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%)
- SVM on orientation histograms(Maji & Malik, 0.8%)

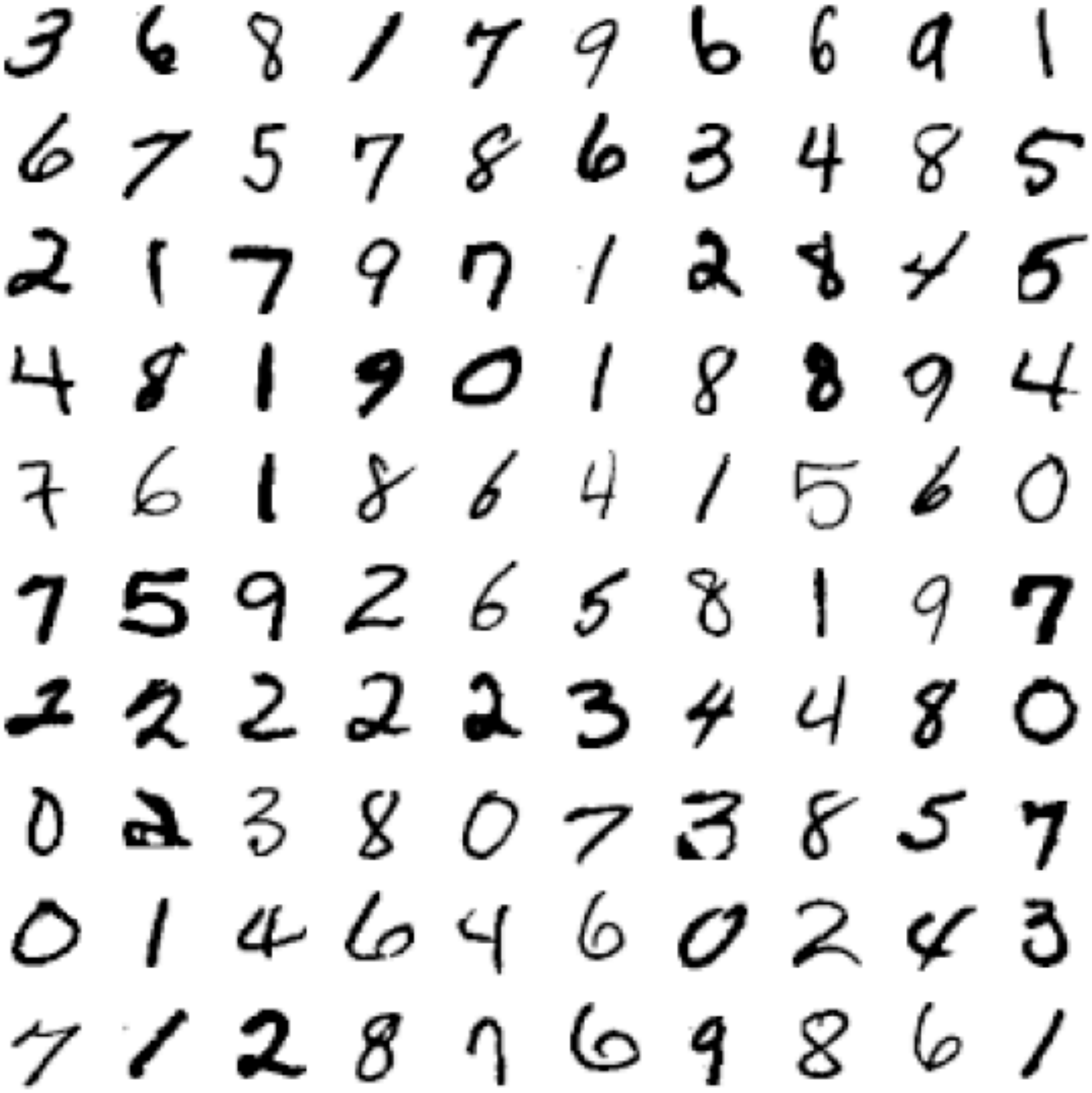


Fig. 4. Size-normalized examples from the MNIST database.

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

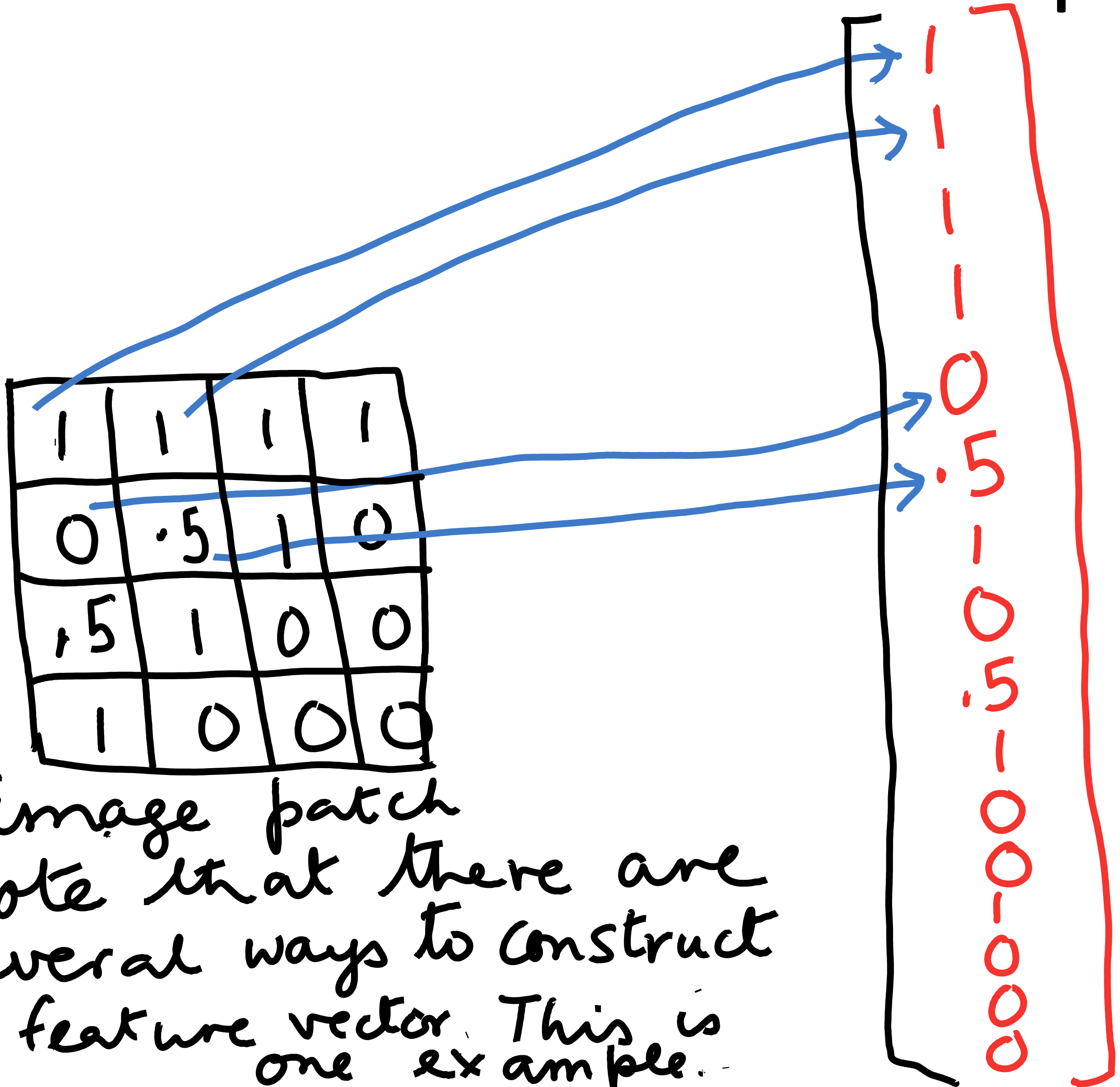
- Training time
  - Compute feature vectors for positive and negative examples of image patches
  - Train a classifier
- Test Time
  - Compute feature vector on image patch
  - Evaluate classifier

Let us take an example...

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

image patch

Let us take an example...



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

image patch

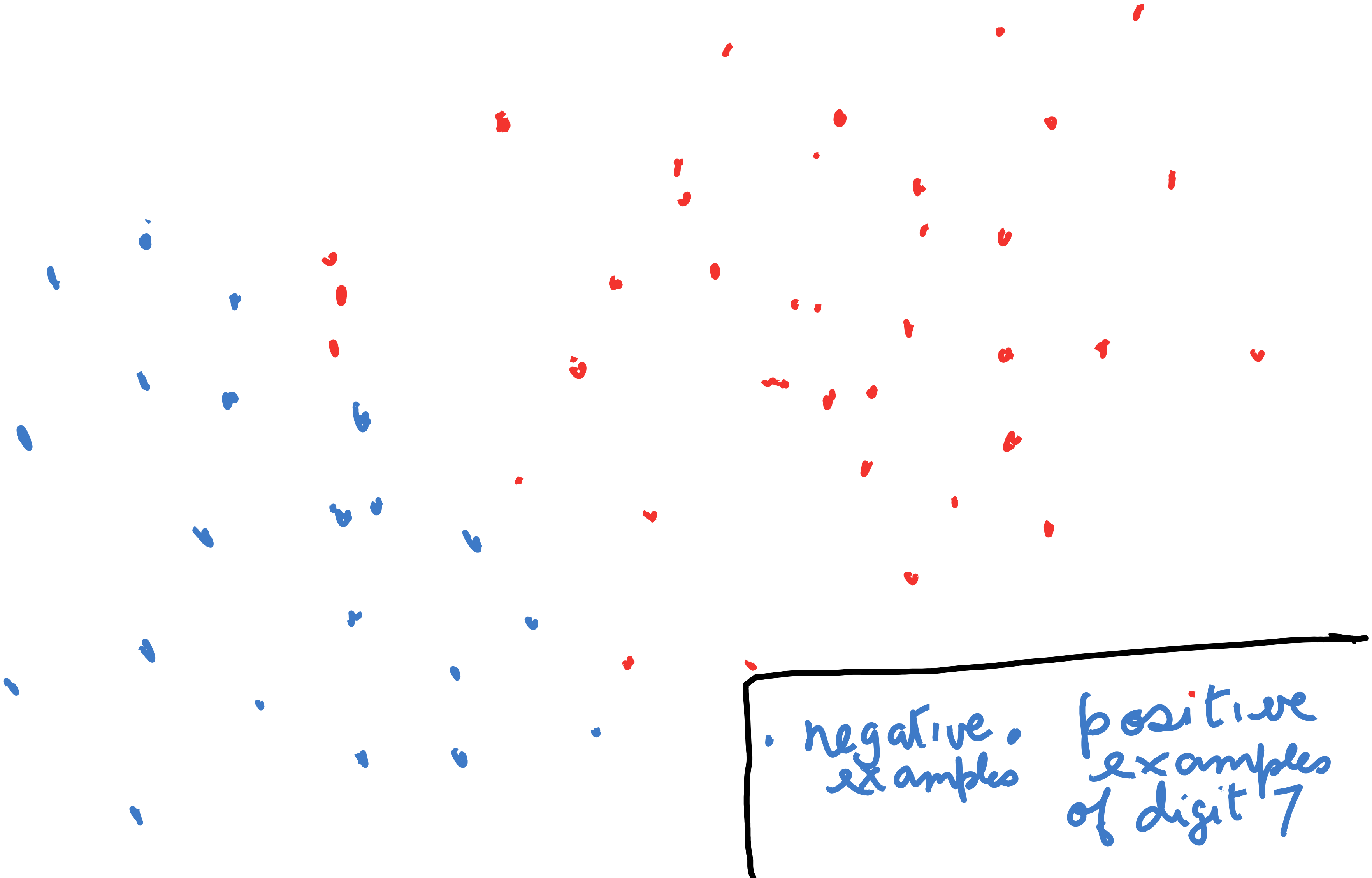
$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ .5 \\ 1 \\ 0 \\ .5 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Feature vector

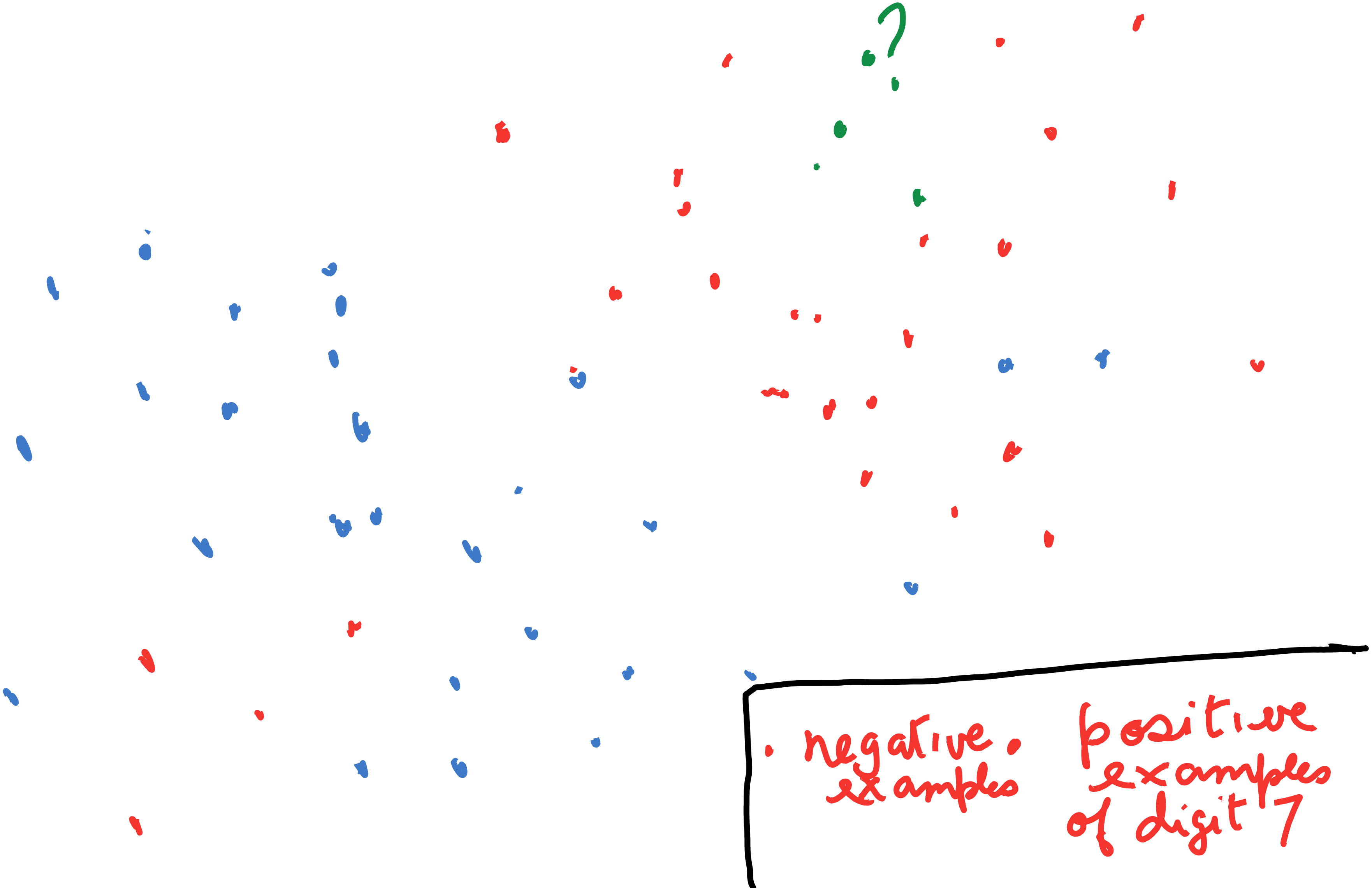
$\mathbb{R}^{16}$

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

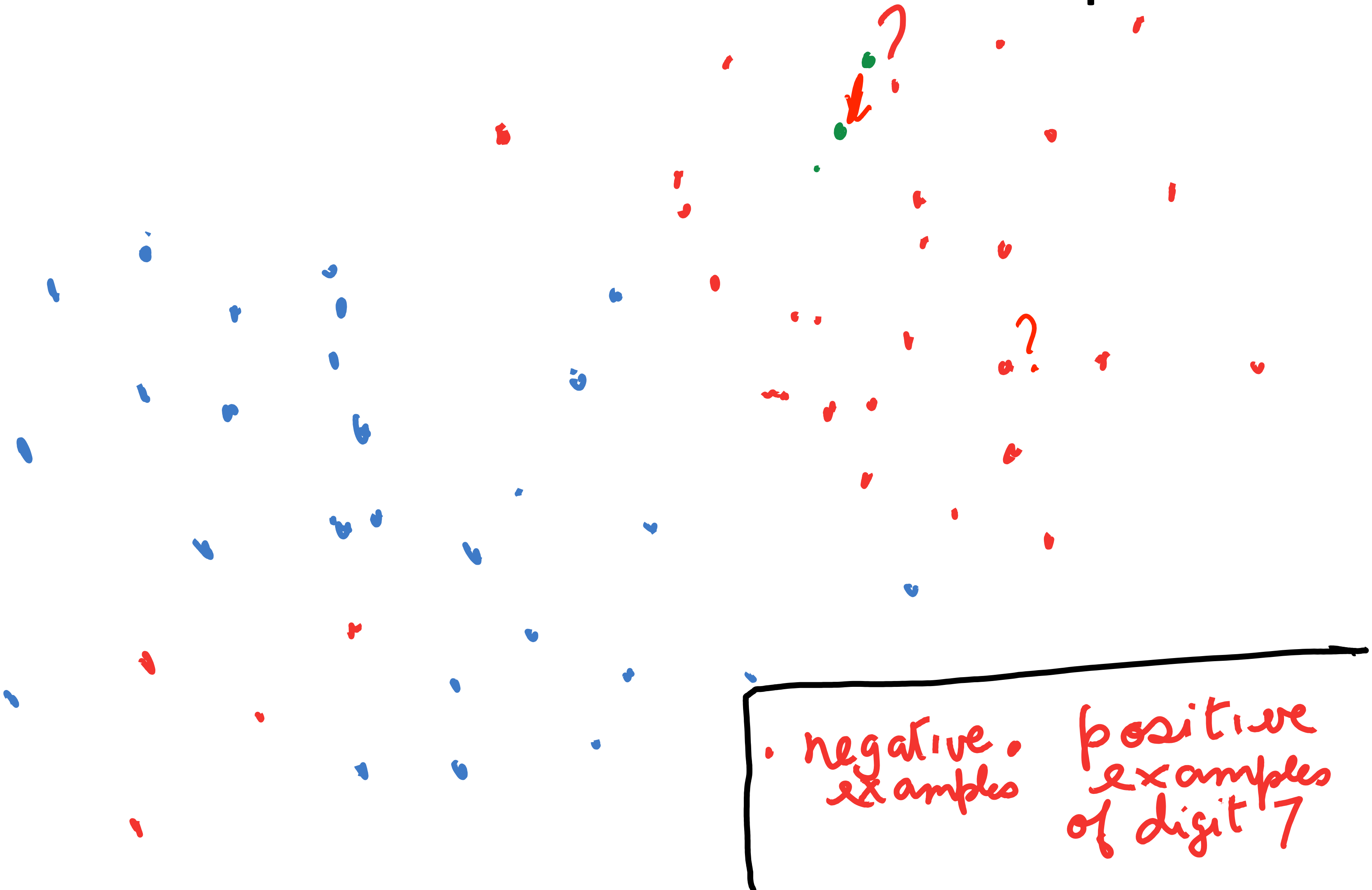
In feature space, positive and negative examples are just points…



How do we classify a new point?



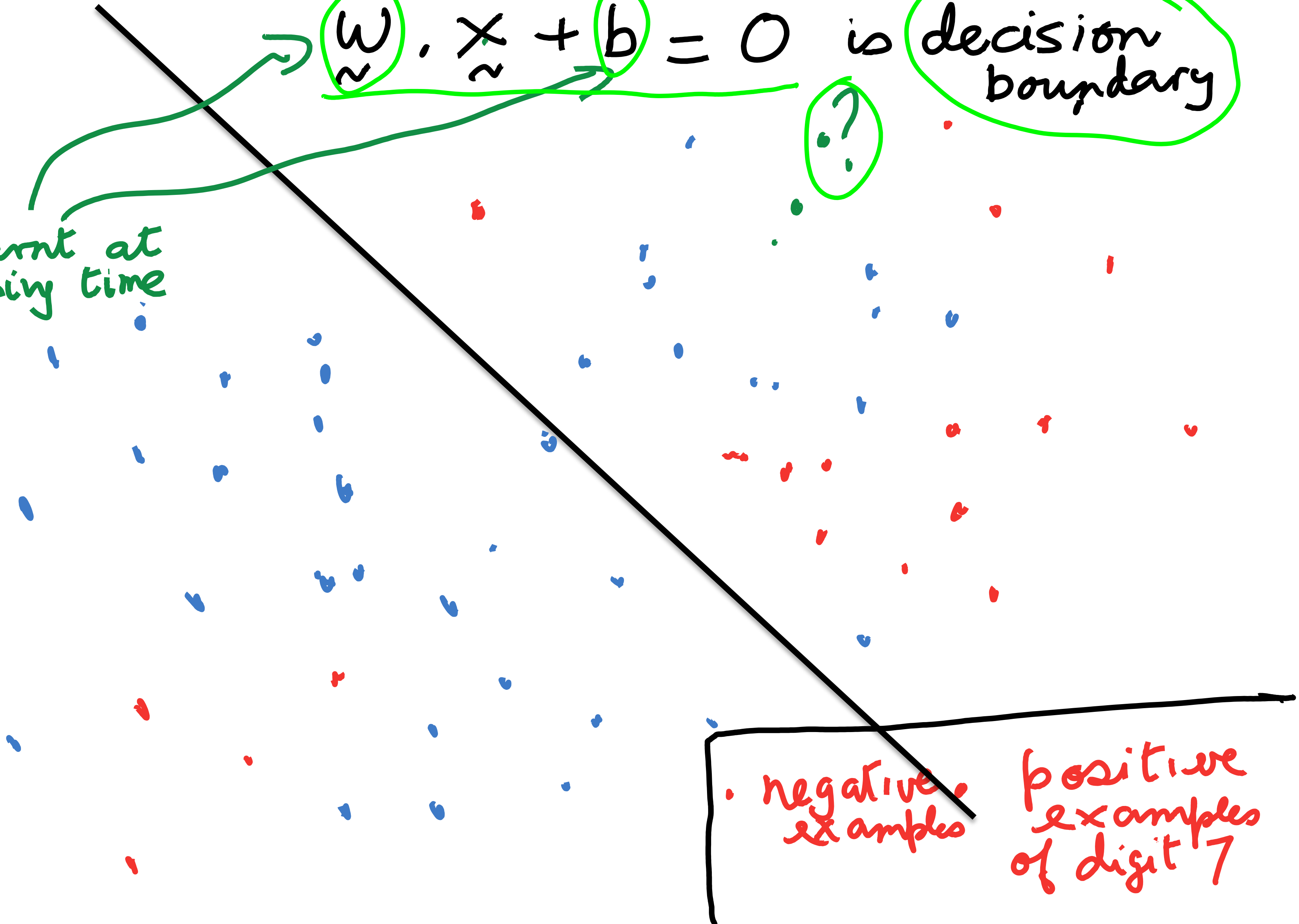
Nearest neighbor rule
“transfer label of nearest example”



Linear classifier rule

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

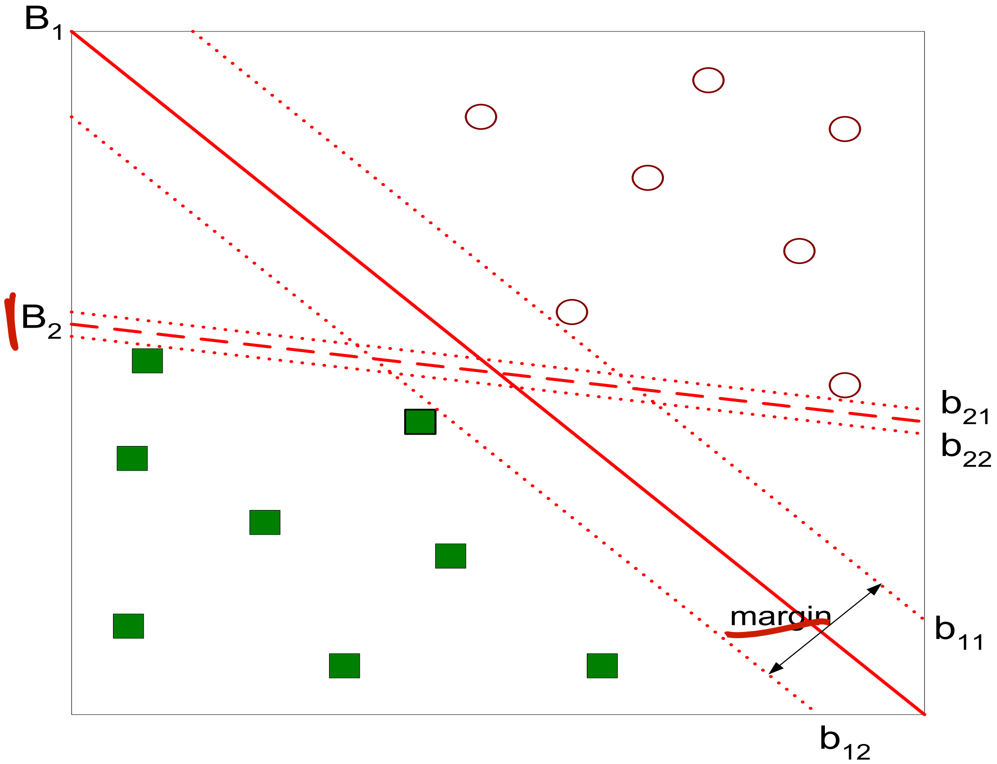
learnt at training time



Different approaches to training classifiers

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

Support Vector Machines



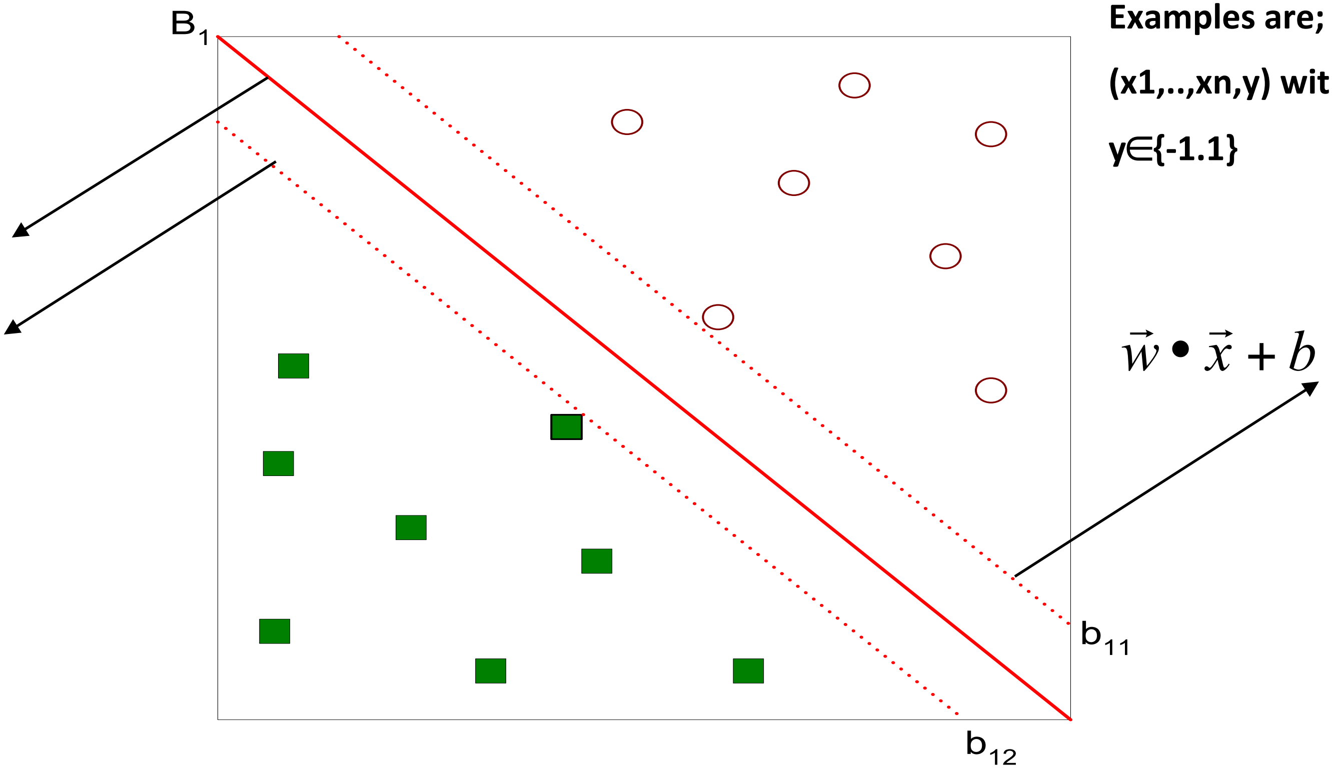
- Find hyperplane maximizes the margin => B1 is better than B2

Support Vector Machines

Examples are;

(x1,..,xn,y) with

y∈{-1.1}



$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

$$\vec{w} \bullet \vec{x} + b = +1$$

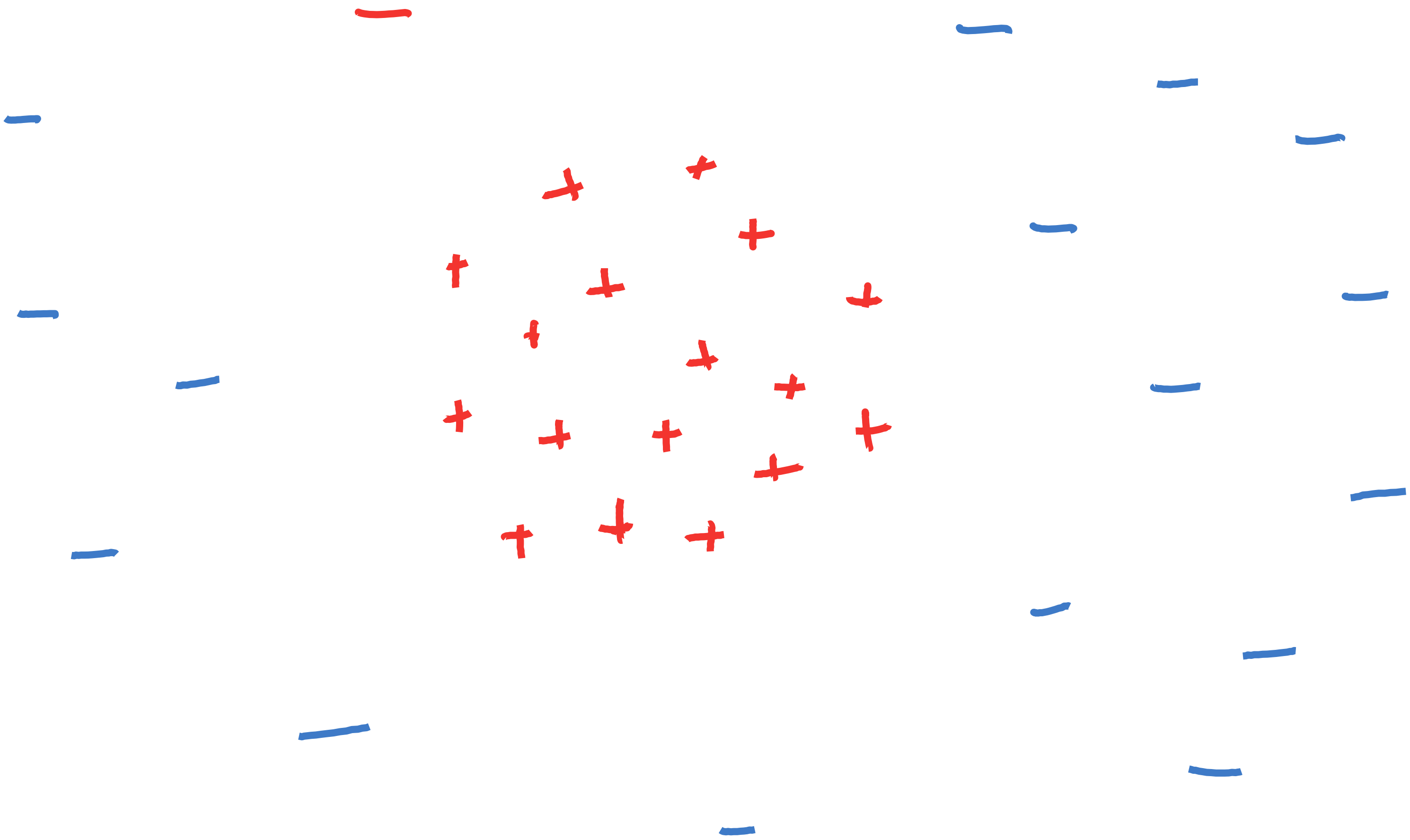
$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}} + \mathrm{b} \geq 1 \\ -1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}} + \mathrm{b} \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\| \vec{w} \|^2}$$

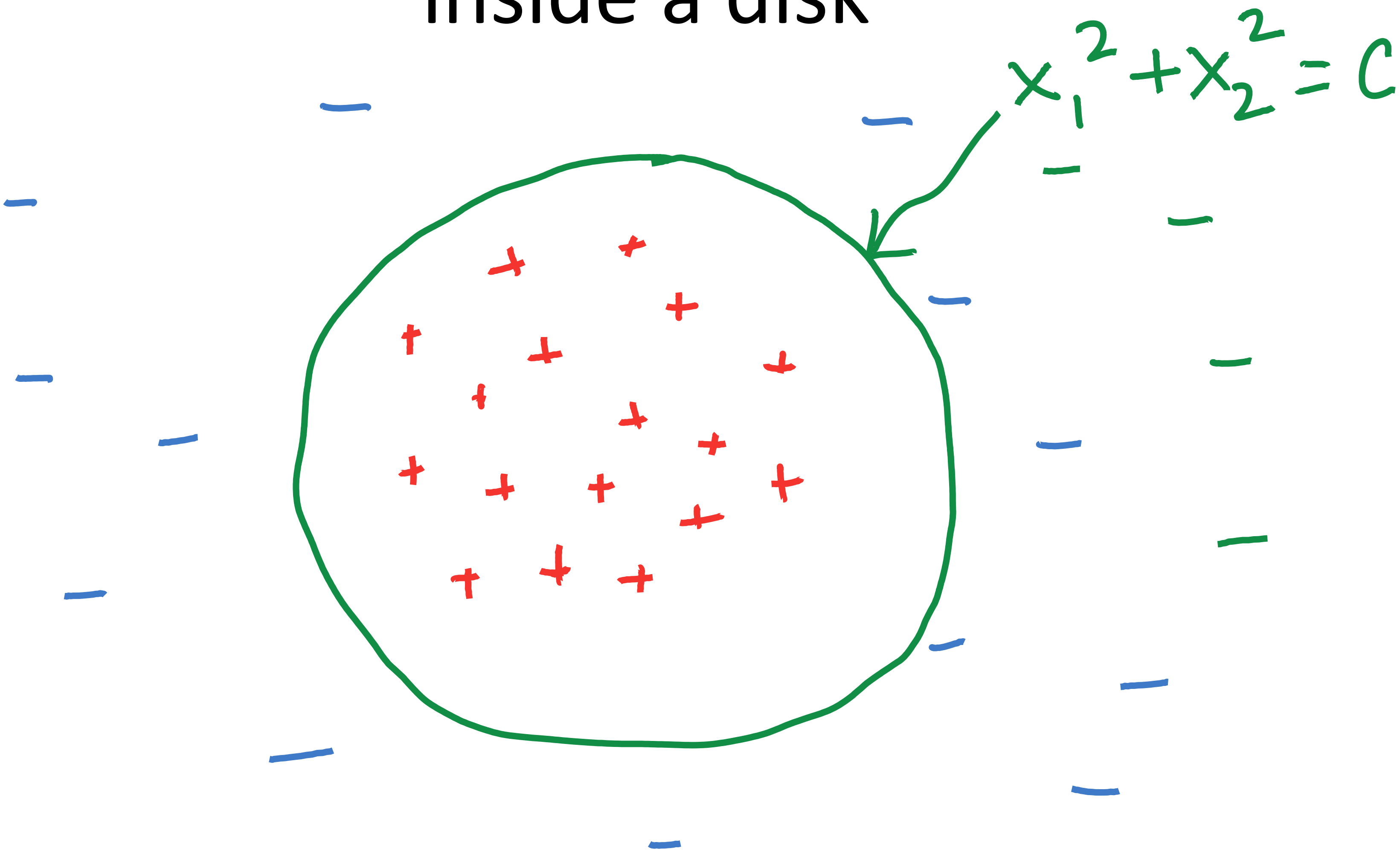
Some remarks..

- 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



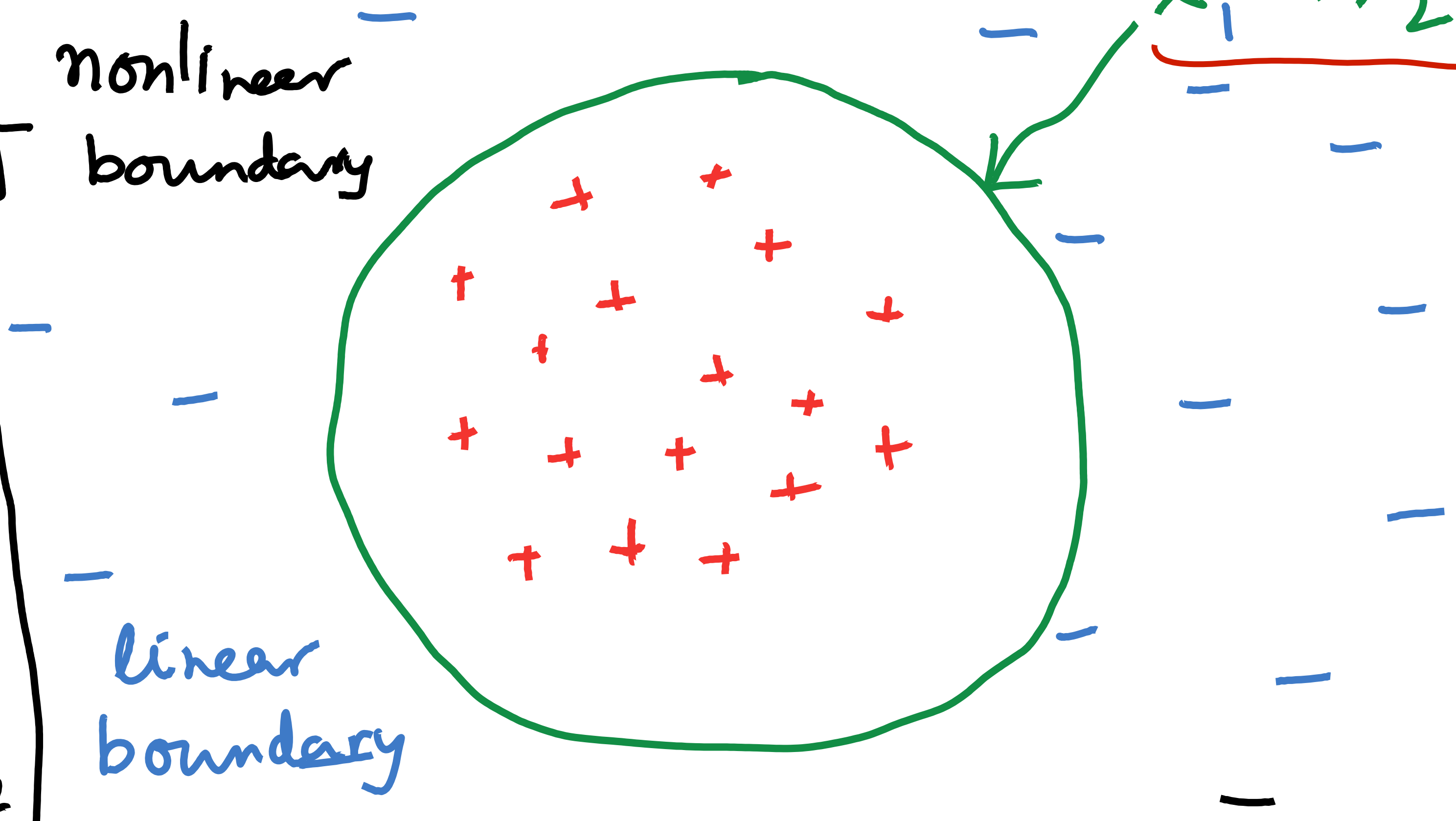
Suppose the positive examples lie inside a disk



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

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

nonlinear boundary



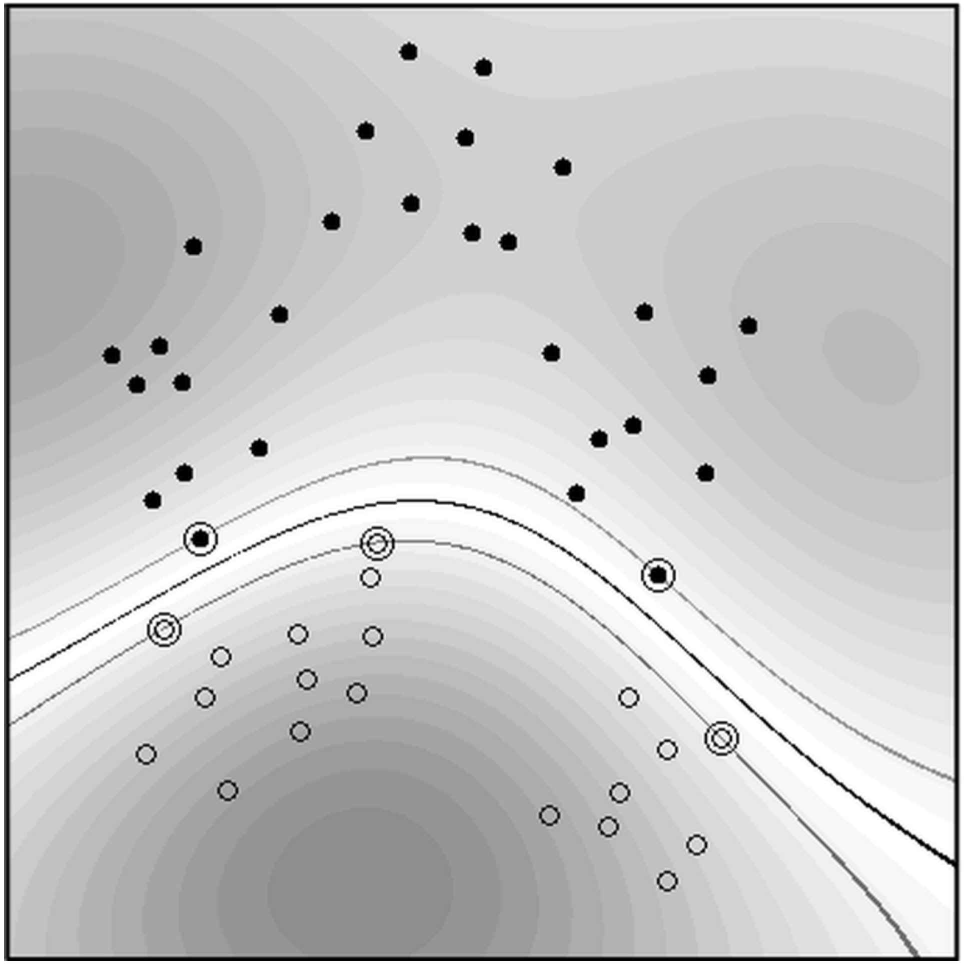
$$x_1^2 + x_2^2 = C$$

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

linear boundary

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

Kernel Support Vector Machines



Kernel :

- Inner Product in Hilbert Space

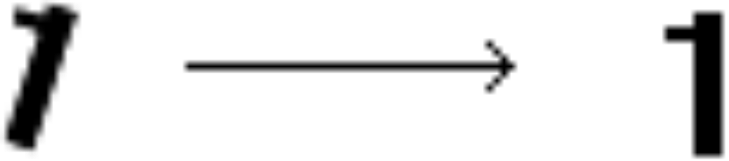
$$\underline{K(x,z)} = \Phi(x)^T \Phi(z)$$

- Can Learn Non Linear Boundaries

Skipping math here, Sorry !

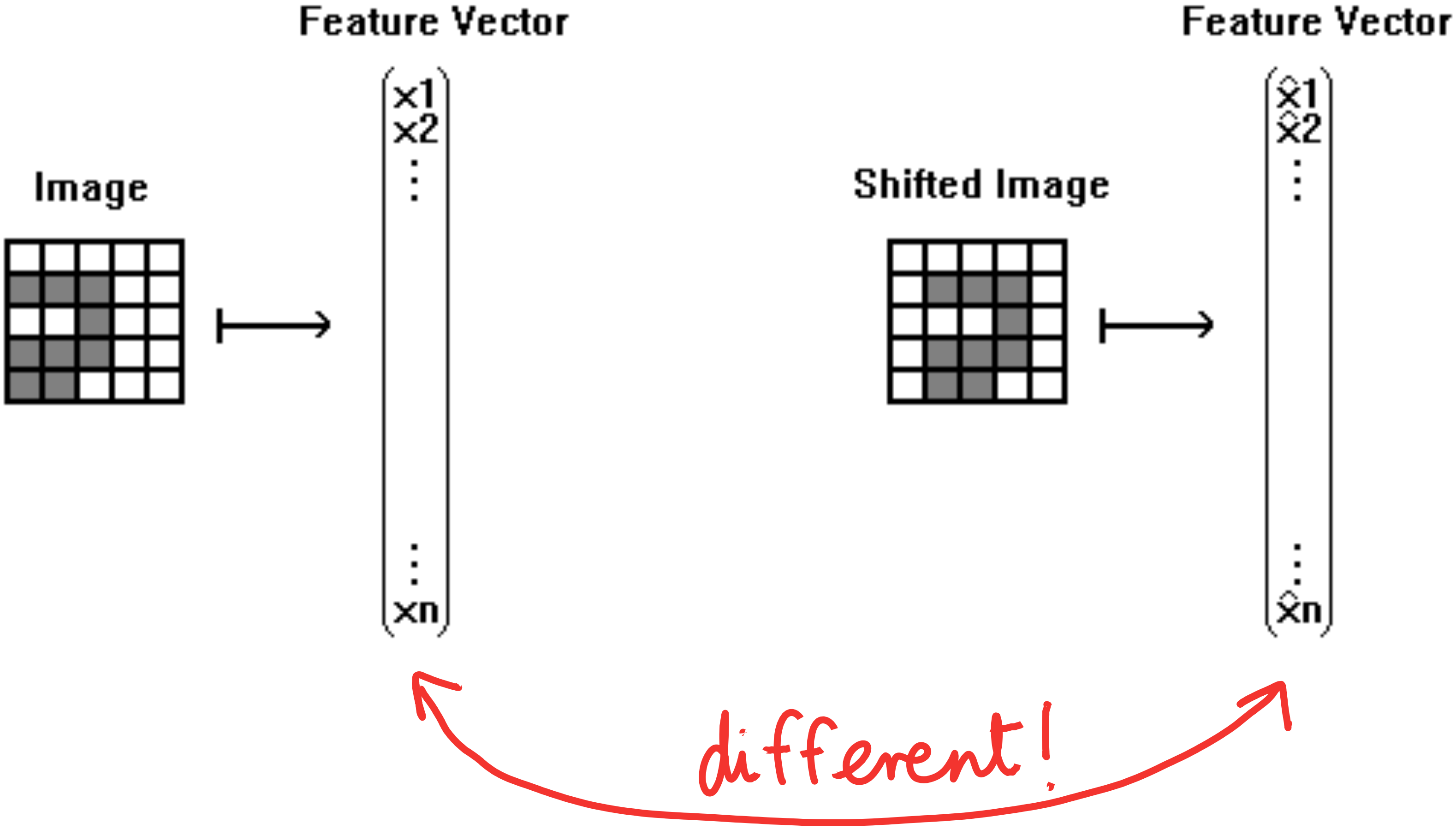
Transformation invariance
(or, why we love orientation histograms so much!)

- 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

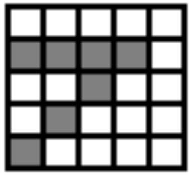


Image 1

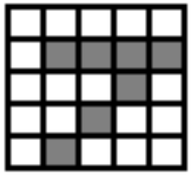
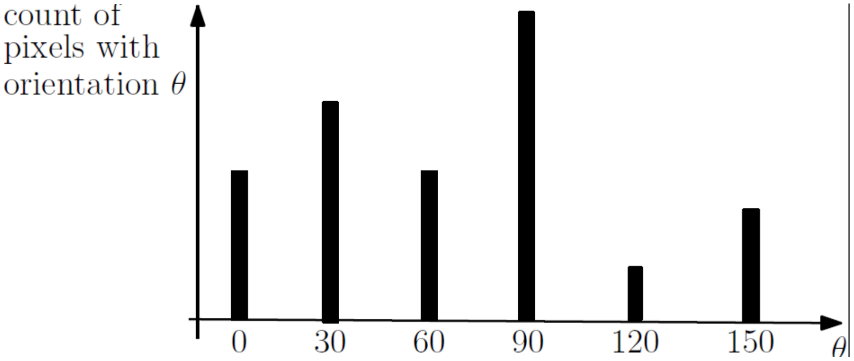


Image 2



- 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*

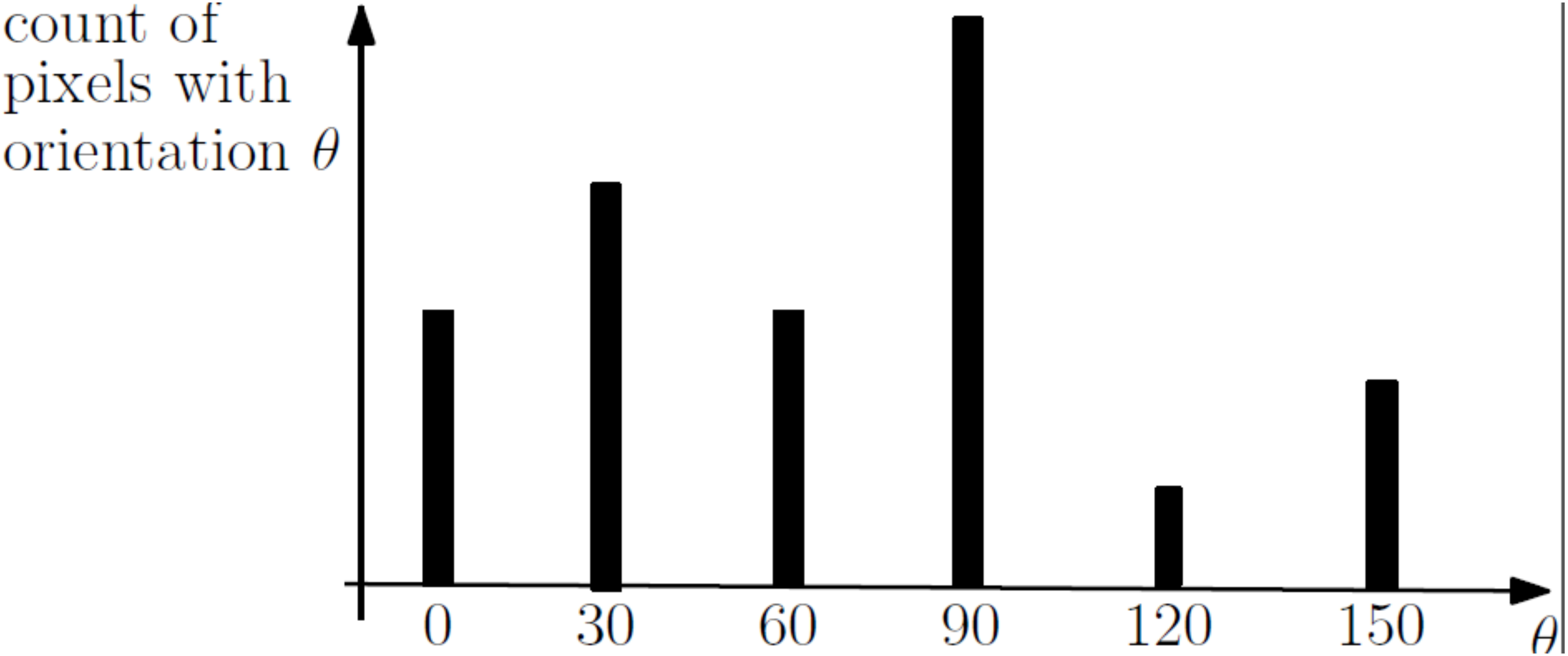
$$K(a,b) = \sum_{i=1}^{n} min(a_i, b_i)$$

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

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.

Orientation histograms



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

LINEAR $k_{lin}(\mathbf{x},\mathbf{y}) = \mathbf{x}\cdot\mathbf{y}$

HISTOGRAM $k_{int}(\mathbf{x},\mathbf{y}) = \min(\mathbf{x},\mathbf{y})$

GENERAL NON-LINEAR $\left\{\begin{aligned} k_{poly}(\mathbf{x},\mathbf{y}) &= (\mathbf{x}\cdot\mathbf{y}+1)^5 \\ k_{rbf}(\mathbf{x},\mathbf{y}) &= \exp(-\gamma||\mathbf{x}-\mathbf{y}||^2) \end{aligned}\right.$

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 /

Arbitrary Kernel

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

Histogram Intersection Kernel

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

Cost: # Support Vectors x Cost of kernel computation

For a linear kernel, h(x) simplifies

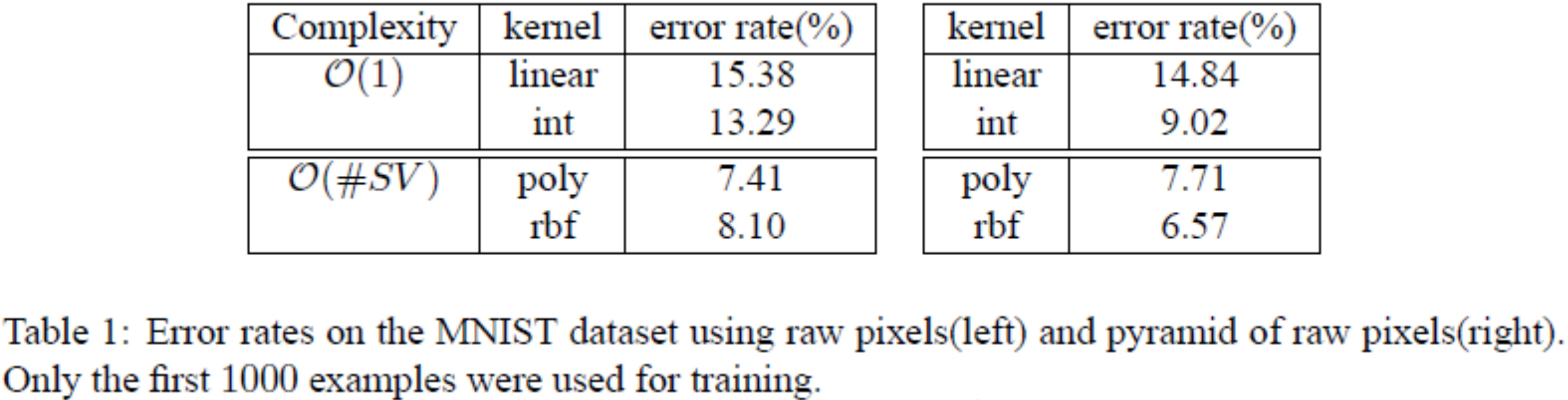
$$\sum \alpha^j x . x_j + b = \underset{\uparrow}{x} . \underbrace{\left(\sum \alpha^j x_j\right)}_{\text{PRECOMPUTED}} + b$$

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.



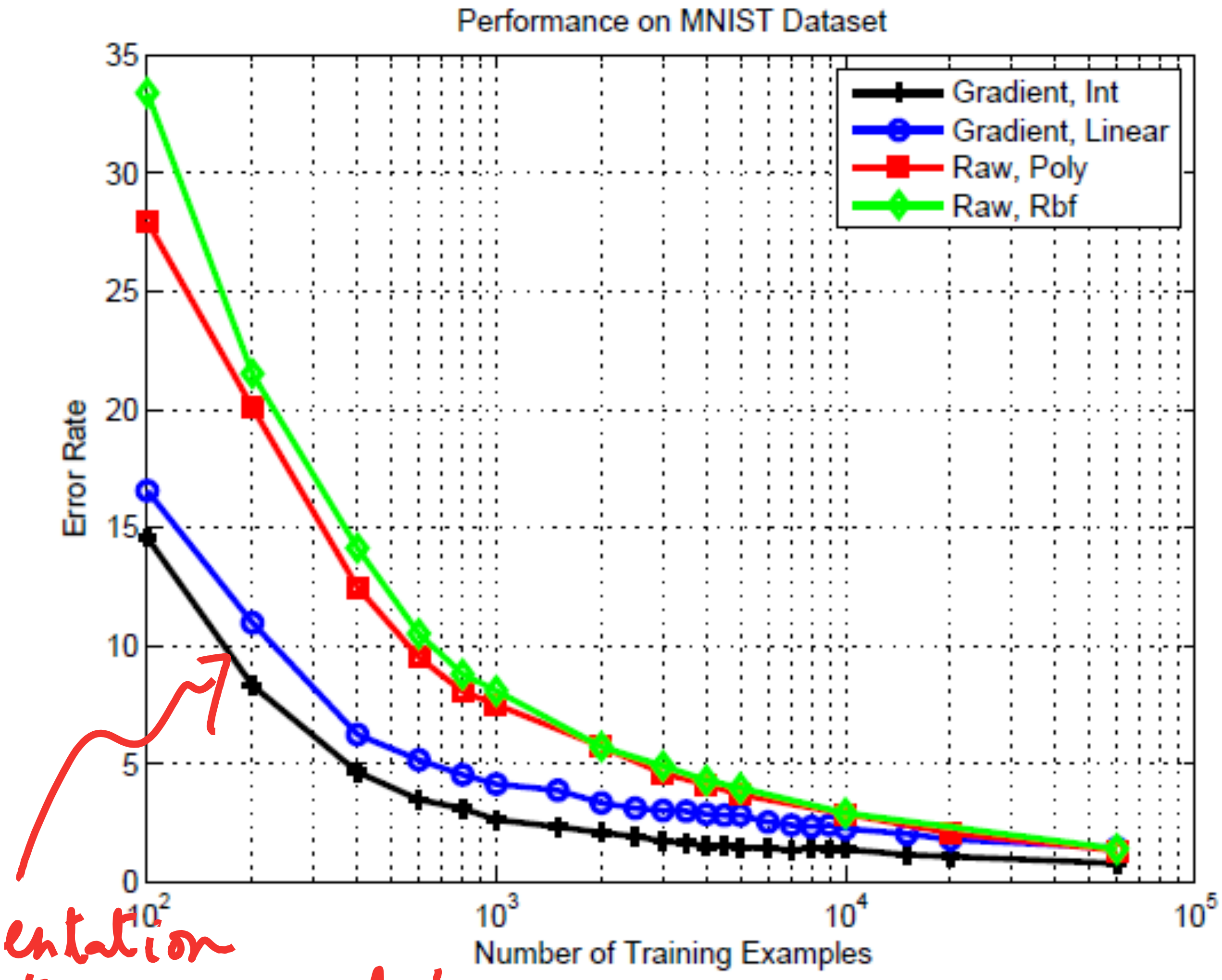
Complexity	kernel	error rate(%)
$\mathcal{O}(1)$	linear	15.38
	int	13.29
$\mathcal{O}(\#SV)$	poly	7.41
	rbf	8.10

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.

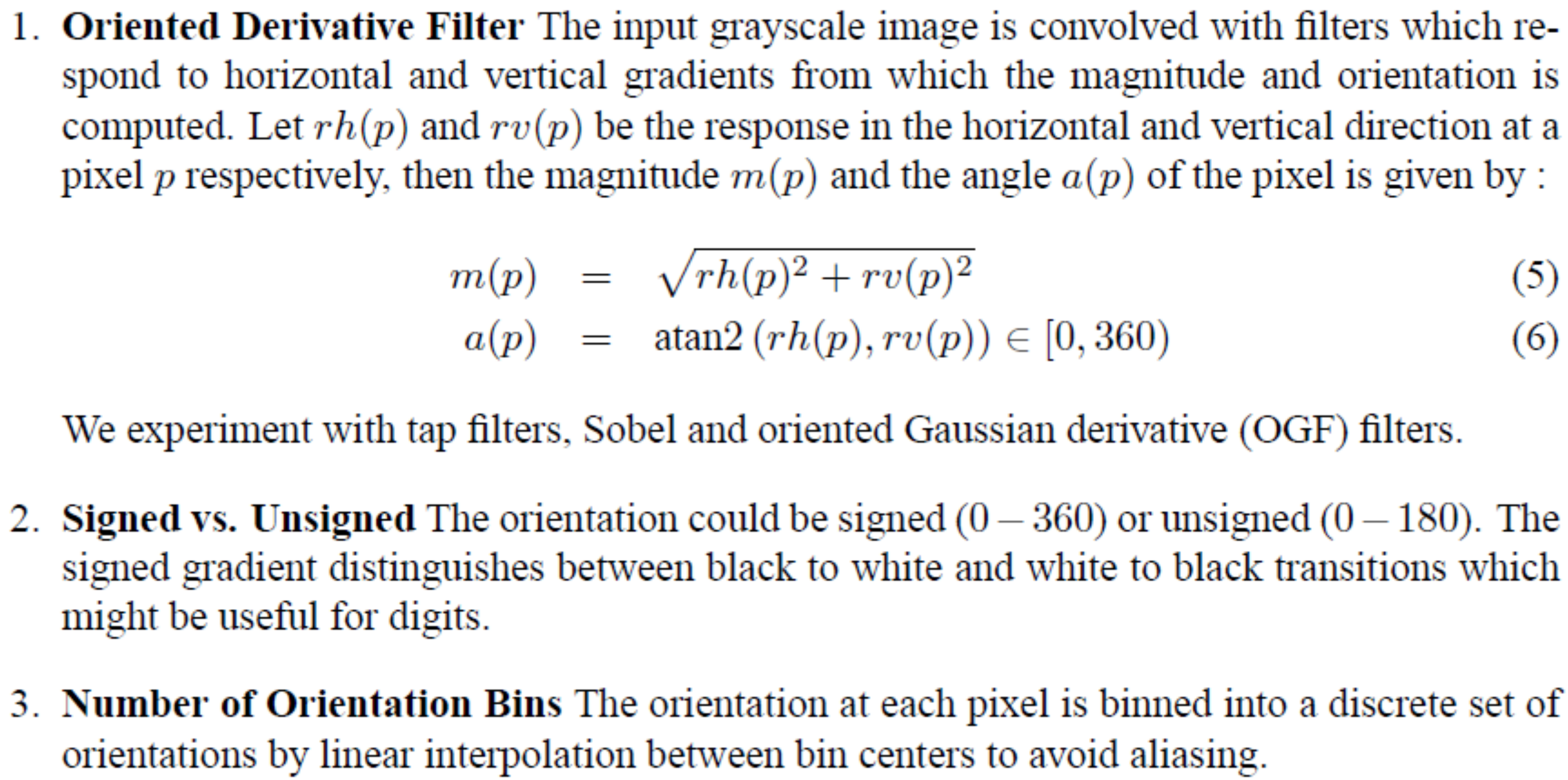
Nonlinear kernels give smaller error rates, but at considerably higher computational expense

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 :

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

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

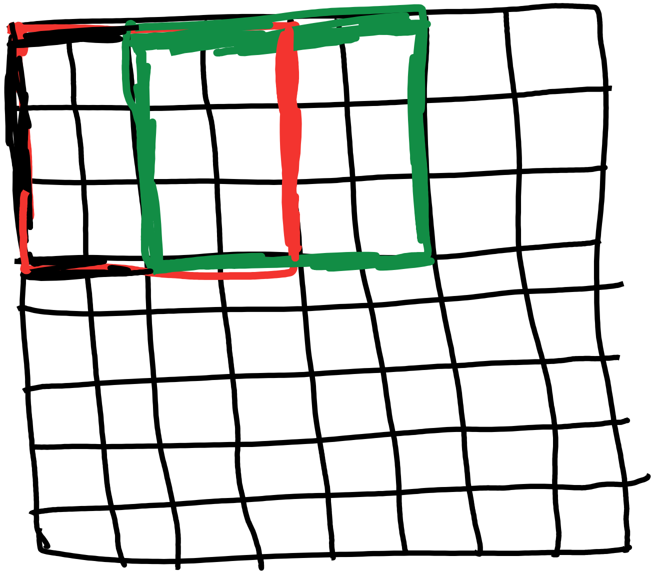
We experiment with tap filters, Sobel and oriented Gaussian derivative (OGF) filters.

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

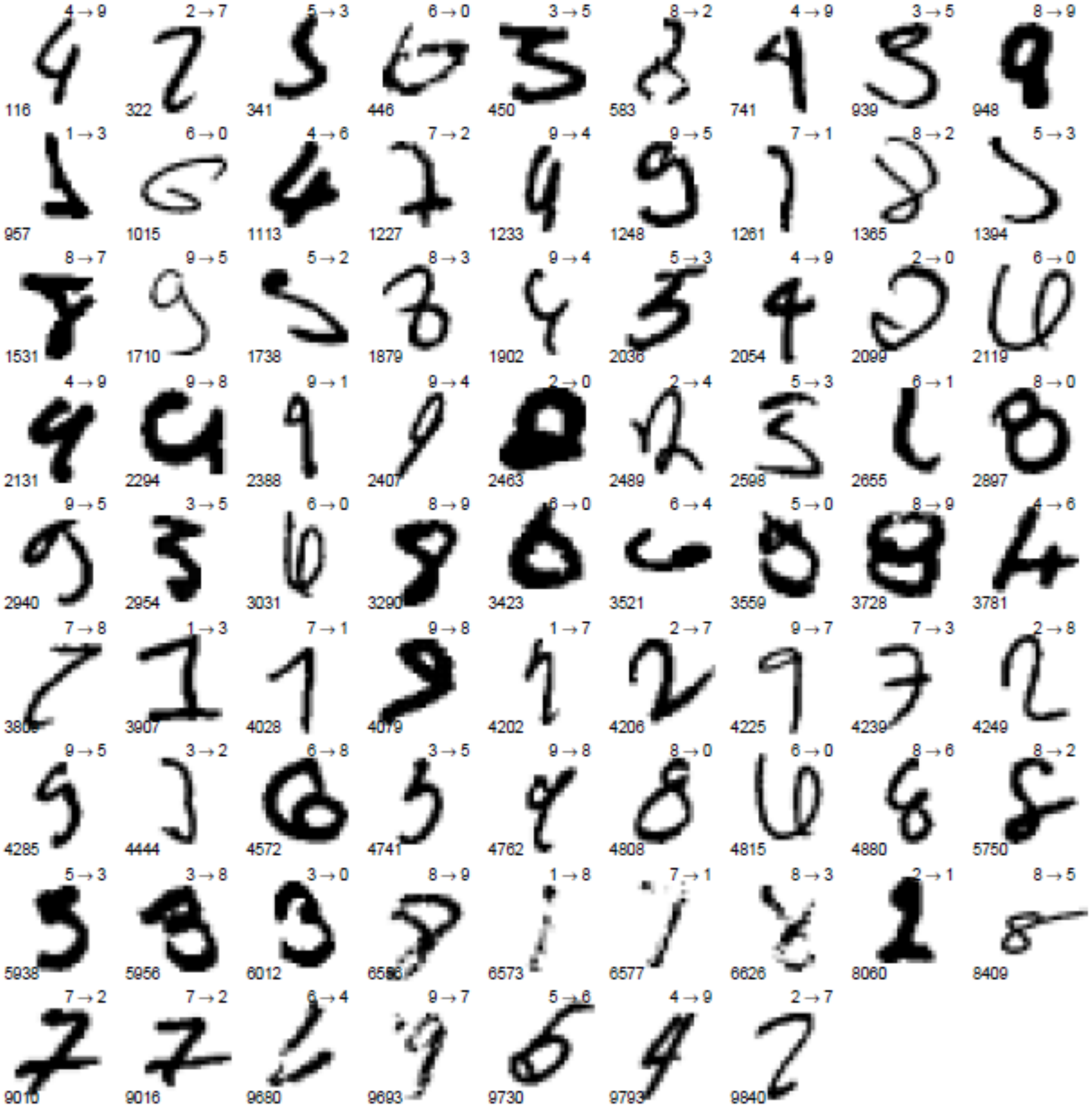


- Compute histogram on a c × c block
- Shift block by c/2 and recompute histogram
- All the block histograms are stacked in a long feature vector

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!

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

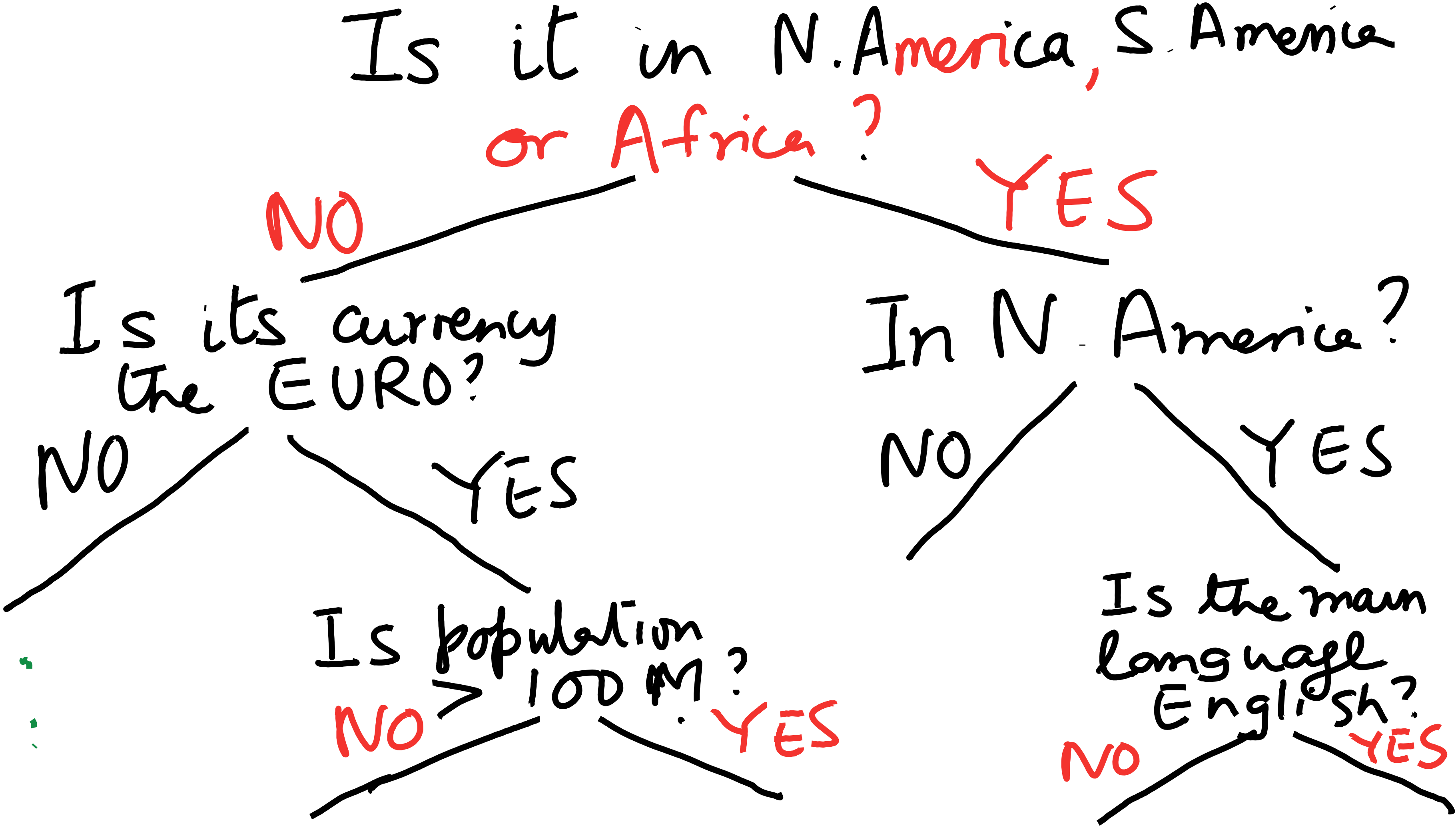
Jitendra Malik

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?


Is it in N. America, S. America or Africa?
NO
YES
Is its currency the EURO?
NO
YES
In N. America?
NO
YES
Is population > 100 M?
NO
YES
Is the main language English?
NO
YES

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

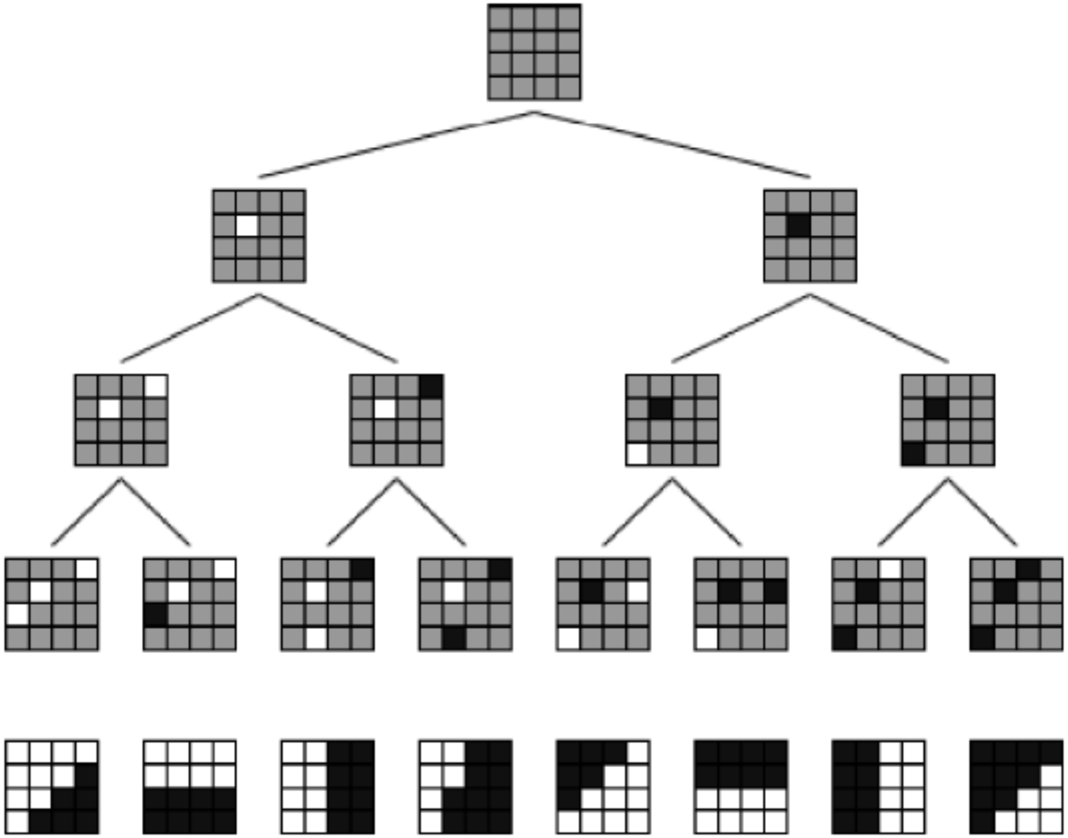


Fig. 1. First three tag levels with most common configurations.

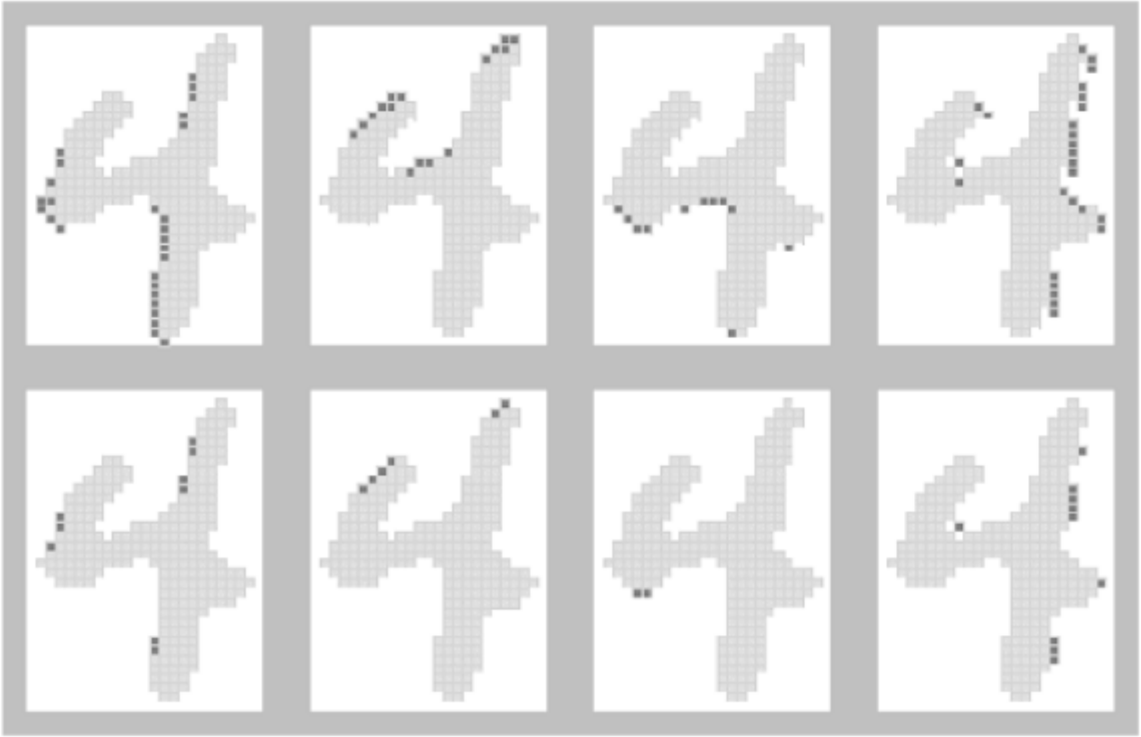


Fig. 2. Top: All instances of four depth three tags. Bottom: All instances of four depth five tags.

An example of such an arrangement

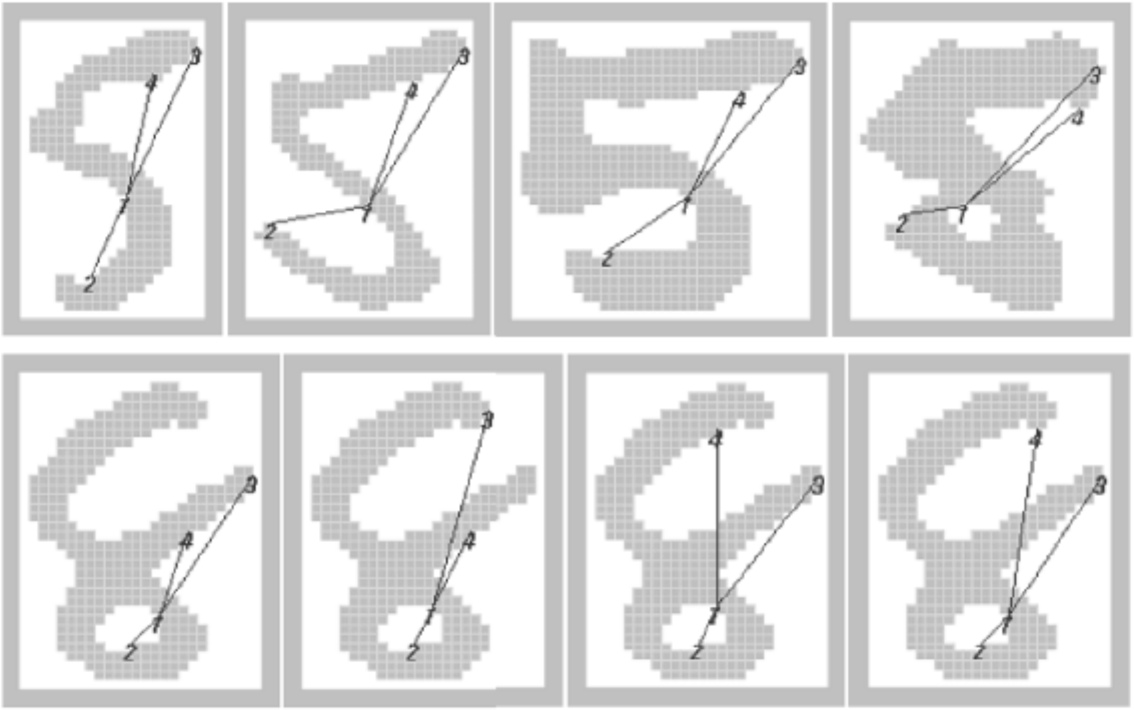


Fig. 3. Top row: Instances of a geometric arrangement in several 5s. Bottom row: Several instances of the geometric arrangement in one 8.

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

Additional questions “grow” the arrangement

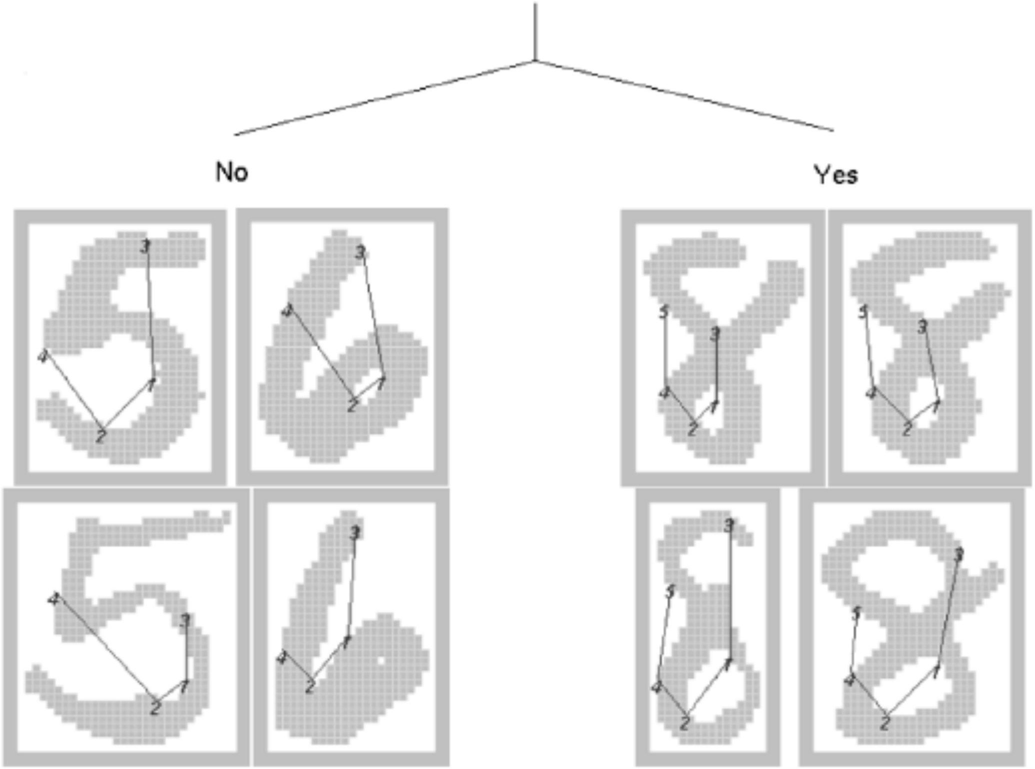


Fig. 4. Example of node-splitting in a typical digit tree; the query involves adding a fifth tag (vertex) to the pending arrangement. Specifically, the proposed arrangement adds a fifth vertex and a fourth relation to the existing graph which has four vertices and three relations.

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”

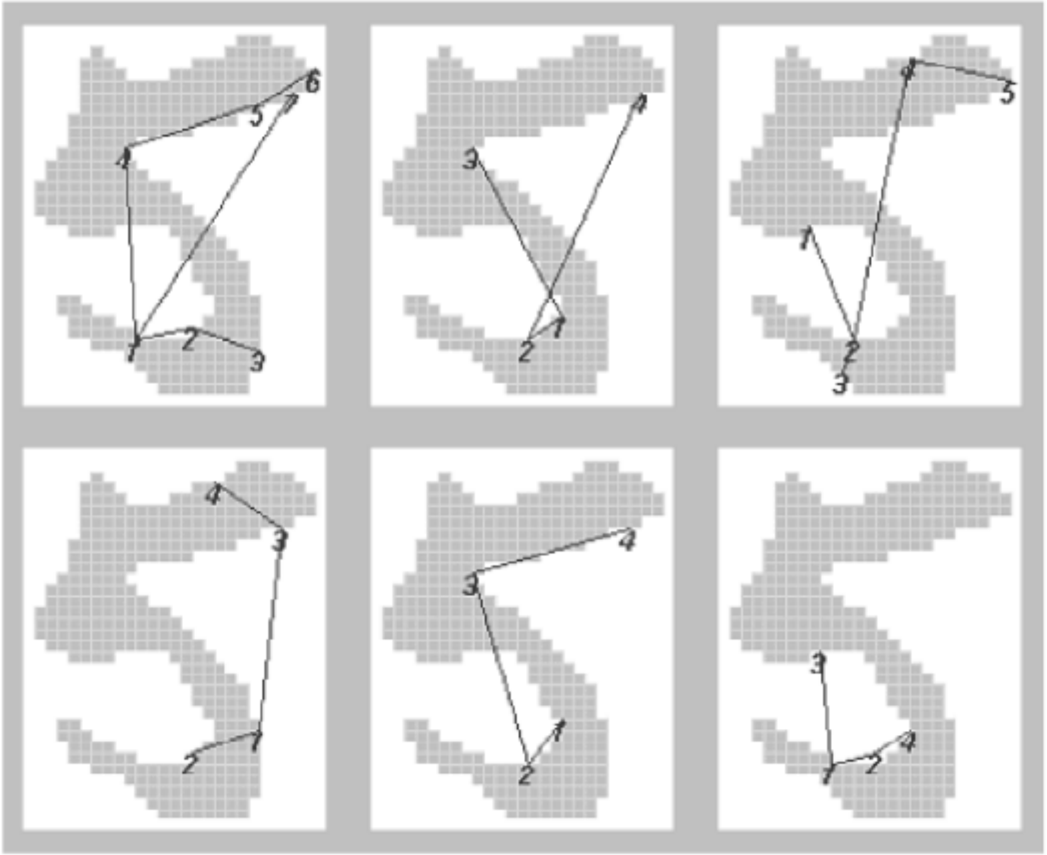
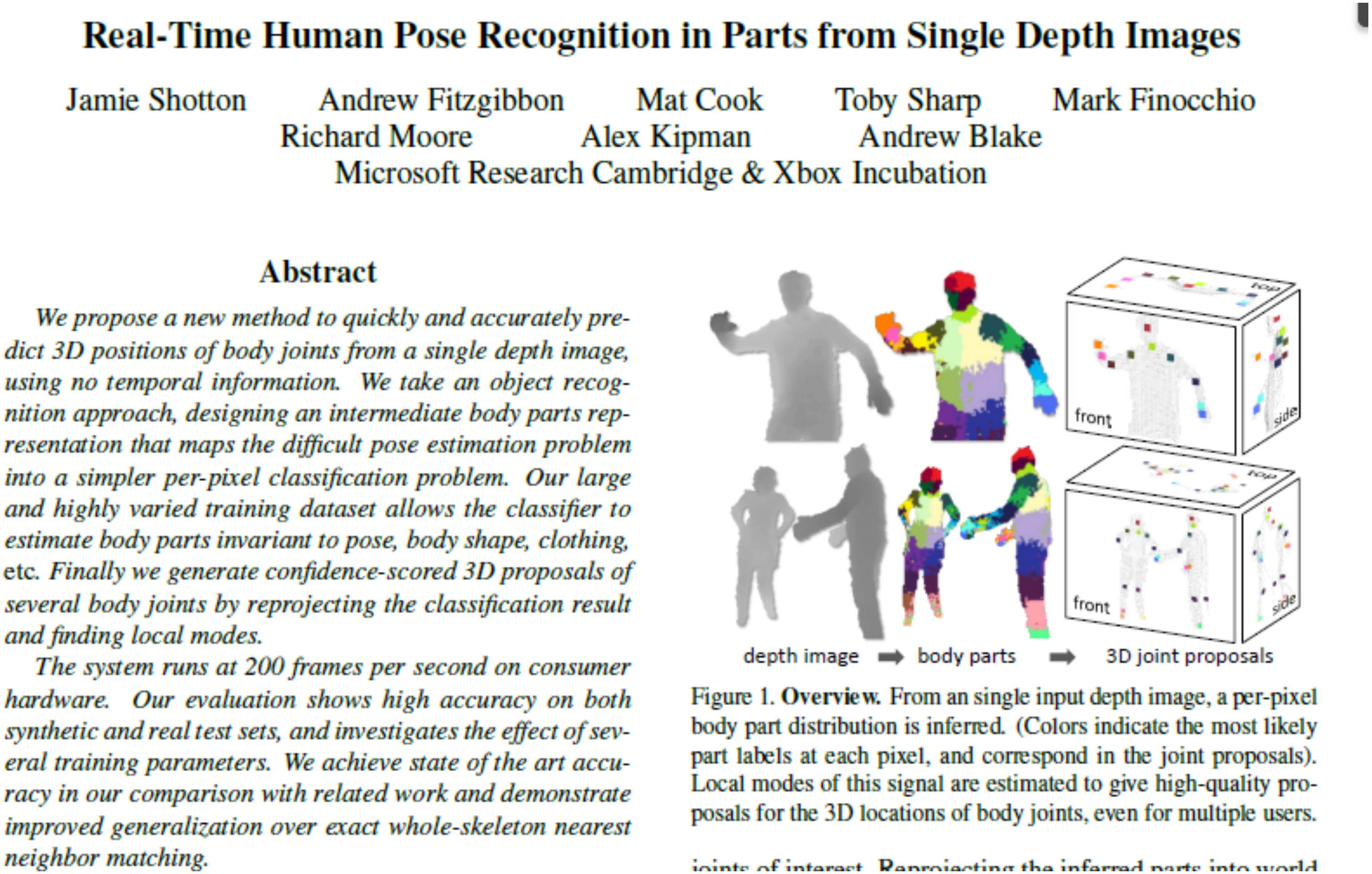


Fig. 5. Arrangements found in an image at terminal nodes of six different trees.

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

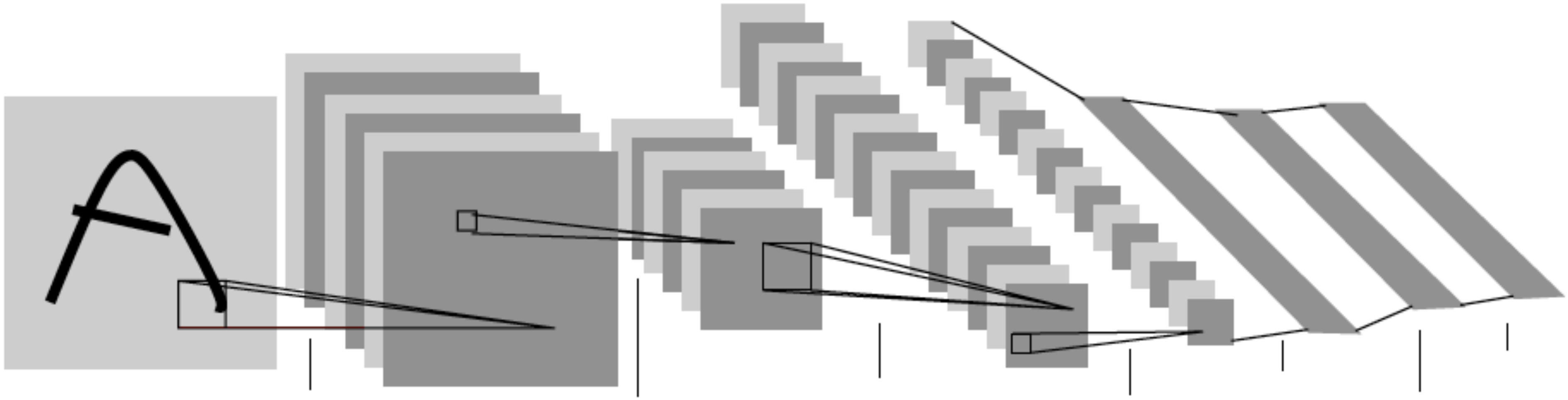
We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel 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.

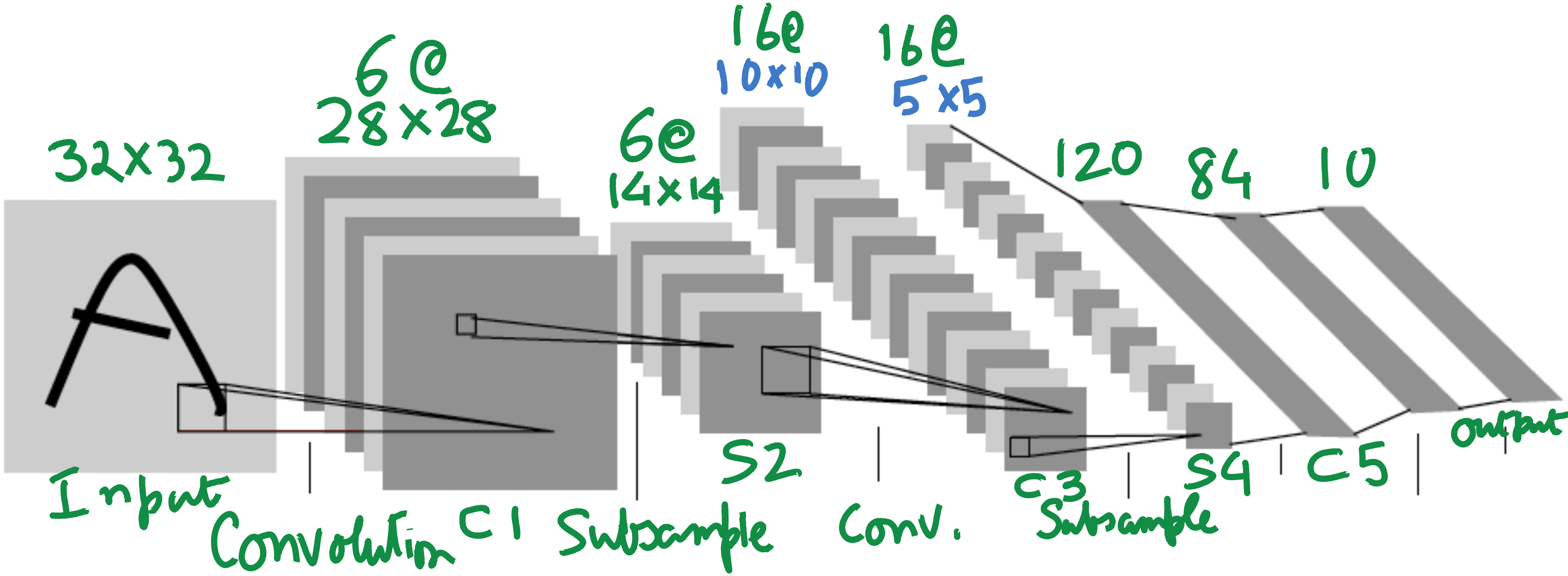
Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of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

joints of interest. Reprojecting the inferred parts into world

Convolutional Neural Networks
LeCun et al (1989)



Convolutional Neural Networks (LeCun et al)



This section describes in more detail the architecture of LeNet-5, the Convolutional Neural Network used in the experiments. LeNet-5 comprises 7 layers, not counting the input, all of which contain trainable parameters (weights). The input is a 32x32 pixel image. This is significantly larger than the largest character in the database (at most 20x20 pixels centered in a 28x28 field). The reason is that it is desirable that potential distinctive features such as stroke end-points or corner can appear *in the center* of the receptive field of the highest-level feature detectors. In LeNet-5 the set of centers of the receptive fields of the last convolutional layer (C3, see below) form a 20x20 area in the center of the 32x32 input. The values of the input pixels are normalized so that the background level (white) corresponds to a value of -0.1 and the foreground (black) corresponds to 1.175. This makes the mean input roughly 0, and the variance roughly 1 which accelerates learning [46].

In the following, convolutional layers are labeled Cx, subsampling layers are labeled Sx, and fully-connected layers are labeled Fx, where x is the layer index.

Layer C1 is a convolutional layer with 6 feature maps. Each unit in each feature map is connected to a 5x5 neighborhood in the input. The size of the feature maps is 28x28 which prevents connection from the input from falling off the boundary. C1 contains 156 trainable parameters, and 122,304 connections.

Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. The 2x2 receptive fields are non-overlapping, therefore feature maps in S2 have half the number of rows and column as feature maps in C1. Layer S2 has 12 trainable parameters and 5,880 connections.

Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map is connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Table I shows the set of S2 feature maps

Layer S2 is a sub-sampling layer with 6 feature maps of size 14x14. Each unit in each feature map is connected to a 2x2 neighborhood in the corresponding feature map in C1. The four inputs to a unit in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias. The result is passed through a sigmoidal function. The 2x2 receptive fields are non-overlapping, therefore feature maps in S2 have half the number of rows and column as feature maps in C1. Layer S2 has 12 trainable parameters and 5,880 connections.

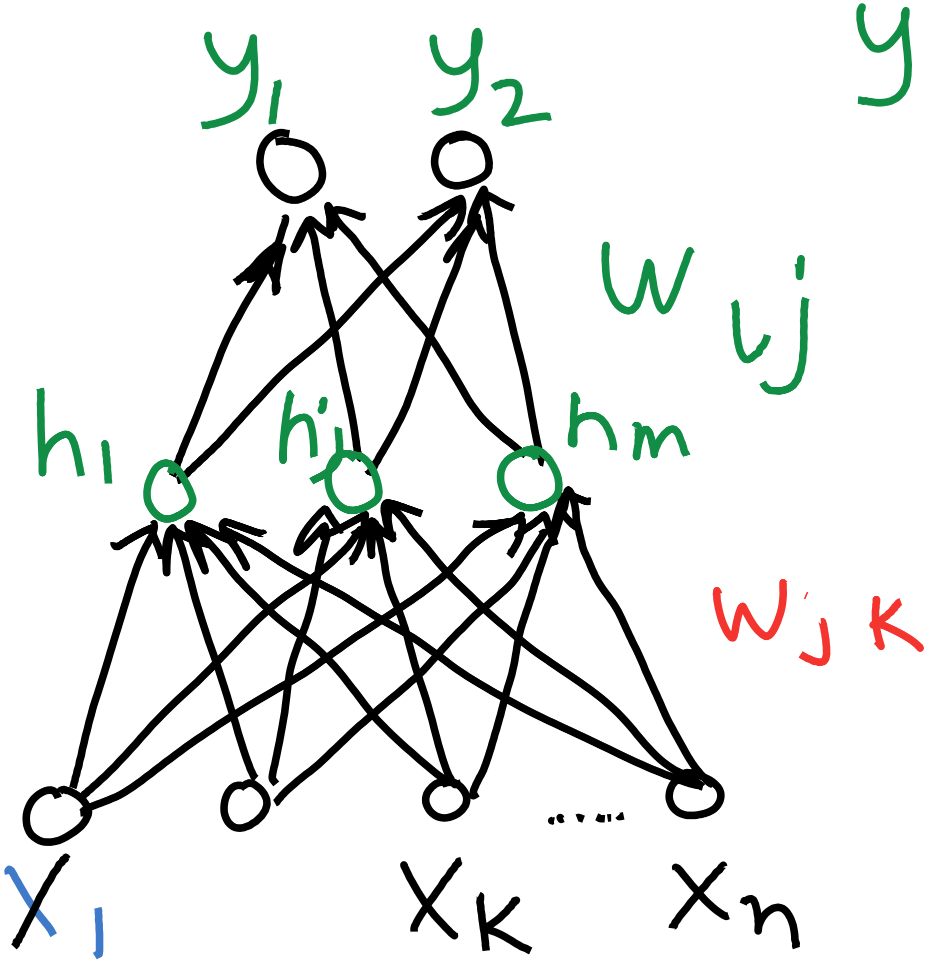
Layer C3 is a convolutional layer with 16 feature maps. Each unit in each feature map is connected to several 5x5 neighborhoods at identical locations in a subset of S2's feature maps. Table I shows the set of S2 feature maps combined by each C3 feature map. Why not connect every S2 feature map to every C3 feature map? The reason is twofold. First, a non-complete connection scheme keeps the number of connections within reasonable bounds. More importantly, it forces a break of symmetry in the network. Different feature maps are forced to extract different (hopefully complementary) features because they get different sets of inputs. The rationale behind the connection scheme in table I is the following. The first six C3 feature maps take inputs from every contiguous subsets of three feature maps in S2. The next six take input from every contiguous subset of four. The next three take input from some discontinuous subsets of four. Finally the last one takes input from all S2 feature maps. Layer C3 has 1,516 trainable parameters and 151,600 connections.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED

Training multi-layer networks



$$y_i = g\left(\sum_j w_{ij} h_j\right)$$

$$h_j = g\left(\sum_k w_{jk} x_k\right)$$

Minimize $\left(y_i - y_i^{desired}\right)^2$ by suitable choice of w's