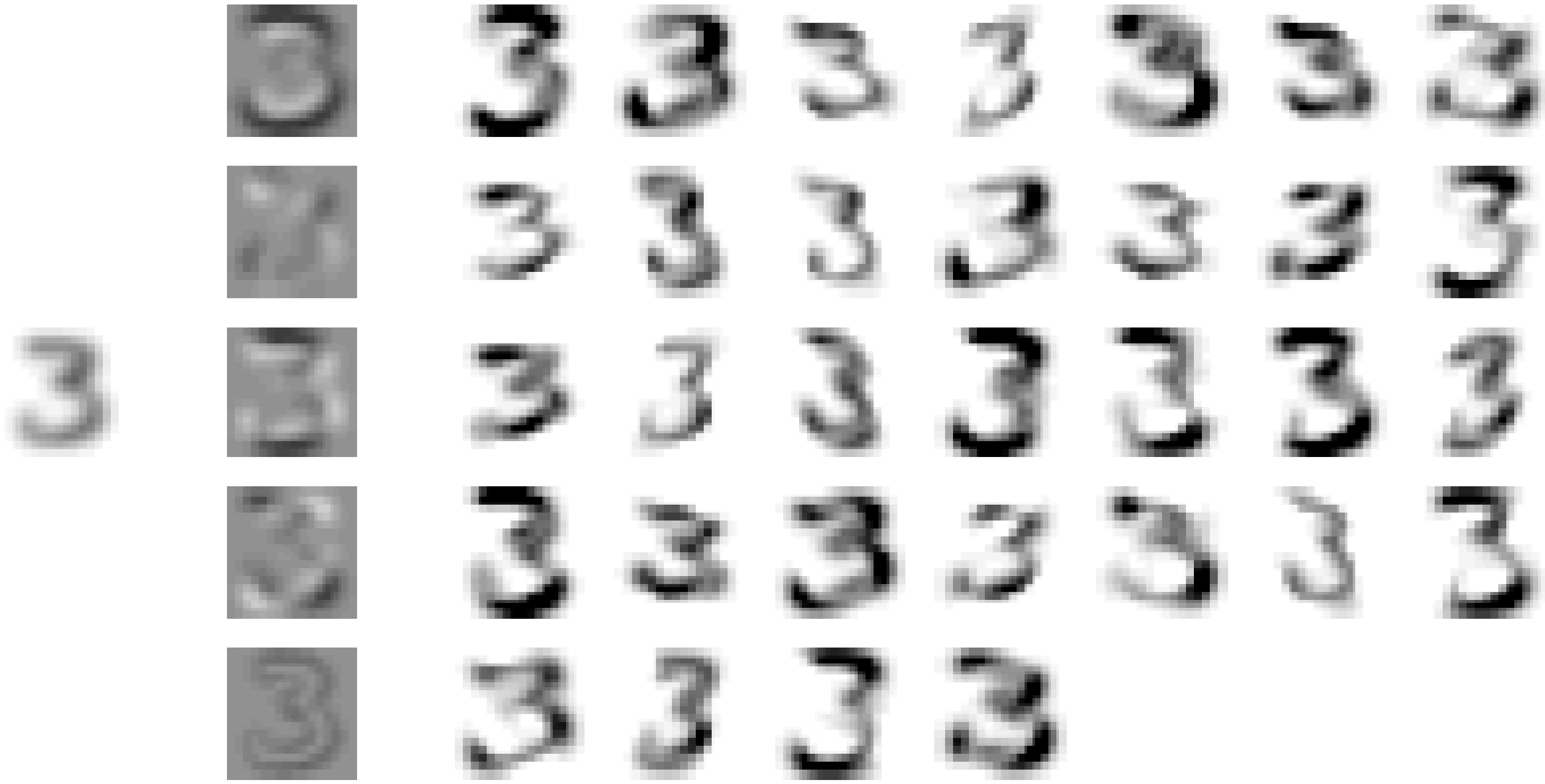


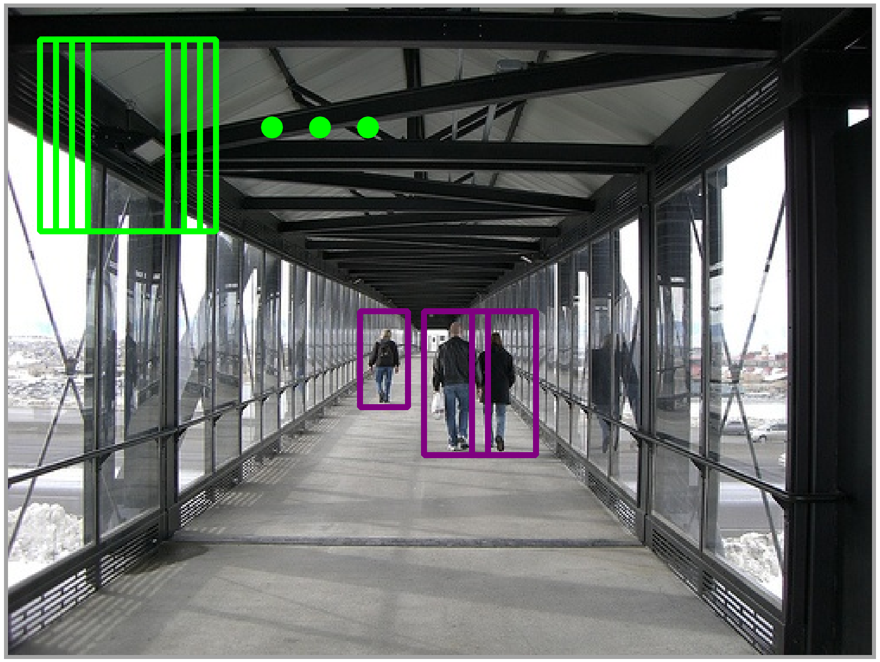
Fig. 6. Left: Original image. Middle: 5 tangent vectors corresponding respectively to the 5 transformations: scaling, rotation, expansion of the X axis while compressing the Y axis, expansion of the first diagonal while compressing the second diagonal and thickening. Right: 32 points in the tangent space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

Fast Intersection Kernel
Maji, Berg & Malik, 2008

- What is Intersection Kernel SVM?
  - Trick to make it fast (exact)
  - Trick to make it very fast (approximate)
- Generalization of linear classifiers
  - Reinterpret the approximate IKSVM
  - Fast training

details in
Maji & Berg, 2009

Detection: Is this an X?



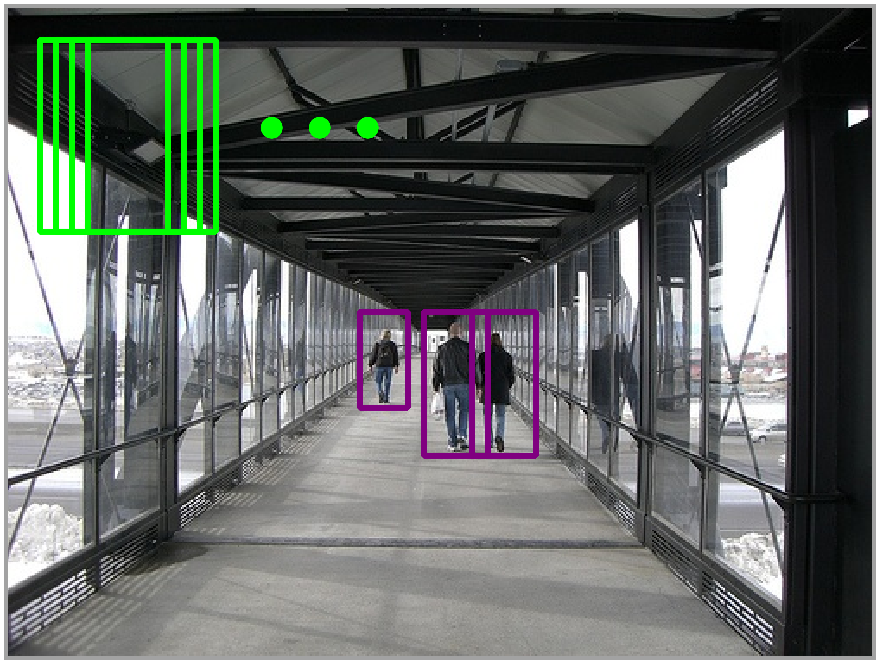
Boosted dec. trees, cascades
+ Very fast evaluation
- Slow training (esp. multi-class)

Linear SVM
+ Fast evaluation
+ Fast training
- Need to find good features

Non-linear kernelized SVM
+ Better class. acc. than linear
. Medium training
- Slow evaluation

Ask this question over and over again,
varying position, scale, multiple categories…
Speedups: hierarchical, early reject, feature sharing,
but same underlying question!

Detection: Is this an X?



Boosted dec. trees, cascades
+ Very fast evaluation
- Slow training (esp. multi-class)

Linear SVM
+ Fast evaluation
+ Fast training
- Need to find good features

This work

Non-linear kernelized SVM
+ Better class. acc. than linear
. Medium training
- Slow evaluation

Ask this question over and over again,
varying position, scale, multiple categories…
Speedups: hierarchical, early reject, feature sharing,
but same underlying question!

Demonstration of Positive Definiteness

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

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

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

Demonstration of Positive Definiteness

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

$$K(a,b) = \sum_{i=1}^{n} min(a_i, b_i) \qquad \begin{aligned} a_i &\geq 0 \\ b_i &\geq 0 \end{aligned}$$

To see that $min(a_i, b_i)$ is positive definite,

represent *a*, *b* in "Unary", *n* is written as *n* ones in a row:

$$min(a_i, b_i) = \left\langle a_{i\,\mathrm{unary}}, b_{i\,\mathrm{unary}} \right\rangle$$

Demonstration of Positive Definiteness

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

$$K(a,b) = \sum_{i=1}^{n} min(a_i, b_i) \qquad \begin{array}{r} a_i \geq 0 \\ b_i \geq 0 \end{array}$$

To see that $min(a_i, b_i)$ is positive definite,

represent *a*, *b* in "Unary", *n* is written as *n* ones in a row:

$$min(a_i, b_i) = \langle a_{i\,\mathrm{unary}}, b_{i\,\mathrm{unary}} \rangle$$

$$min(3,5) = \langle (1,1,1,0,0), (1,1,1,1,1) \rangle = 3$$

The Trick

Decision function is $\mathrm{sign}\,(h(x))$ where:

$$
\begin{aligned}
h(x) &= \sum_{j=1}^{\#\mathrm{sv}} \alpha^j \left(\sum_{i=1}^{\#\mathrm{dim}} \min(x_i, x_i^j) \right) + b \\
&= \sum_{i=1}^{\#\mathrm{dim}} \left(\sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) \right) + b \\
&= \sum_{i=1}^{\#\mathrm{dim}} h_i(x_i)
\end{aligned}
$$

$$
\begin{aligned}
h_i(x_i) &= \sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) + b \\
&= \sum_{x_i^j < x_i} \alpha^j x_i^j + \left(\sum_{x_i^j \geq x_i} \alpha^j \right) x_i
\end{aligned}
$$

Just sort the support vector values in each coordinate, and pre-compute

To evaluate, find position of x_i in the sorted support vector values x_i^j (cost: log #sv)
look up values, multiply & add

The Trick

~~#support vectors x #dimensions~~

log(#support vectors) x #dimensions

Decision function is $\mathrm{sign}\left(h(x)\right)$ where:

$$
\begin{aligned}
h(x) &= \sum_{j=1}^{\#\mathrm{sv}} \alpha^j \left(\sum_{i=1}^{\#\mathrm{dim}} \min(x_i, x_i^j) \right) + b \\
&= \sum_{i=1}^{\#\mathrm{dim}} \left(\sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) \right) + b \\
&= \sum_{i=1}^{\#\mathrm{dim}} h_i(x_i)
\end{aligned}
$$

Just sort the support vector values in each coordinate, and pre-compute

$$
\begin{aligned}
h_i(x_i) &= \sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) + b \\
&= \sum_{x_i^j < x_i} \alpha^j x_i^j + \left(\sum_{x_i^j \geq x_i} \alpha^j \right) x_i
\end{aligned}
$$

To evaluate, find position of x_i in the sorted support vector values x_i^j (cost: log #sv)
look up values, multiply & add

~~#support vectors x #dimensions~~

~~log(#support vectors) x #dimensions~~

constant x #dimensions

The Trick 2

Decision function is $\mathrm{sign}\left(h(x)\right)$ where:

$$\begin{aligned} h(x) &= \sum_{i=1}^{\#\mathrm{dim}} \left(\sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) \right) + b \\ &= \sum_{i=1}^{\#\mathrm{dim}} h_i(x_i) \end{aligned}$$

For IK h_i is piecewise linear, and quite smooth, blue plot. We can *approximate* with fewer uniformly spaced segments, red plot. Saves time & space!

$$\begin{aligned} h_i(x_i) &= \sum_{j=1}^{\#\mathrm{sv}} \alpha^j \min(x_i, x_i^j) + b \\ &= \sum_{x_i^j < x_i} \alpha^j x_i^j + \left(\sum_{x_i^j \geq x_i} \alpha^j \right) x_i \end{aligned}$$