Karlsruhe Institute of Technology

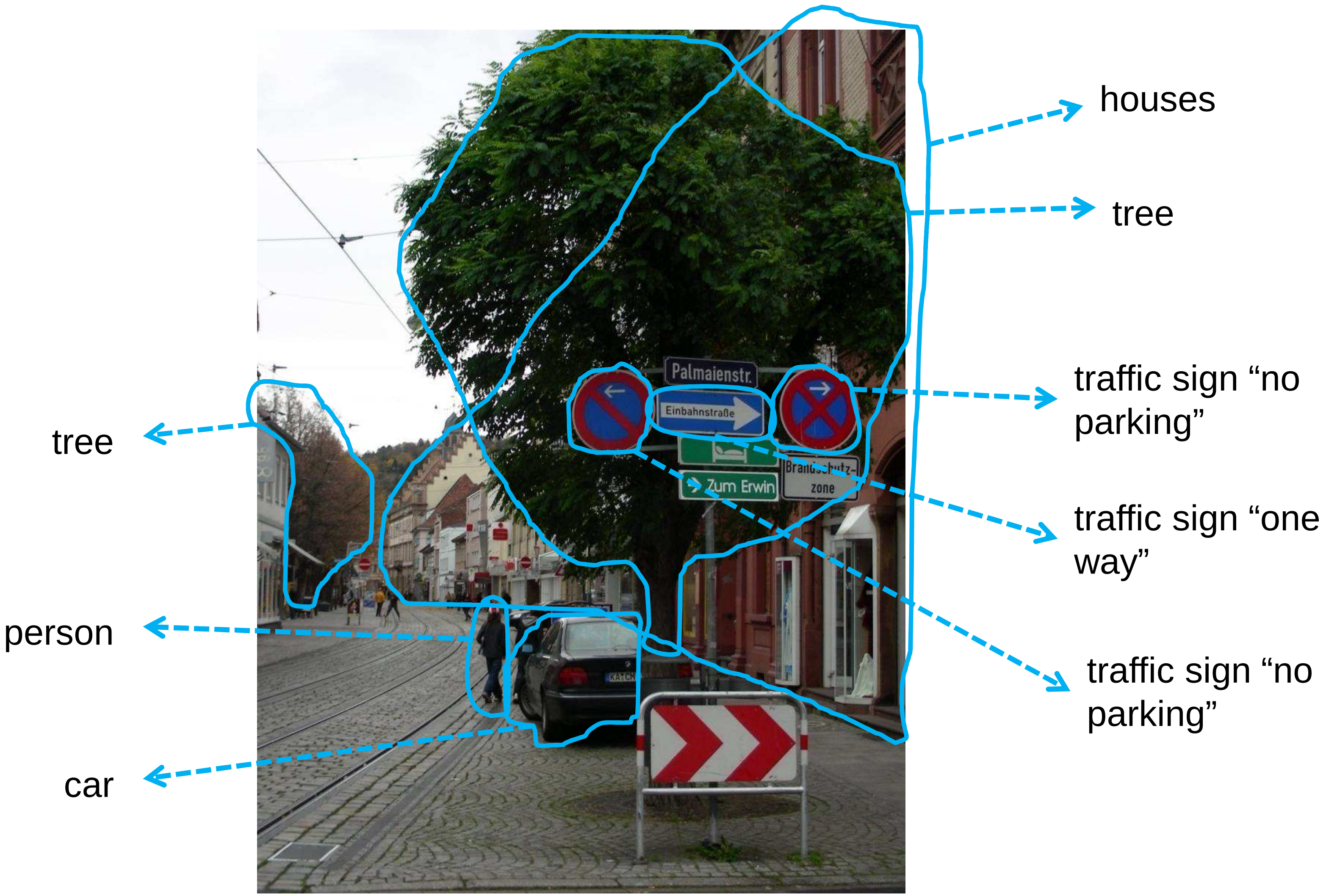
Machine Vision

Chapter 10: Pattern Recognition

Dr. Martin Lauer

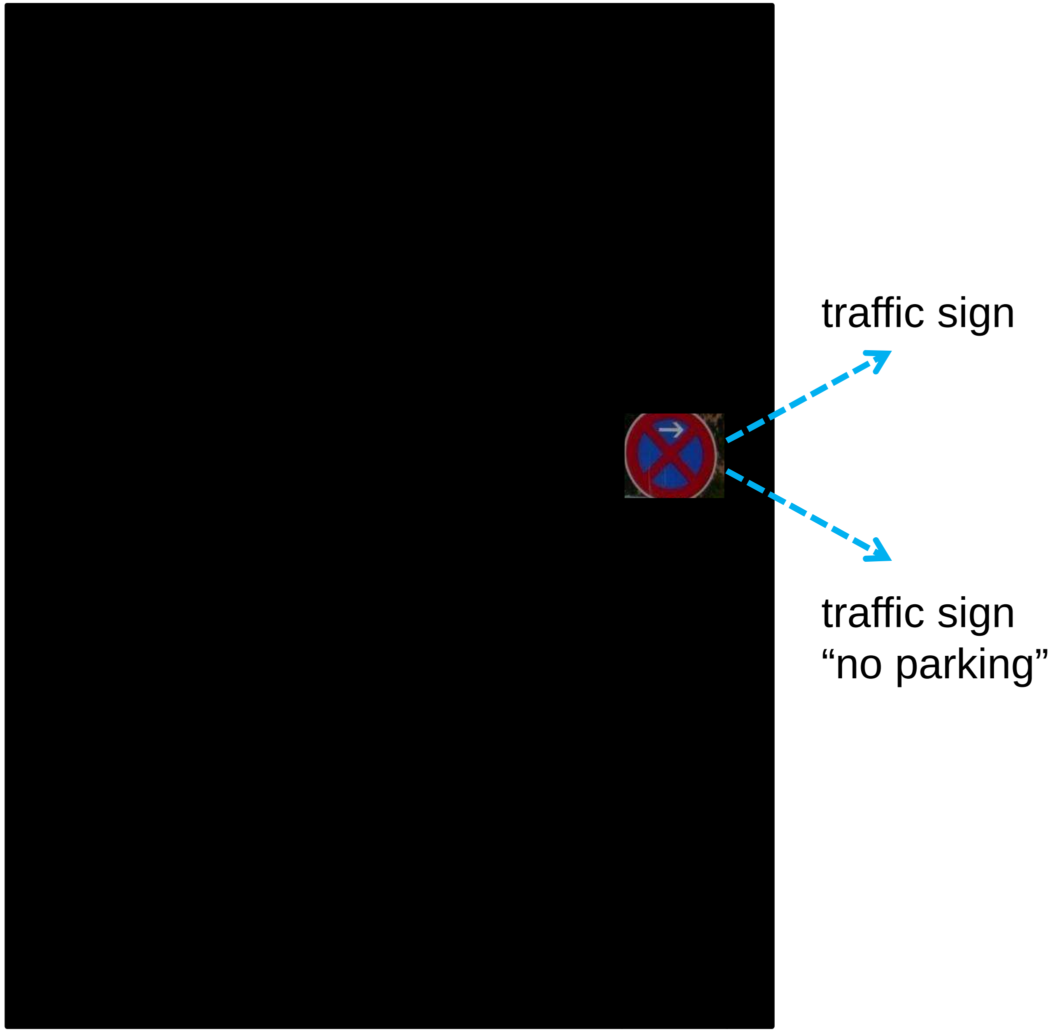
Institut für Mess- und Regelungstechnik

Classification



Classification: assigning objects to categories (classes)

Classification

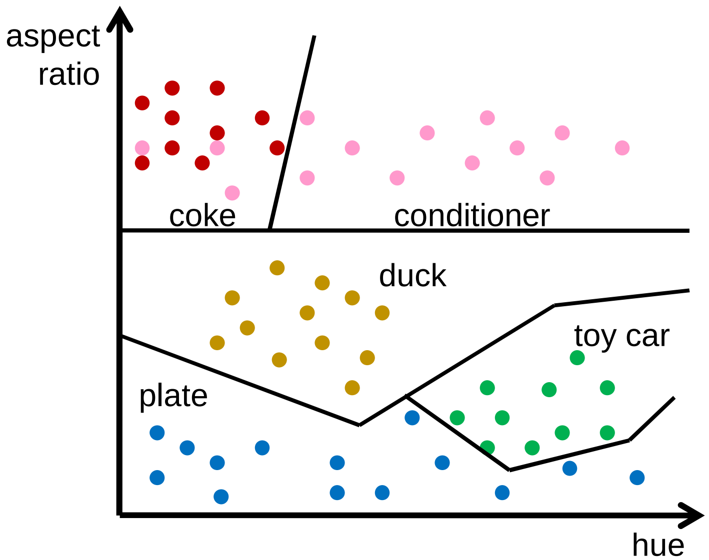


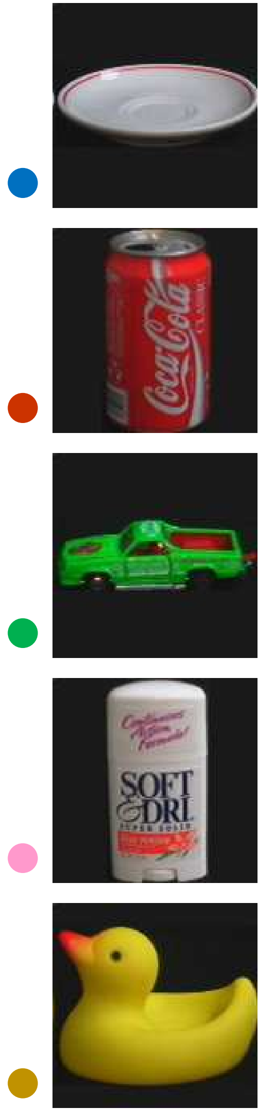
in machine vision:

- extract the relevant object from the background (segmentation)
- assign object to a category (classification)
- both steps might depend on each other

Classification cont.

- how can we distinguish these objects?
  – geometric features like aspect ratio, roundness, …
  – color features like dominant hue, average saturation, variance of color, …
  – …
- from a sample of images we get:

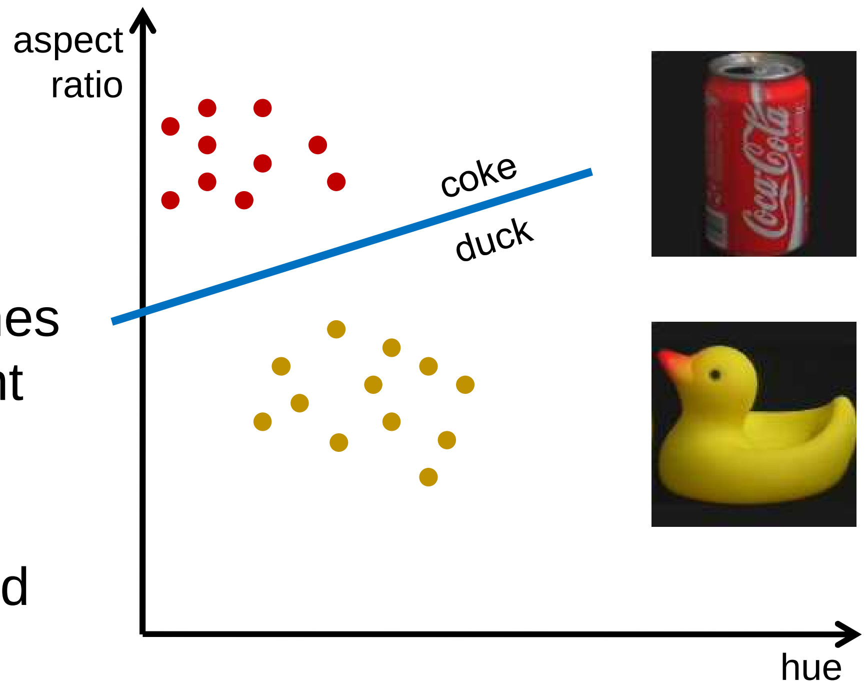




images source: COIL-100 database, http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php

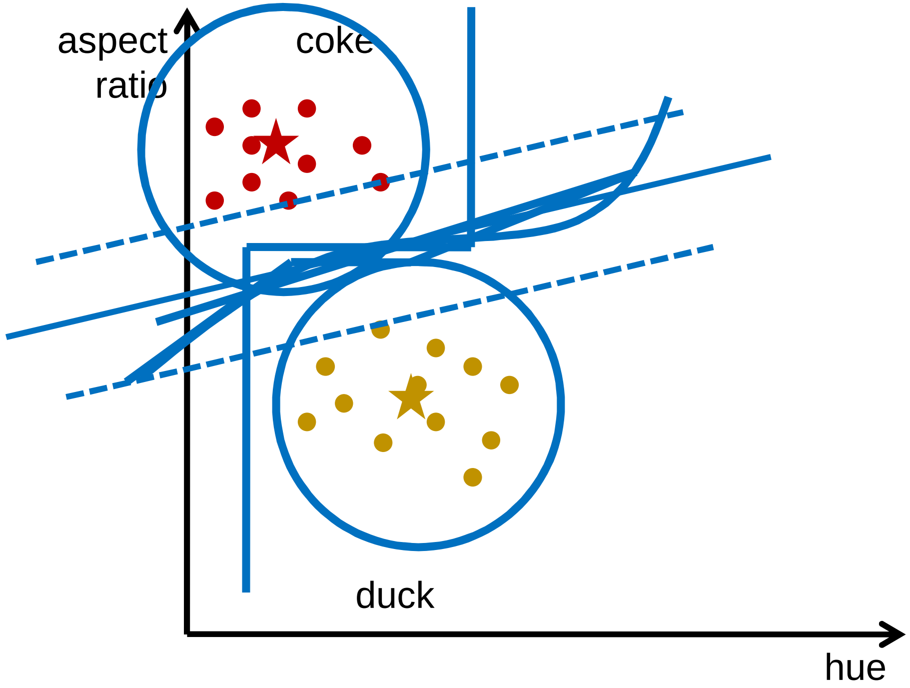
Classification

- learning from examples
  - collecting images of objects
  - creating a feature vector for each object (“*pattern*”)
  - find a decision rule that distinguishes between feature vectors of different classes
  - the process of creating a decision rule from example patterns is called “*learning*” or “*training*”



Classification cont.

- many approaches for decision rules and learning
  - linear classification
  - artificial neural networks/ deep learning
  - prototype-based methods
  - case based reasoning
  - decision trees
  - support vector machines
  - boosting (meta algorithm)
  - …



- in this lecture:

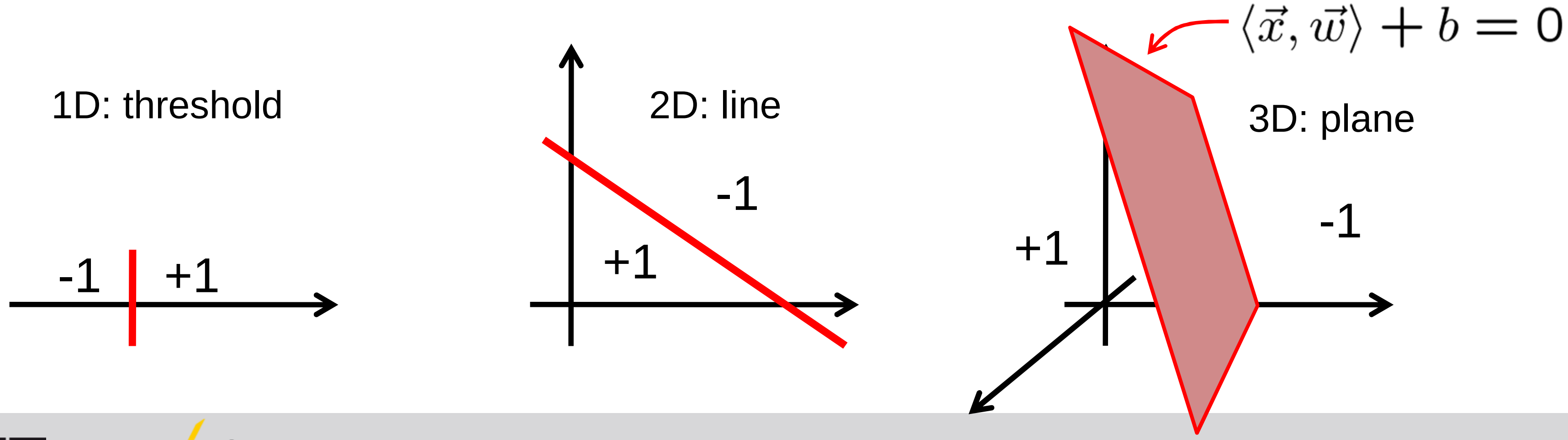
  linear classification, support vector machines, boosting, decision trees, deep learning

Linear Classification

- a linear classifier is a function that implements a function of the kind:

$$\vec{x} \mapsto \begin{cases} +1 & \text{if } \langle \vec{x}, \vec{w} \rangle + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

  - $\vec{w}$ is the *weight vector* of the linear classifier
  - b is the *bias weight* of the classifier
- a linear classifier subdivides an input space into two half spaces. The decision boundary is a hyperplane



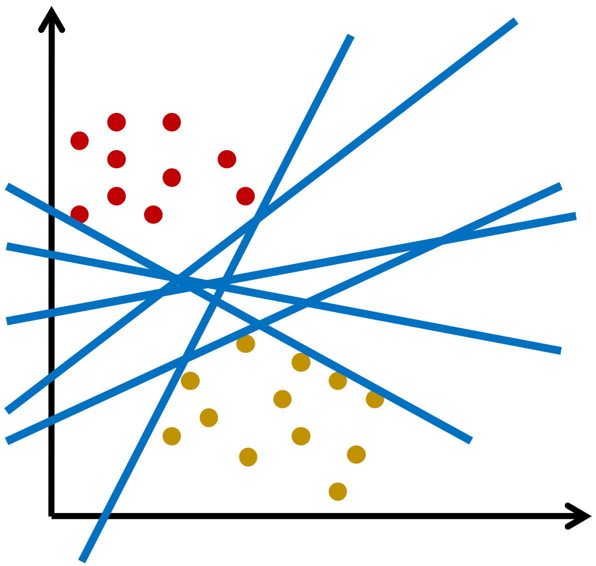
Linear Classification cont.

- learning task:
  - given a set of training examples $\{(\vec{x}^{(1)}, d^{(1)}), \ldots, (\vec{x}^{(p)}, d^{(p)})\}$

    $d^{(i)} = +1$ for examples belonging to the one class ("positive examples")

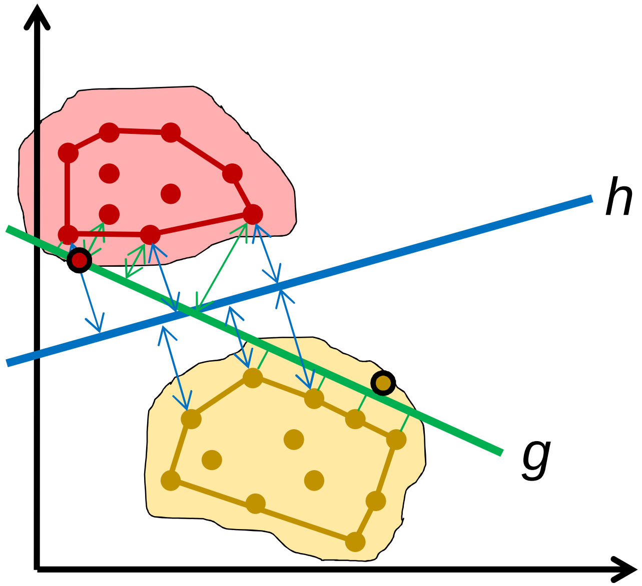
    $d^{(i)} = -1$ for examples belonging to the other class ("negative examples")
  - find $\vec{w}$ and b so that:

    $$d^{(i)} \cdot (\langle \vec{x}^{(i)}, \vec{w} \rangle + b) > 0 \qquad \text{for all } i \in \{1, \ldots, p\}$$
  - many possible solutions, which one is the best?



Linear Classification cont.

- many possible solutions, which one is the best?
  - *g* and *h*, both don't make classification errors
  - *g* has shorter distance to patterns than *h*
  - risk of misclassification for new pattern is larger for *g* than for *h*
  - (unknown) support of the class probability distributions is similar to convex hull of training examples



Maximising the distance of the separating hyperplane to the convex hull of the training patterns means minimising the risk of misclassification (result from computational learning theory)

Linear Classification cont.

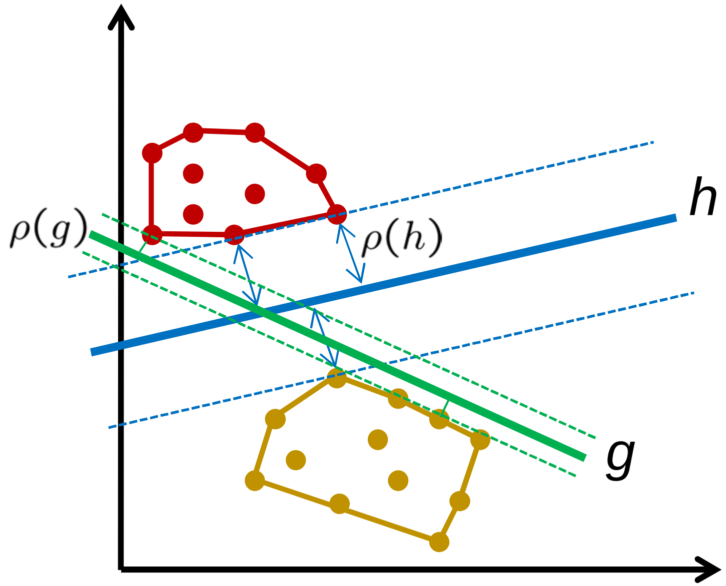
- margin:

  the minimal distance between a hyperplane and the convex hull of the training patterns

$$\rho = \min_i \left(d^{(i)} \cdot \frac{\langle \vec{x}^{(i)}, \vec{w} \rangle + b}{||\vec{w}||} \right)$$

- support vector machine (SVM):

  linear classifier that maximises the margin



SVM

- support vector machine (SVM):

  training a SVM means solving:

$$\begin{aligned} &\underset{\rho,\vec{w},b}{maximise} \quad \rho^2 \\ &subject\ to \quad d^{(i)} \cdot \frac{\langle \vec{x}^{(i)}, \vec{w}\rangle + b}{||\vec{w}||} \geq \rho \qquad \text{for all } i \\ &\qquad\qquad\quad \rho > 0 \end{aligned}$$

  – one degree of freedom: $||\vec{w}||$

  – for convenience set $||\vec{w}|| = \frac{1}{\rho}$

$$\begin{aligned} &\underset{\vec{w},b}{minimise} \quad \frac{1}{2}||\vec{w}||^2 \\ &subject\ to \quad d^{(i)} \cdot (\langle \vec{x}^{(i)}, \vec{w}\rangle + b) \geq 1 \qquad \text{for all } i \end{aligned}$$

SVM cont.

- a simple example:
  - patterns are 1D:
    positive: 5, 10
    negative: -1, 2
  - parameters: w_1, b
  - optimization problem:

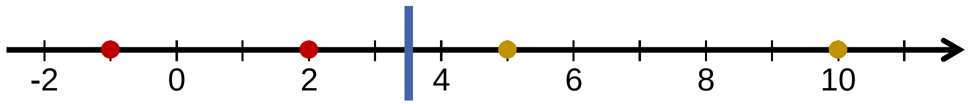
$$\underset{w_1, b}{\text{minimise}} \quad \frac{1}{2} w_1^2$$

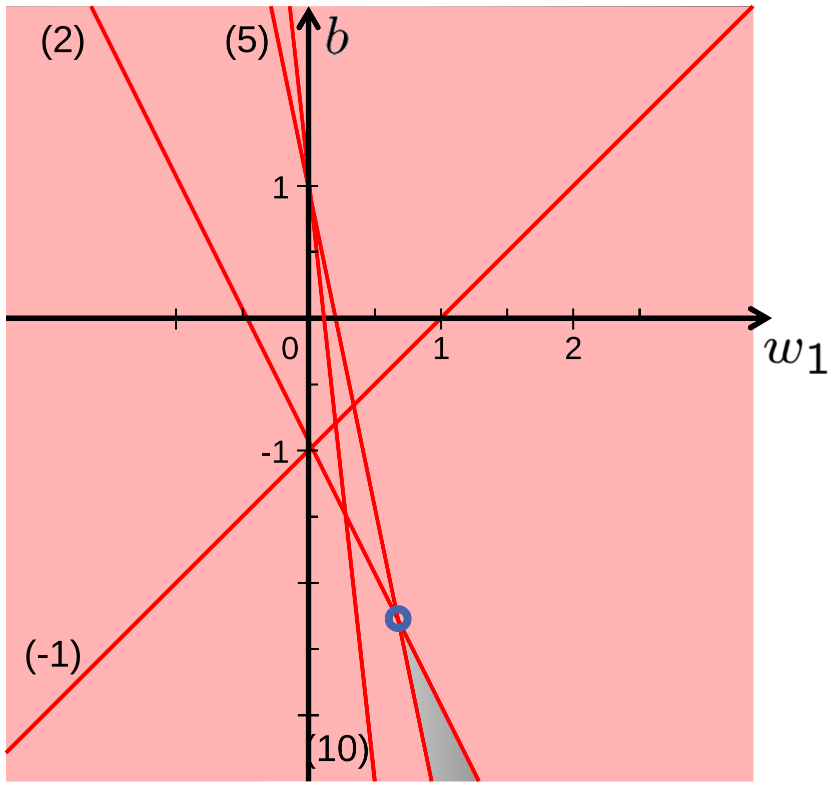
$$\text{subject to} \quad b \geq 1 - 5w_1$$

$$b \geq 1 - 10w_1$$

$$b \leq -1 + w_1$$

$$b \leq -1 - 2w_1$$





SVM cont.

- how to train a SVM?

$$\underset{\vec{w},b}{minimise} \quad \frac{1}{2}||\vec{w}||^2$$

$$subject\ to \quad d^{(i)} \cdot (\langle \vec{x}^{(i)}, \vec{w} \rangle + b) \geq 1 \qquad \text{for all } i$$

- … skipping all details …
  - theory of Lagrange multipliers applies
  - one Lagrange multiplier α_i per training pattern $\vec{x}^{(i)}$

  - the solution is completely described by the Lagrange multipliers
  - many Lagrange multipliers are zero
  - algorithms exist to calculate the Lagrange multipliers

SVM cont.

- the solution:

$$\vec{w} = \boxed{\sum_i \alpha_i d^{(i)} \vec{x}^{(i)}}$$

$$b = d^{(j)} - \langle \vec{x}^{(j)}, \vec{w} \rangle = \boxed{d^{(j)} - \sum_i \alpha_i d^{(i)} \langle \vec{x}^{(j)}, \vec{x}^{(i)} \rangle}$$

for j with $\alpha_j \neq 0$

- margin:

$$\rho = \frac{1}{||\vec{w}||} = \boxed{\frac{1}{\sqrt{\sum_i \sum_j \alpha_i \alpha_j d^{(i)} d^{(j)} \langle \vec{x}^{(i)}, \vec{x}^{(j)} \rangle}}}$$

- classifying a new pattern $\vec{x}_{new}$:

$$sign\left(\langle \vec{x}_{new}, \vec{w} \rangle + b \right)$$
$$= \boxed{sign\left(\sum_i \alpha_i d^{(i)} \langle \vec{x}_{new}, \vec{x}^{(i)} \rangle + b \right)}$$

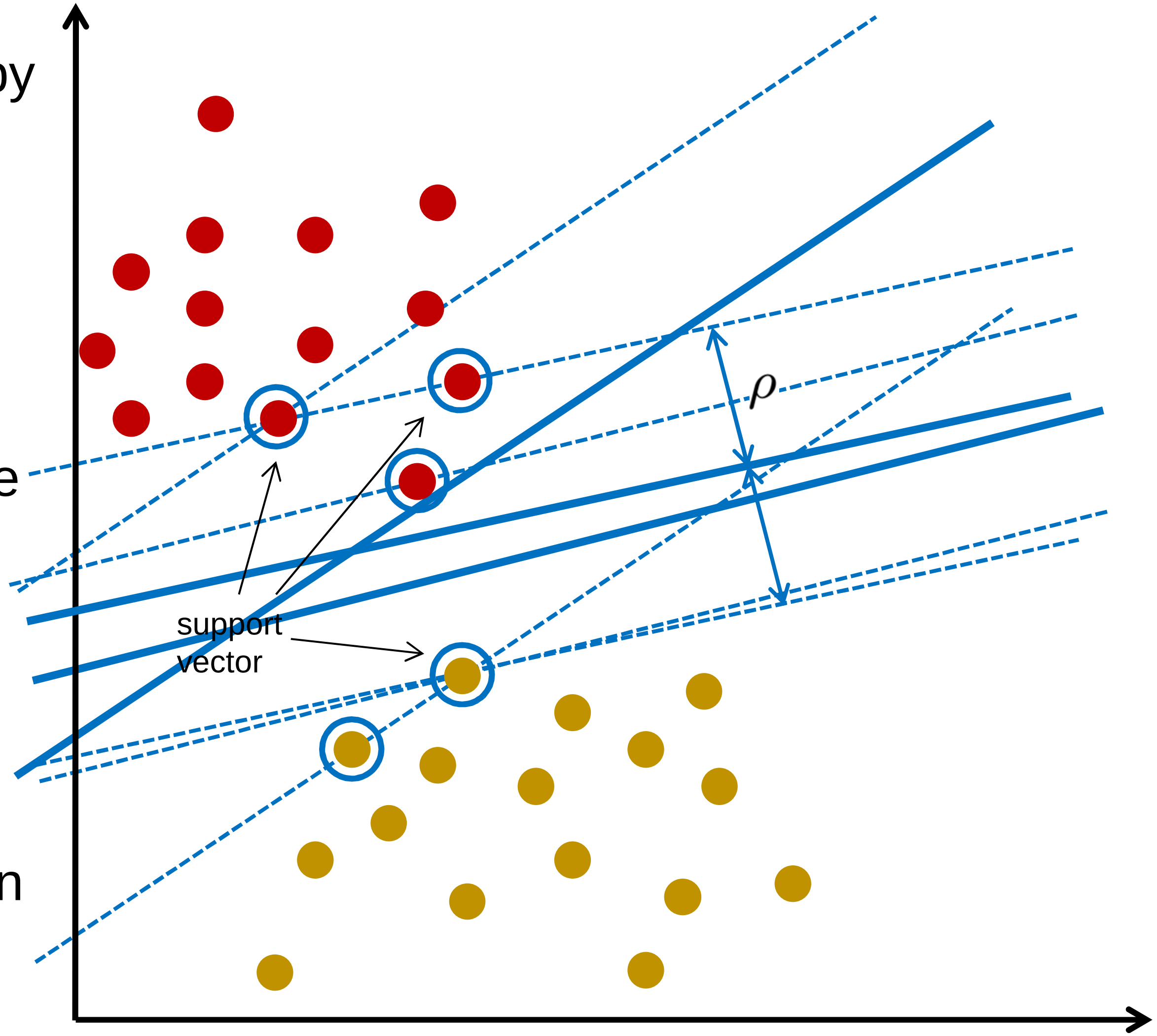
only support vectors ($\alpha_i \neq 0$) contribute

can be expressed without using $\vec{w}$

patterns $\vec{x}$ occur only inside dot products

SVM cont.

- optimal separating hyperplane determined by support vectors
- removing non-support vectors does not change solution
- adding patterns with distance of more than the margin does not change solution
- removing support vector changes solution
- adding pattern with distance less than margin changes solution

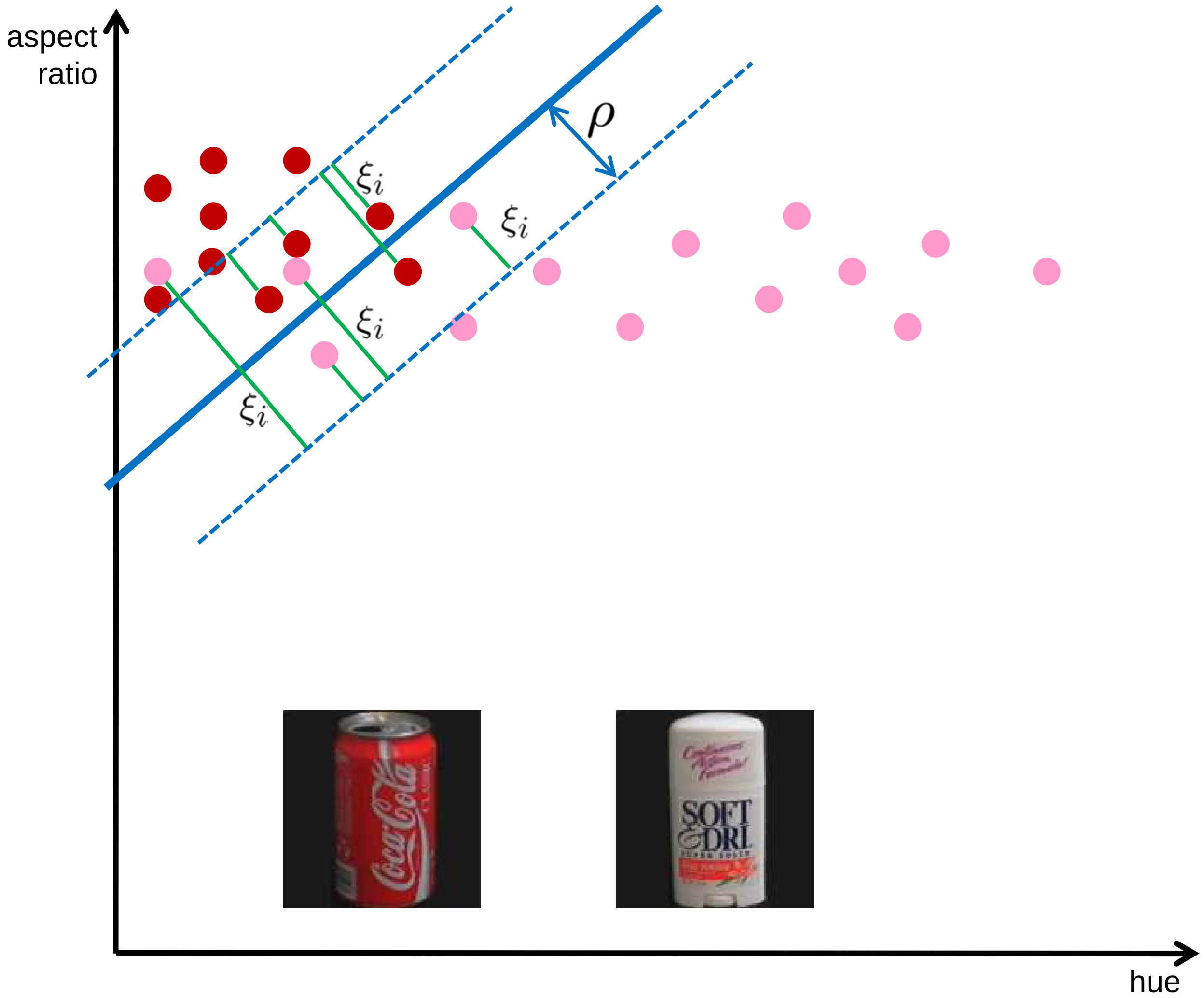


Fault-tolerant SVMs

- overlapping classes force to make errors
- individual errors ξ_i
- conflicting targets:

  maximise ρ

  minimise ξ_i



Fault-tolerant SVMs cont.

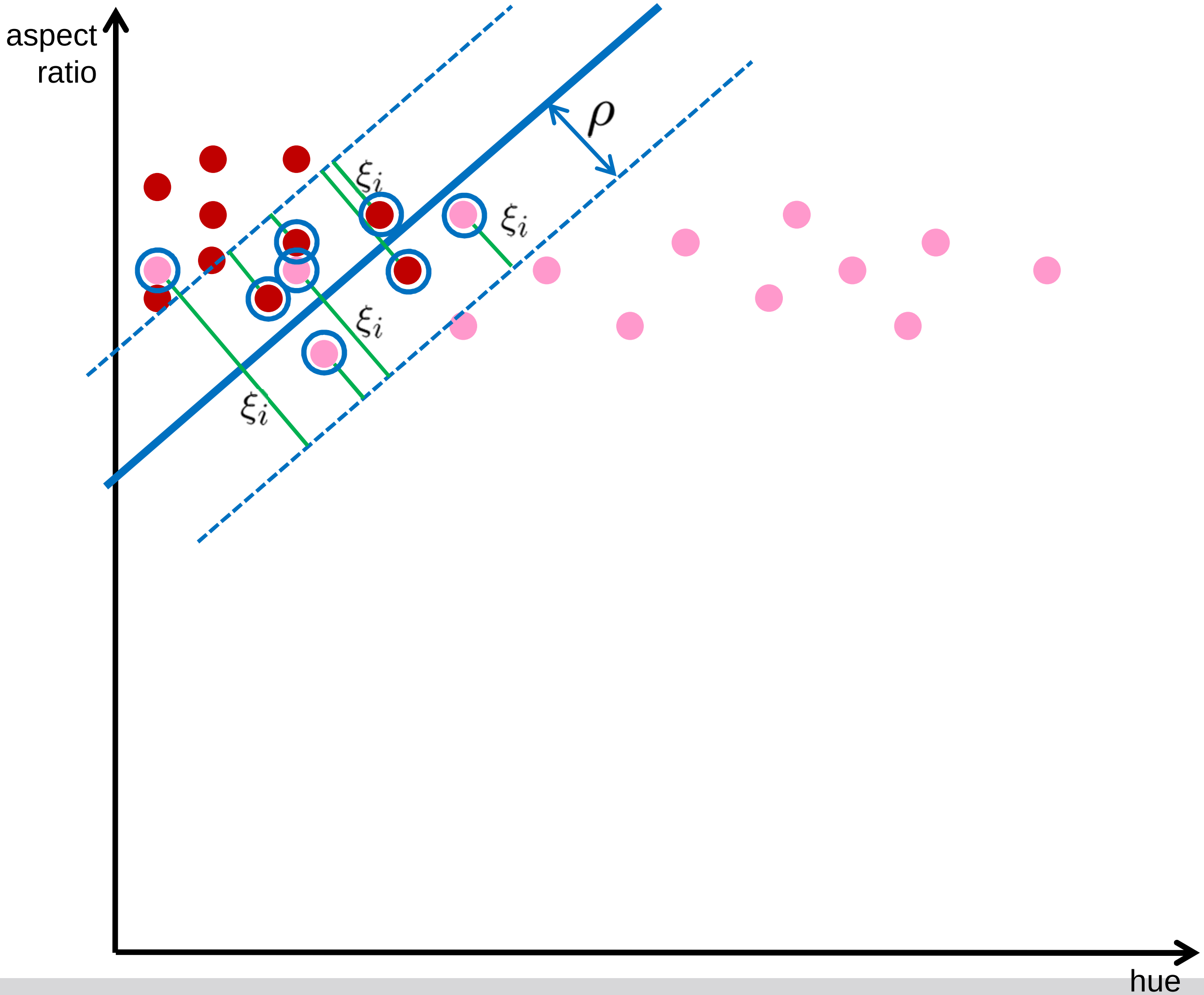
- optimization problem:

$$\begin{aligned} &\underset{\vec{w},b}{minimise} \quad \frac{1}{2}||\vec{w}||^2 \boxed{+ C\sum_i \xi_i} && \text{penalize errors} \\ &subject\ to \quad d^{(i)} \cdot (\langle \vec{x}^{(i)}, \vec{w}\rangle + b) \geq 1 \boxed{- \xi_i} \qquad \text{for all } i && \text{allow errors} \\ &\qquad\qquad\quad \xi_i \geq 0 \qquad \text{for all } i \end{aligned}$$

- *C>0*: regularization parameter controls balance between small errors and large margin (has to be chosen manually)
- fault-tolerant SVMs are known as *“soft-margin-SVMs”* (in contrast to *“hard-margin-SVMs”*)

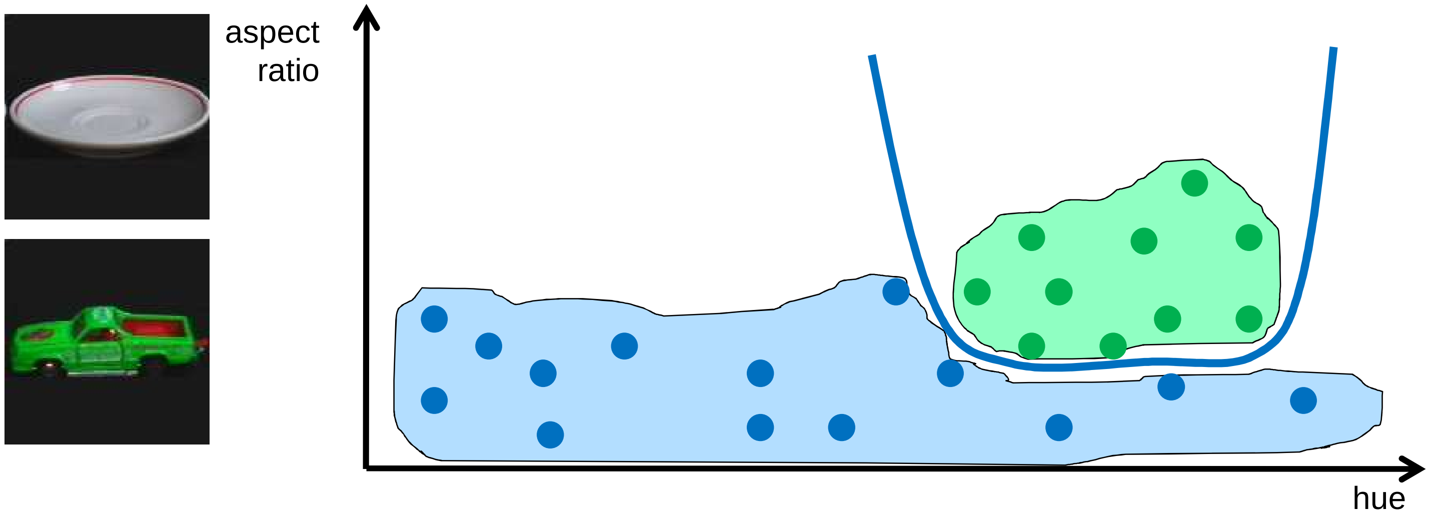
Fault-tolerant SVMs

- similar solution as in the hard-margin case
- support vectors are all patterns that create an individual error or which are on the boundary of the margin area



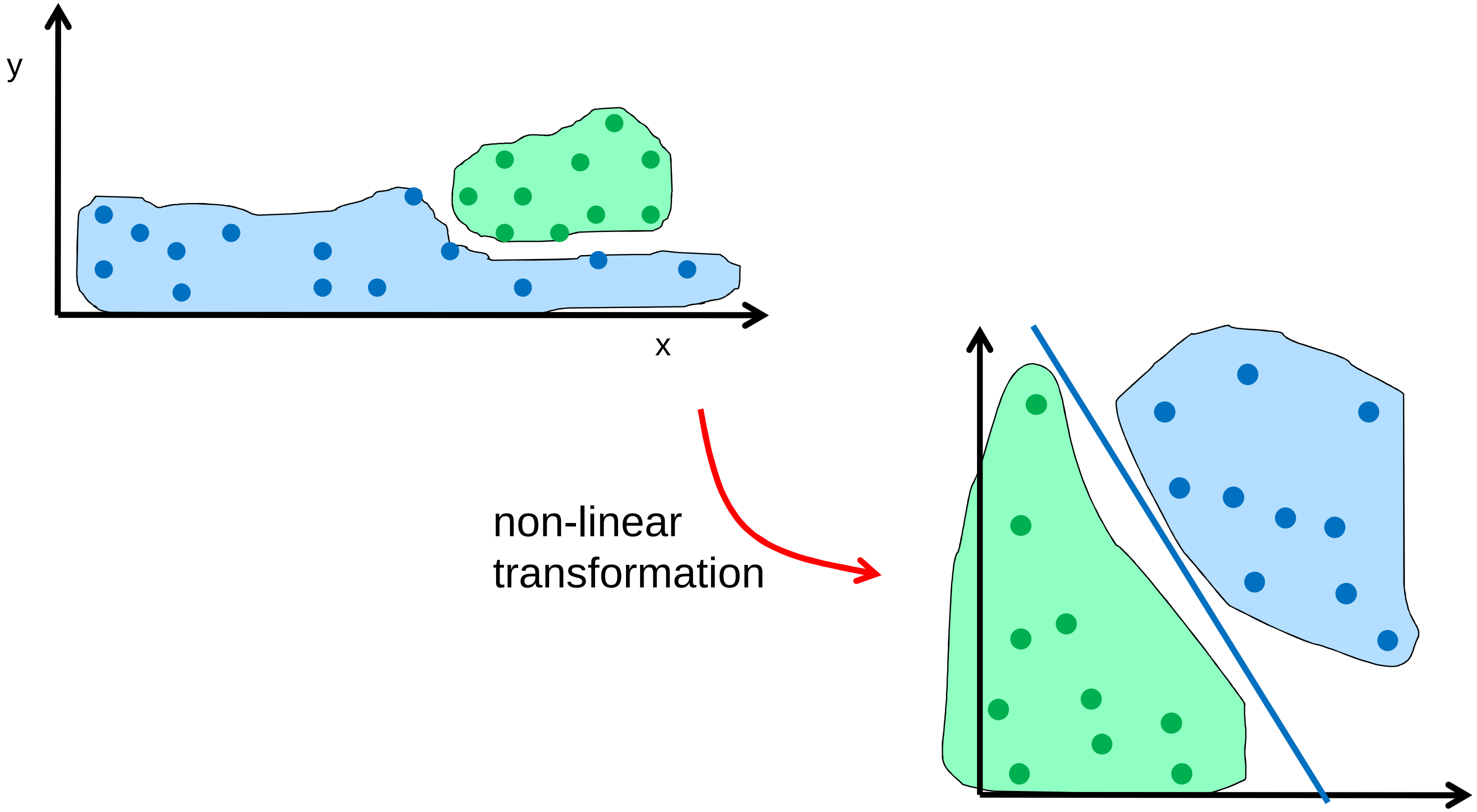
Nonlinear SVMs

- classes with non-overlapping support might not be linearly separable → non-linear classifier
- direct way: use circle/ellipse/non linear curve for classification → difficult to analyse
- indirect way: transform data non-linearly and classify transformed data instead



Nonlinear SVMs cont.

- a non-linear problem might become linear after non-linear transformation



Nonlinear SVMs cont.

- assume nonlinear transformation

$$\Phi : \begin{cases} \mathbb{R}^n \longrightarrow \mathbb{R}^m \\ \vec{x} \mapsto \Phi(\vec{x}) = \vec{X} \end{cases}$$

- find SVM that solves:

$$\underset{\vec{W},b}{minimise} \quad \frac{1}{2}||\vec{W}||^2$$

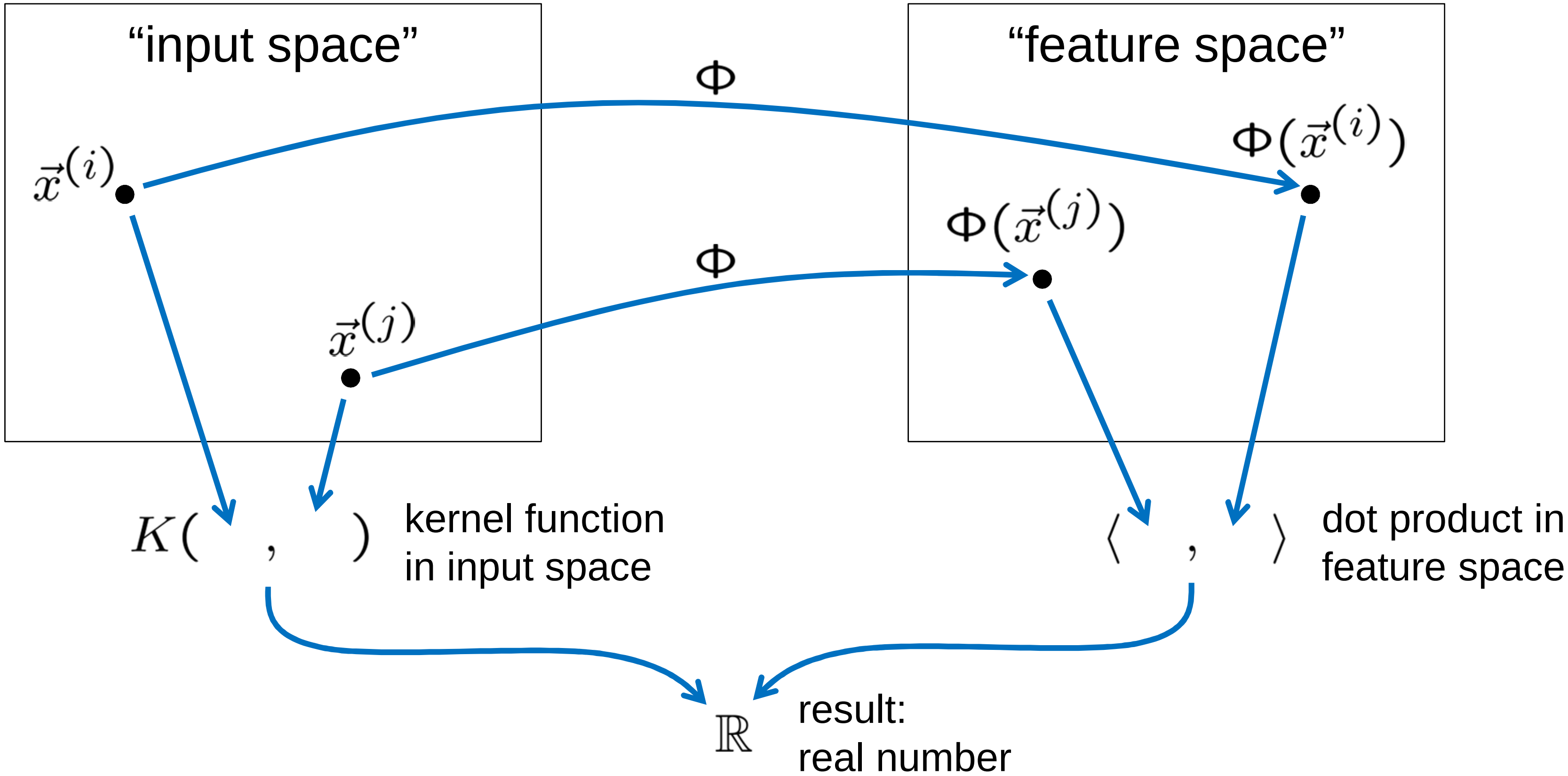
$$subject\ to \quad d^{(i)} \cdot (\langle \vec{X}^{(i)}, \vec{W} \rangle + b) \geq 1 \qquad \text{for all } i$$

- solution is completely determined knowing the Lagrange multipliers *(cf. slide 18)*
  - *don't need do calculate* $\vec{W}$
  - *patterns only occur pairwise as arguments of dot products*

$$\langle \vec{X}^{(i)}, \vec{X}^{(j)} \rangle = \langle \Phi(\vec{x}^{(i)}), \Phi(\vec{x}^{(j)}) \rangle$$

Nonlinear SVMs cont.

- considering $\langle \Phi(\vec{x}^{(i)}), \Phi(\vec{x}^{(j)}) \rangle$



- shortcut: kernel-function $K(\vec{x}, \vec{y}) = \langle \Phi(\vec{x}), \Phi(\vec{y}) \rangle$
- substituting $\langle \Phi(\vec{x}), \Phi(\vec{y}) \rangle$ by $K(\vec{x}, \vec{y})$ eliminates Φ

Kernel Function

- kernel-function: a nasty trick to hide the complexity?
  - example:

$$\Phi(x) = \begin{pmatrix} x^2 \\ x \end{pmatrix}$$

    - evaluating Φ(x) and Φ(y) needs 2 multiplications
    - evaluating dot product in feature space needs 2 multiplications and 1 addition, in total: 4 multiplications, 1 addition

$$K(x, y) = \langle \Phi(x), \Phi(y) \rangle = (xy)^2 + (xy)$$

    - evaluating the kernel function needs 2 multiplications and 1 addition

  - some kernels are based on Hilbert spaces with infinite dimension

Kernel Function cont.

- useful kernel-functions:
  - dot product

$$K(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle$$

  - polynomial kernels

$$K(\vec{x}, \vec{y}) = (\langle \vec{x}, \vec{y} \rangle)^d \text{ or } (\langle \vec{x}, \vec{y} \rangle + 1)^d$$

  - radial basis function (RBF) kernels

$$K(\vec{x}, \vec{y}) = e^{-\frac{||\vec{x}-\vec{y}||^2}{2\sigma^2}}$$

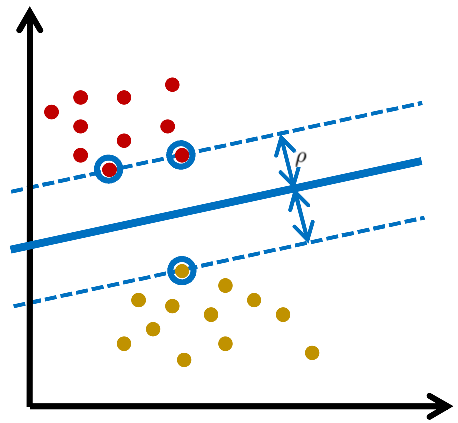
  - Histogram intersection kernel (only for histogram features)

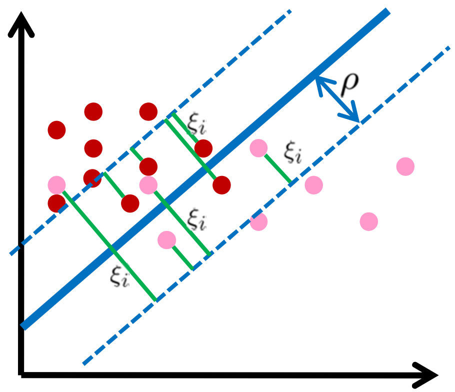
$$K(\vec{x}, \vec{y}) = \sum_i \min\{x_i, y_i\}$$

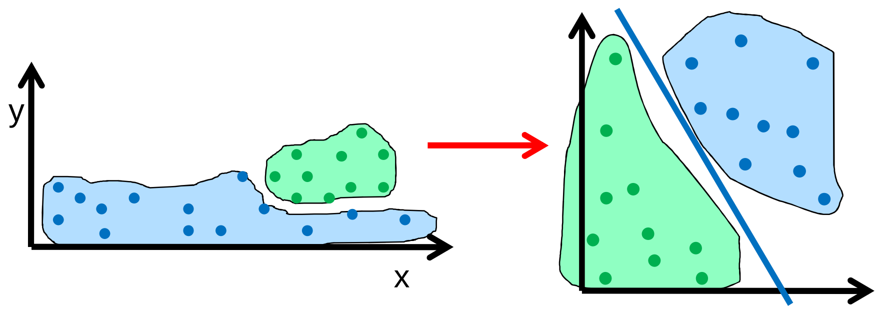
kernel parameters have to be set manually

SVMs cont.

- combining all ideas:
  - SVMs <u>maximise the margin</u> to minimise the risk of misclassification
  - soft-margin SVM allow <u>individual errors</u>. Balance between margin size and errors controlled by parameter *C*
  - <u>kernel functions allow non-linear classification</u> without changing the theoretical framework. Kernel type and kernel parameters control the degree of non-linearity





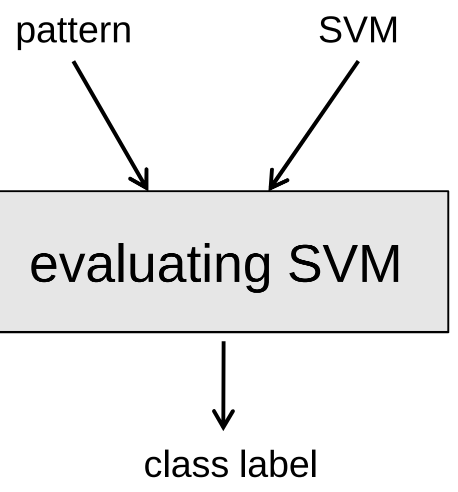


SVMs cont.

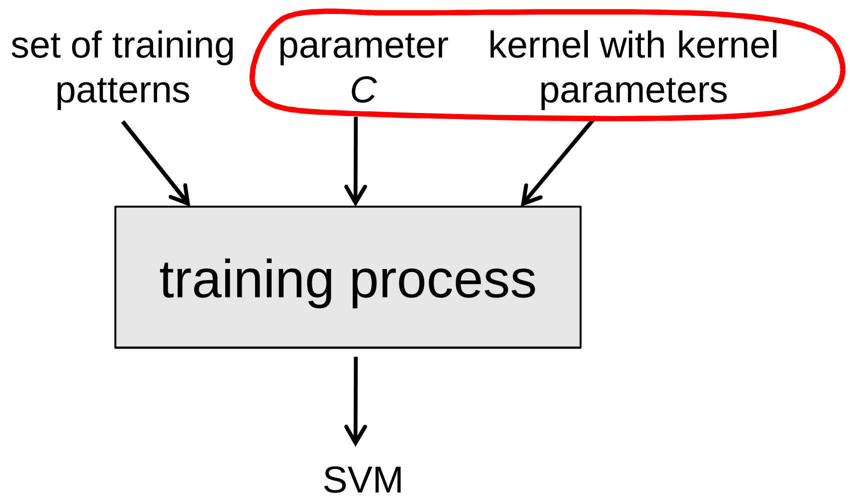
- toy demo

Working with SVMs

- applying a SVM:

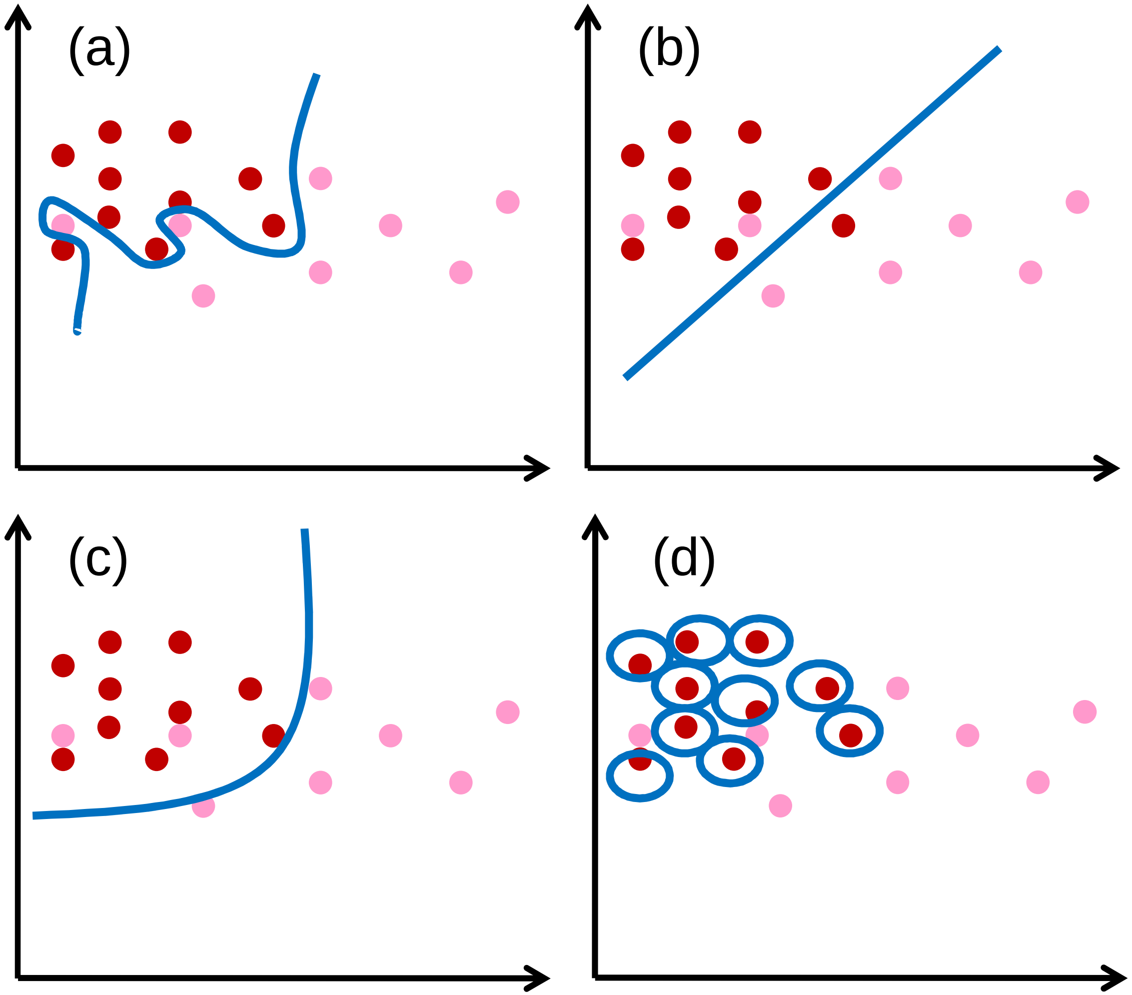


- training a SVM:
  - how can we determine C and kernel?



Validation Process

- which SVM is better?



Validation Process cont.

- expected risk of misclassification:

risk of misclassification = risk of "*false negative*" + risk of "*false positive*"

$$E = \int_{\boxed{A_-}} \boxed{P(+)} \cdot \boxed{p_+(x)}\,dx + \int_{\boxed{A_+}} \boxed{P(-)} \cdot \boxed{p_-(x)}\,dx$$

negative and positive halfspace given by SVM

prior probabilities of positive and negative class

distribution (pdf) of positive and negative class

- *E* unknown, but can be approximated from a sample set

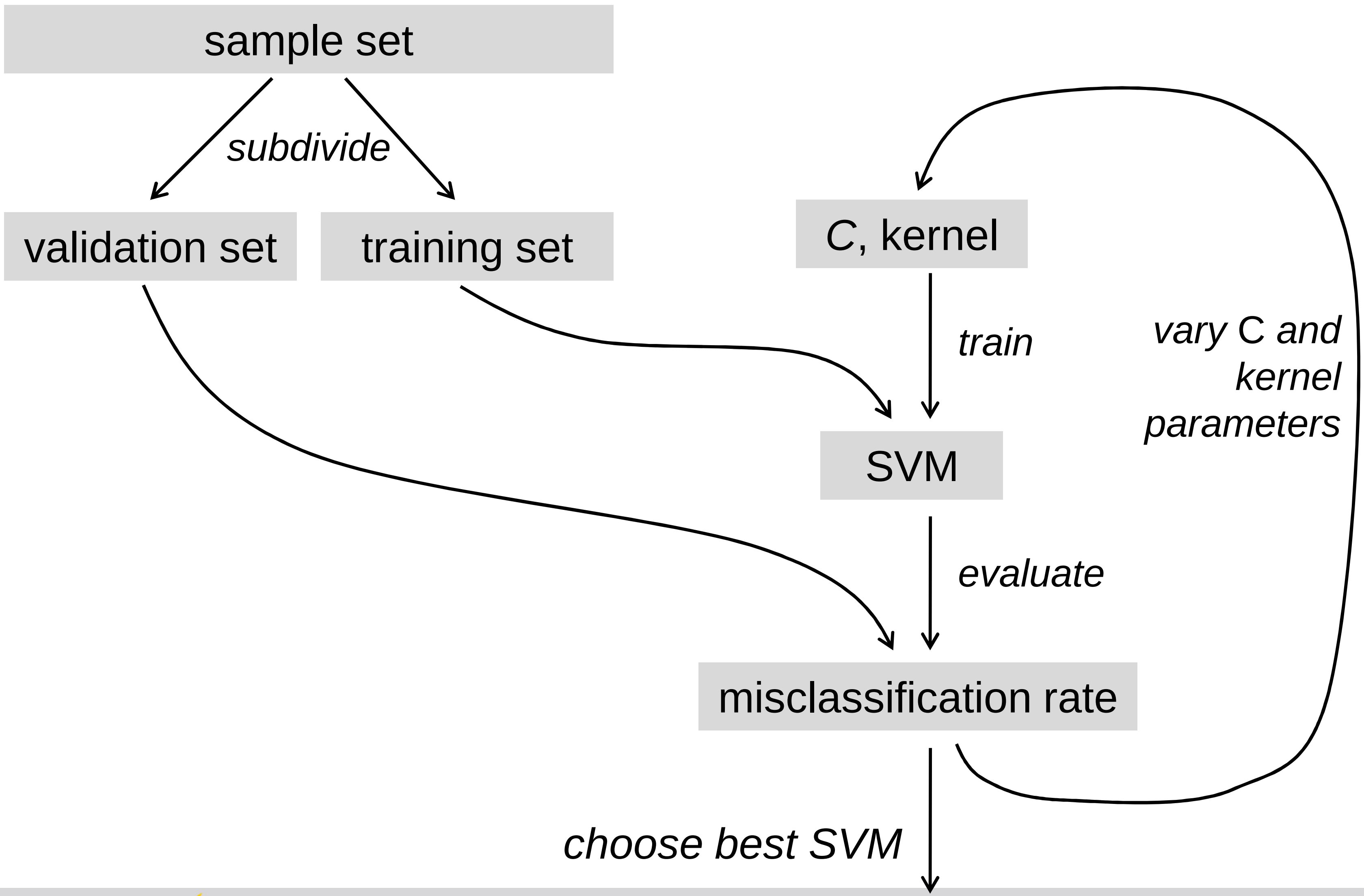
$$E \approx \frac{n_{fn} + n_{fp}}{n}$$

  – n number of elements in sample set
  – n_{fp} number of false positives in sample set
  – n_{fn} number of false negatives in sample set

Validation Process cont.

- validation is a process to test the performance of a classifier on a sample set (“test set”, “validation set”)
- choose the SVM with the smallest misclassification rate on the test set
- test set must be independent of training set!
- validation allows to compare the performance of SVMs trained with different values for *C* and different kernels

Validation Process cont.



Cross-Validation

- disadvantage of validation process:
  - only a part of the data are used for training
  - only a part of the data are used for validation
- *k*-fold cross-validation
  - idea: repeat the training/validation process several times with different training and validation sets
  - *k* is the number of repetitions (between 2 and number of patterns)

Cross-Validation cont.

- *k*-fold cross-validation
  1. subdivide pattern set into *k* disjoint subsets of equal size
  2. repeat for every subset *j*:
     - 2.1. train SVM from subsets *1,…,j-1,j+1,…k*
     - 2.2. evaluate misclassification rate on subset *j*
  3. average misclassification rates

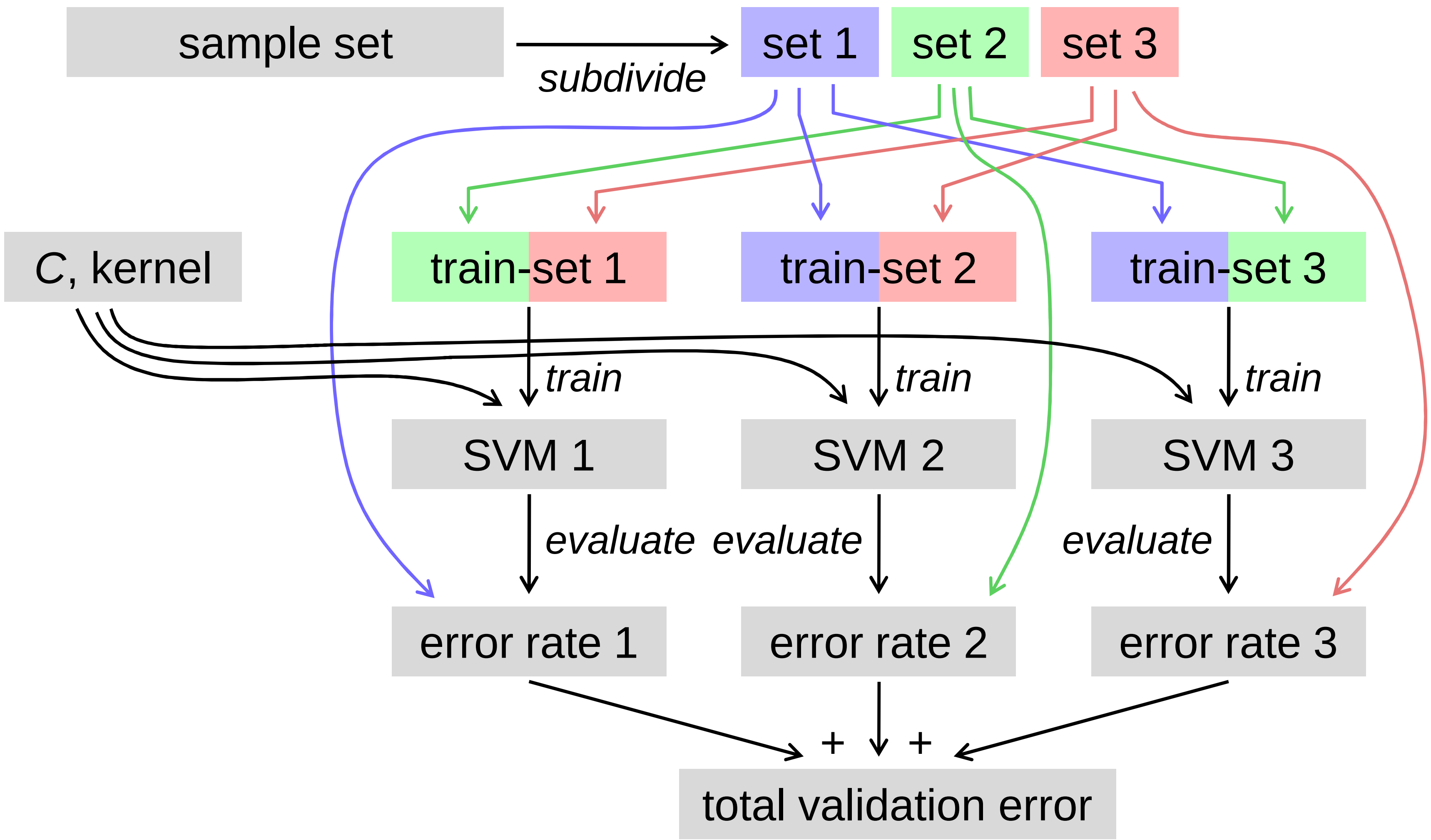
- advantage:
  - all patterns are used for validation
  - training set contains a rate of $\frac{k-1}{k}$ patterns

- if *k* is equal to the total number of patterns
  → *leave-one-out-error*

Karlsruhe Institute of Technology

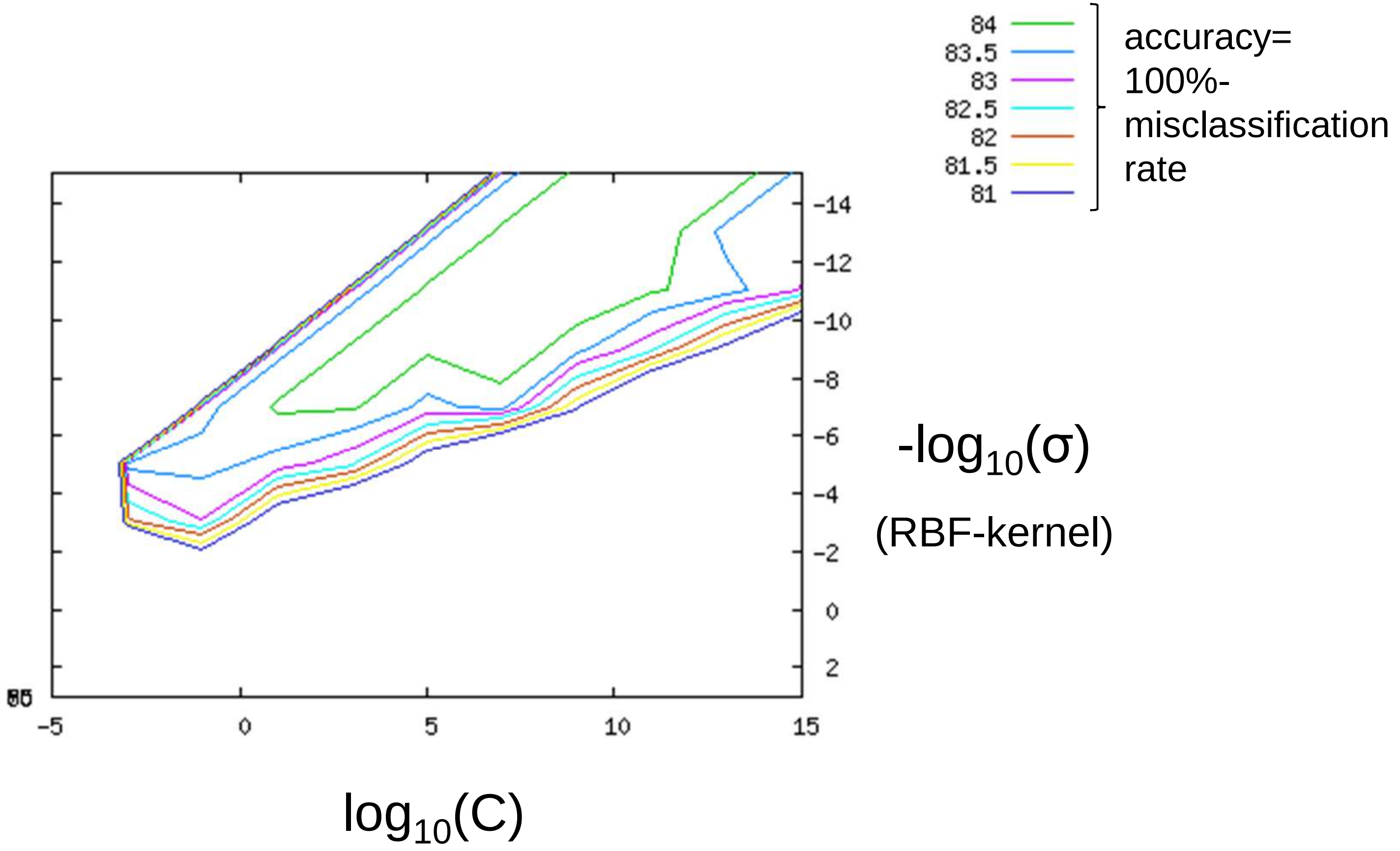
Cross-Validation cont.

- example: 3-fold-cross-validation



Cross-Validation cont.

- possibility to search the parameter space for optimal parameters, e.g.



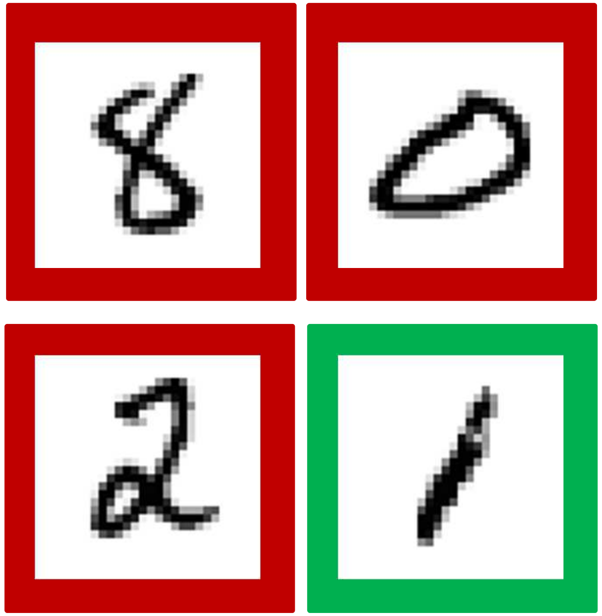
Generalization

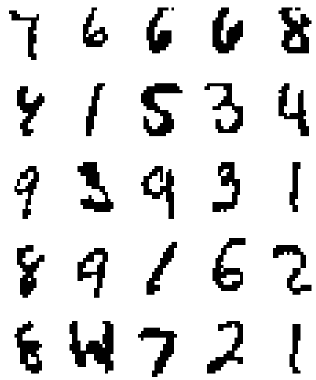
- some concepts you should have heard of:
  - overfitting: a classifier performs well on training data but poor on validation or test data
  - underfitting: a classifier performs poor on both, training and validation data
  - generalization: learn a concept from the training examples that also works on test data, not just memorize the training examples
  - regularization: “help” an overfitted classifier to improve generalization

Experiment: Digit Recognition

- classify images of hand-written digits (US postal zip codes)
- simplified task: classify
  - image shows digit “1”
  - image does not show digit “1”

- here:
  - for training and validating: 500 images of “1”, 500 images of “no-1”
  - for testing: 500 images of “1”, 500 images of “not-1”





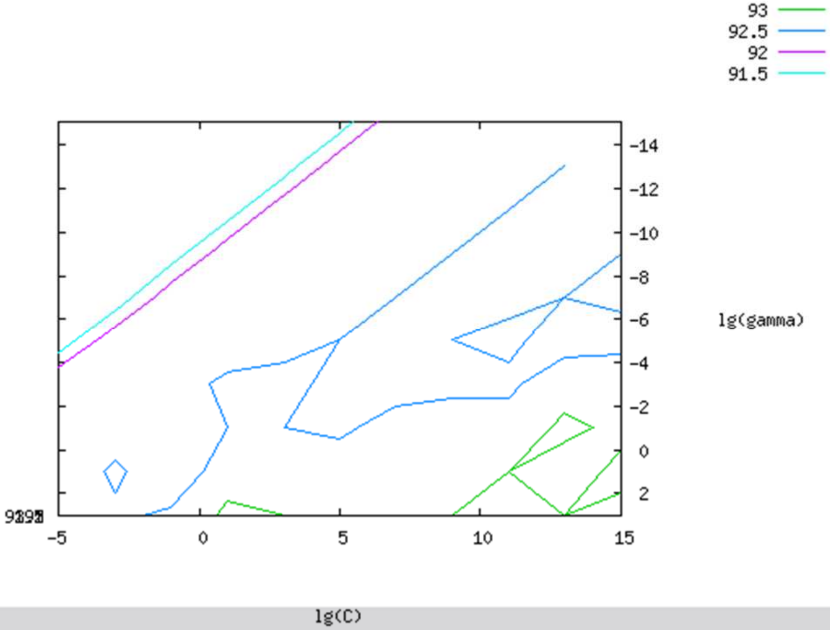
dataset collected by Yann LeCun / http://yann.lecun.com/

Digit Recognition cont.

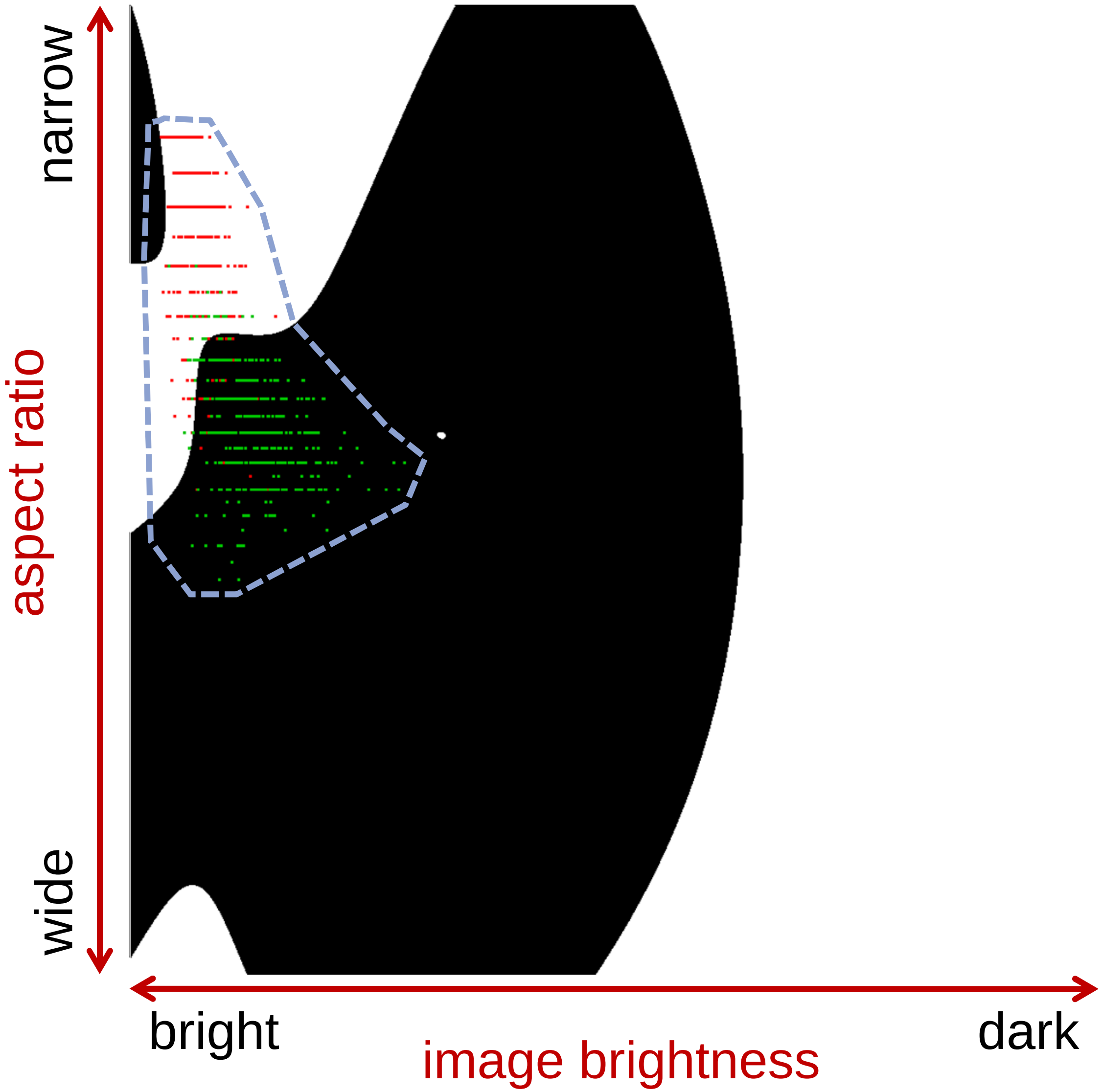
- 1st approach:
  - 2-dimensional patterns
    - average grey value
    - aspect ratio
  - patterns are rescaled to interval [-1, +1]
  - soft-margin SVM with RBF-kernel
  - 5-fold cross validation
  - grid search in parameter space:
    - $10^{-5} \leq C \leq 10^{15}$ (on log scale)
    - $10^{-3} \leq \sigma \leq 10^{15}$ (on log scale)

- accuracy:
  - training set: 93.1%
  - test set: 80.3%
- number of support vectors:

  167 (out of 1000)



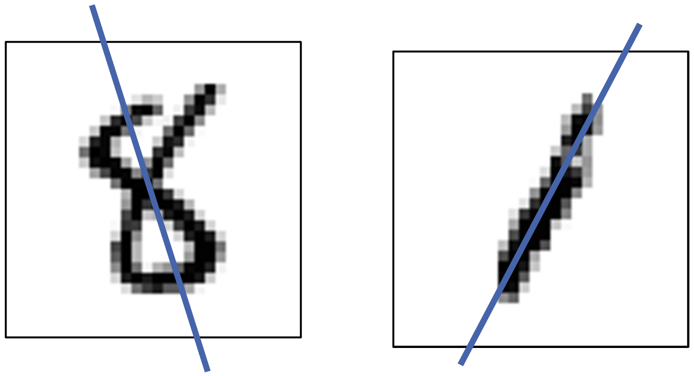
Digit Recognition cont.



- white: “1”
- black: “no-1”

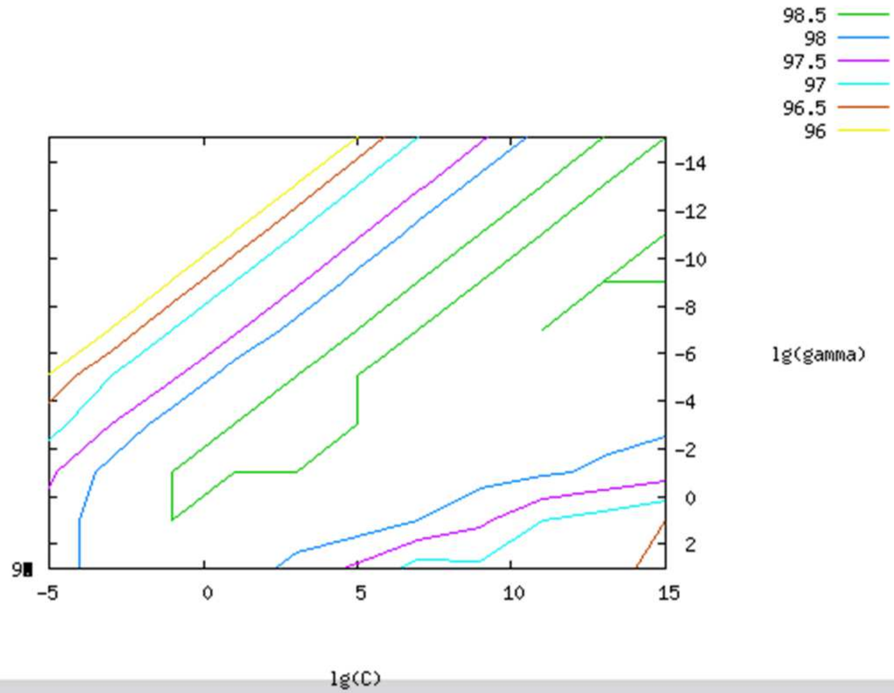
Digit Recognition cont.

- 2nd approach:
  - add a third feature:
    - average distance from line fitted to the dark pixels



- accuracy:
  - training set: 98.5%
  - test set: 98.7%
- number of support vectors:

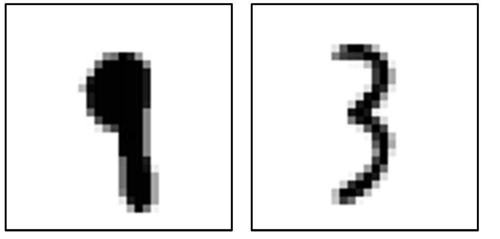
  95 (out of 1000)

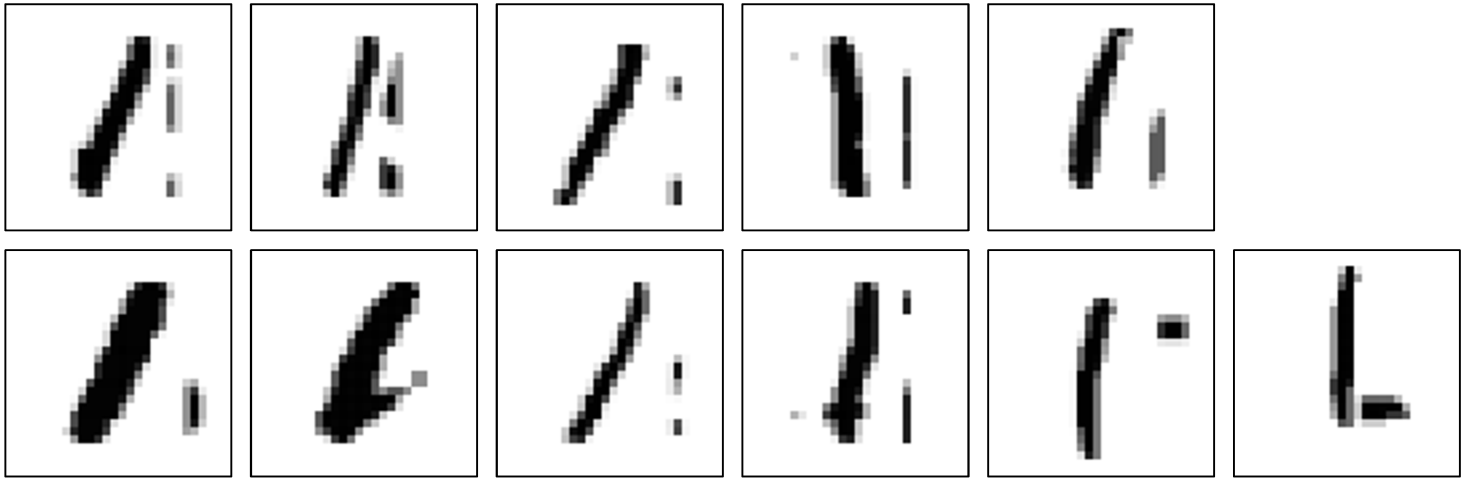


Digit Recognition cont.

- confusion matrix:

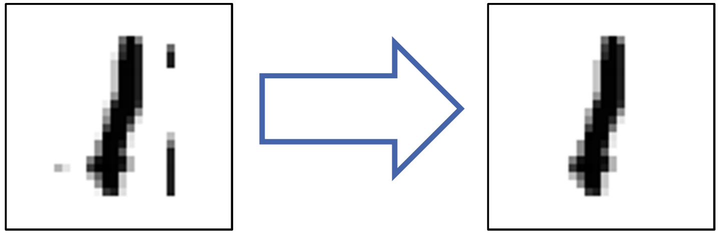
	is 1	is not 1
classified as 1	489	2
classified as not 1	11	498





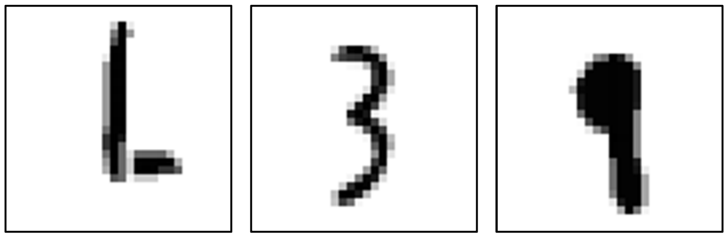
Digit Recognition cont.

- 2nd approach, improvement:
  - find connected components (CCL) and mask out all except the largest segment
  - calculate features from preprocessed image



	is 1	is not 1
classified as 1	499	2
classified as not 1	1	498

- accuracy:
  - training set: 98.5%
  - test set: 99.7%
- number of support vectors:
  95 (out of 1000)



Digit Recognition cont.

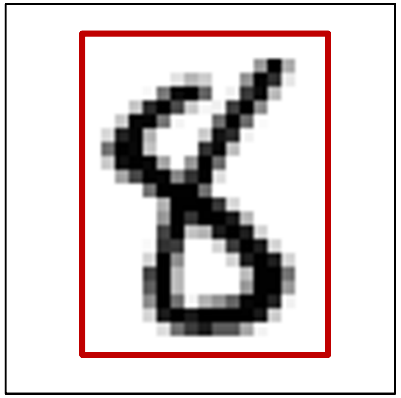
- 3rd approach:
  - resize all images to 28x28 pixels and use grey values of pixels as features → 784-dimensional patterns

- accuracy:
  - training set: 99.0%
  - test set: 98.7%
- number of support vectors:
  220 (out of 1000)

	is 1	is not 1
classified as 1	487	0
classified as not 1	13	500

Digit Recognition cont.

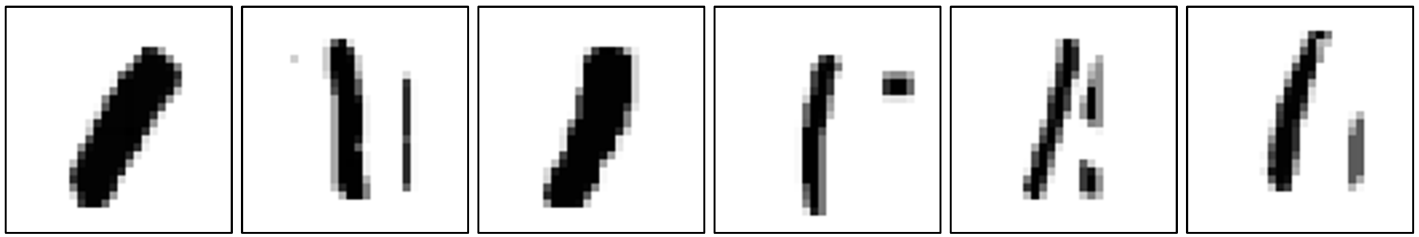
- 3rd approach improvement:
  - observation: a lot of pixels do not contribute to the decision, e.g. boundary pixels
  - use only a subset of all pixels, e.g. a 24x18 subarea → 432-dimensional patterns



- accuracy:
  - training set: 99.0%
  - test set: 99.4%
- number of support vectors:

  219 (out of 1000)

	is 1	is not 1
classified as 1	494	0
classified as not 1	6	500

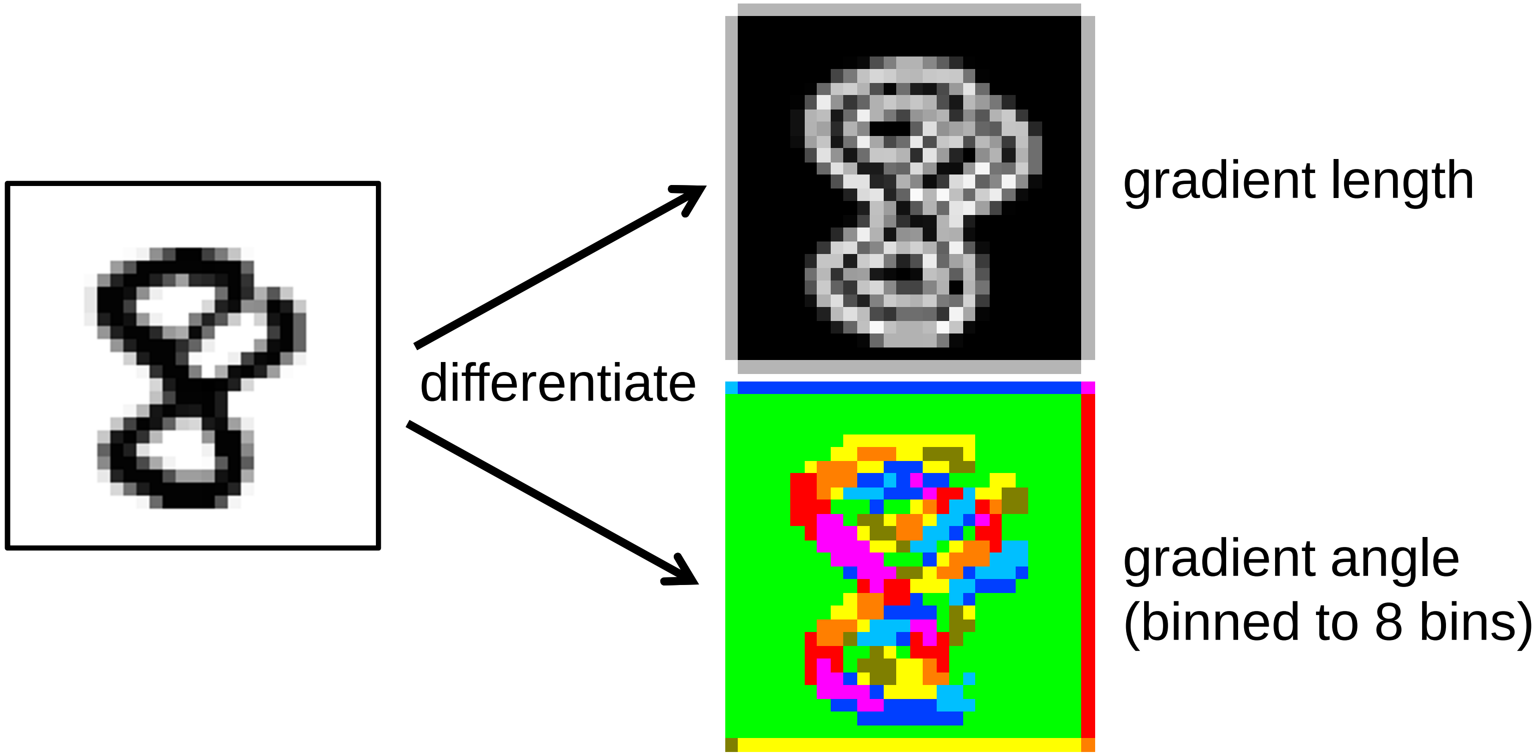


Digit Recognition cont.

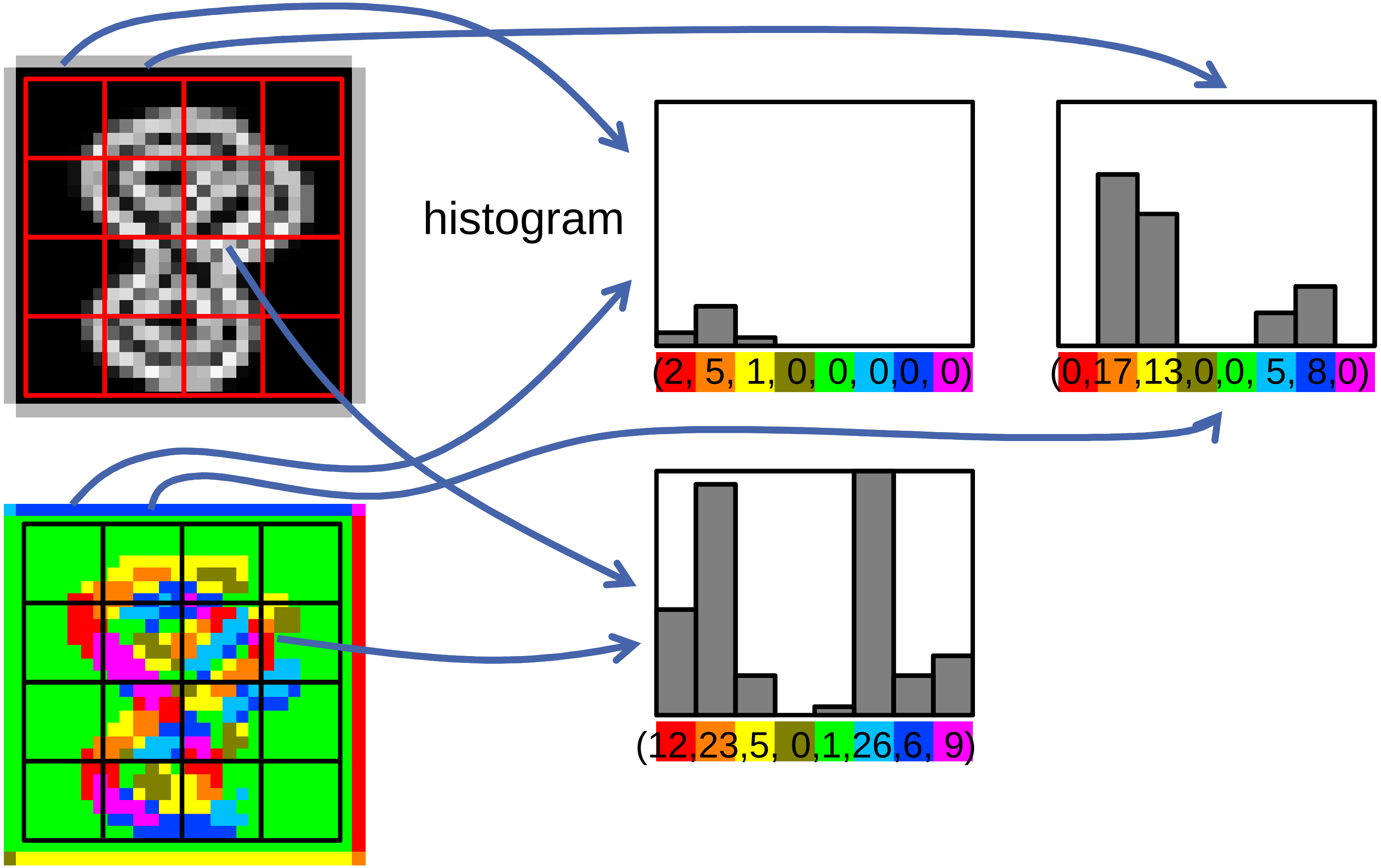
- 4th approach:
  - *HOG*-features

    histogram of oriented gradients (Dalal&Triggs, 2005)

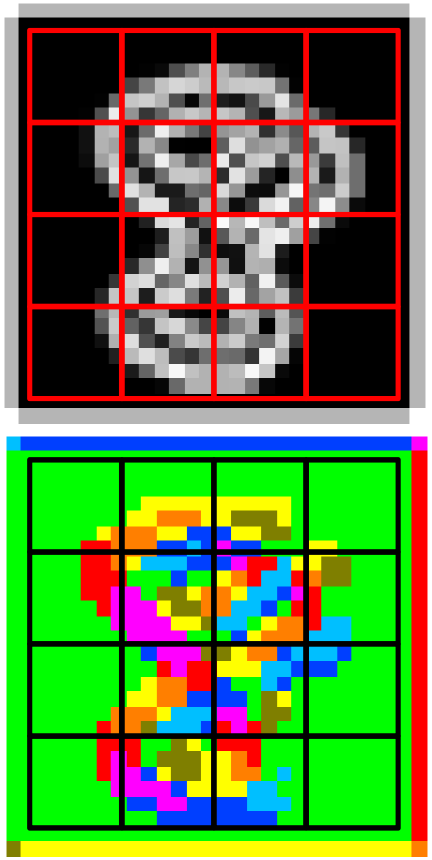
    use gradient information instead of grey levels



Digit Recognition cont.



Digit Recognition cont.



(2, 5, 1, 0, 0, 0,0, 0)

(0,17,13,0,0, 5, 8,0)

⋮

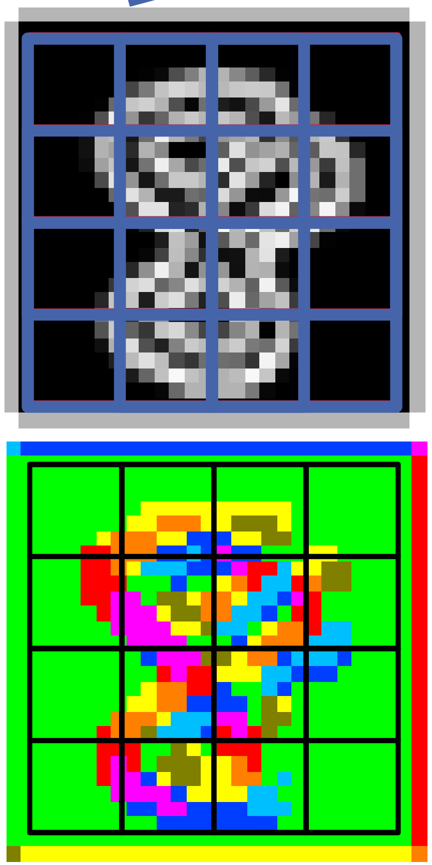
(12,23,5, 0,1,26,6, 9)

⋮

in total: 16 histograms, one for each block

Digit Recognition cont.

– HOG arranges normalized blocks of 4 adjacent cells



- assemble histograms of adjacent cells:

$$\vec{V}_1 = (\underbrace{2,5,1,0,0,0,0,0,}_{\text{from cell 1}}\ \underbrace{0,17,13,0,0,5,8,0,}_{\text{from cell 2}}\ \underbrace{15,0,0,0,0,0,0,7,}_{\text{from cell 5}}\ \underbrace{0,2,4,3,2,3,2,12}_{\text{from cell 6}})$$

- normalize descriptor:

$$\vec{V}_1^{norm} = \frac{\vec{V}_1}{||\vec{V}_1|| + \epsilon}$$

- assemble descriptors of all blocks:

$$\vec{V} = (\vec{V}_1^{norm}, \ldots, \vec{V}_9^{norm})$$

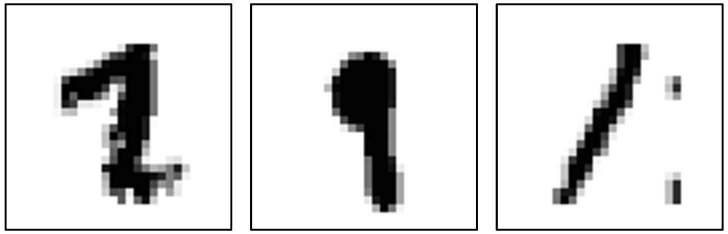
- apply vectors $\vec{V}$ to SVM

Digit Recognition cont.

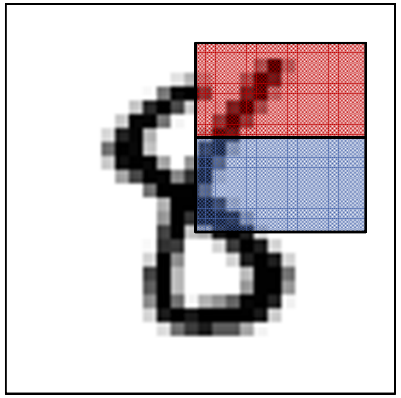
- 4th approach:
  - use only HOG features
    → 288-dimensional patterns

- accuracy:
  - training set: 99.4%
  - test set: 99.7%
- number of support vectors:
  174 (out of 1000)

	is 1	is not 1
classified as 1	499	2
classified as not 1	1	498

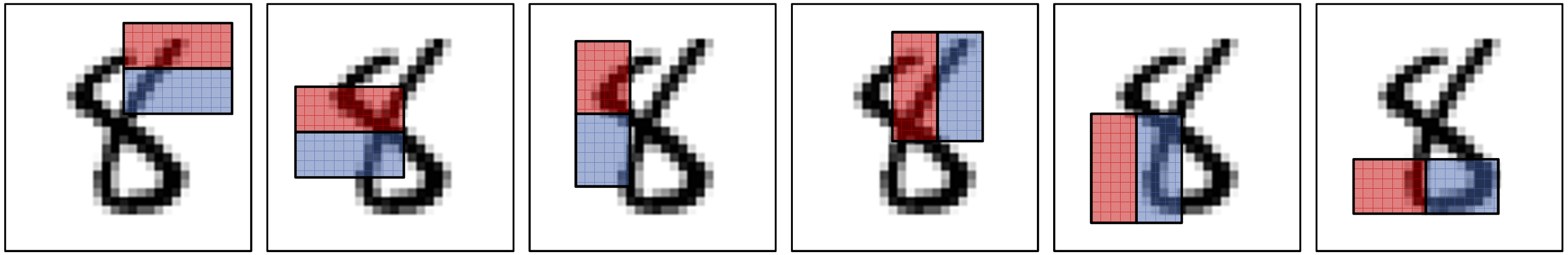


Digit Recognition cont.

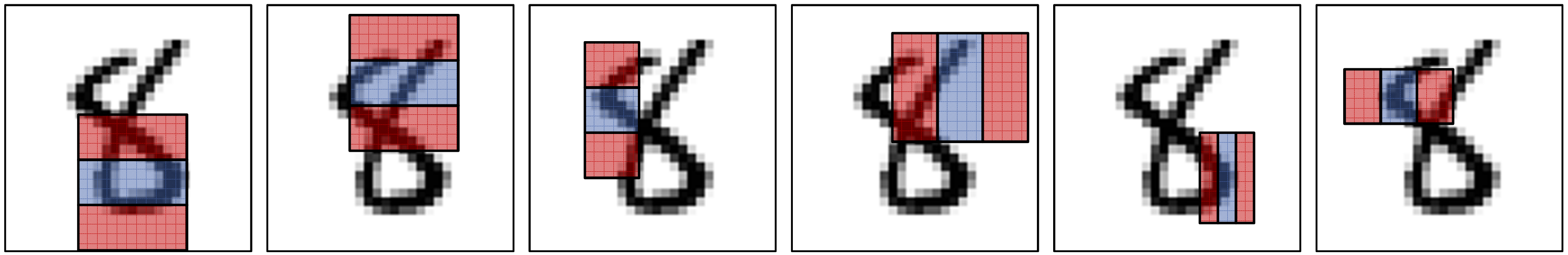


- 5th approach:
  - *Haar features*
    - *compare gray levels of rectangular areas, e.g. average gray level in red area minus average gray level in blue area*
    - *very many possible features*

- edge features

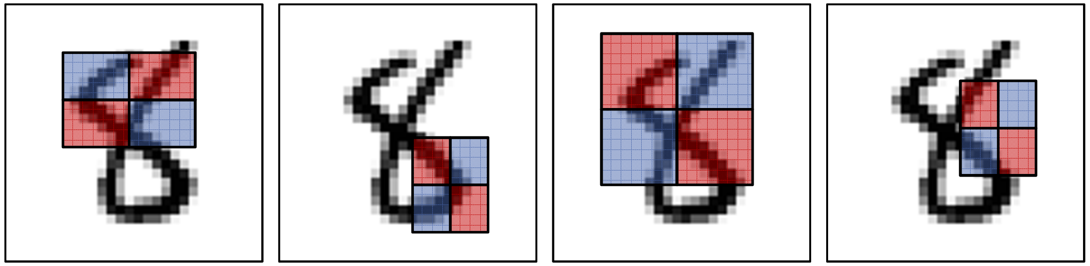


- line features

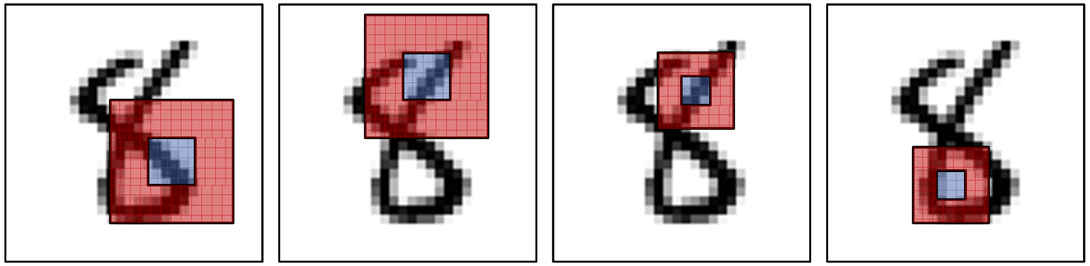


Digit Recognition cont.

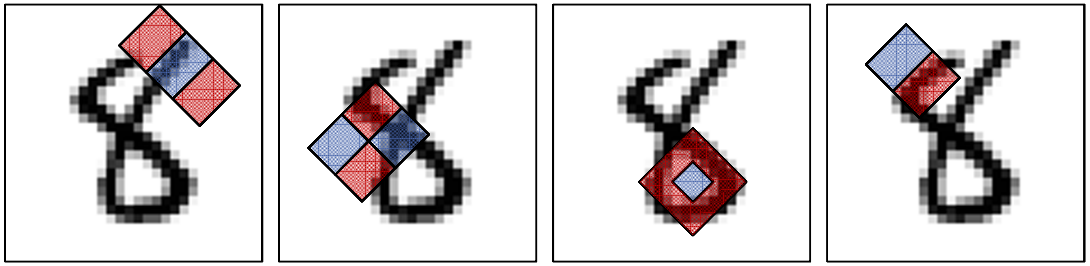
- chessboard features



- center-surround features

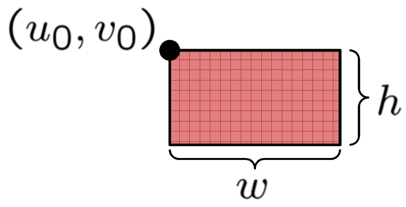


- features with diagonal orientation



Digit Recognition cont.

- Calculating Haar features:
  - the naïve way:

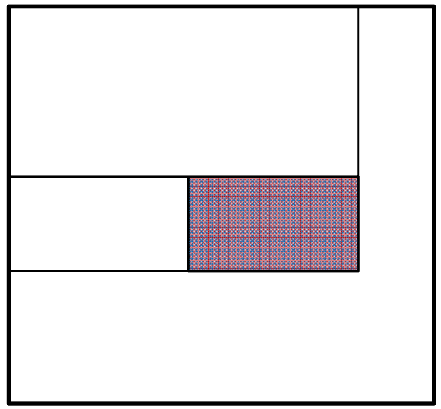


$$s = \sum_{u=u_0}^{u_0+w-1} \sum_{v=v_0}^{v_0+h-1} g(u,v)$$

implementing this with for loops requires $O(w \cdot h)$ operations.

  - the smart way:

$$s = \sum_{u=0}^{u_0+w-1} \sum_{v=0}^{v_0+h-1} g(u,v) - \sum_{u=0}^{u_0-1} \sum_{v=0}^{v_0+h-1} g(u,v) + \sum_{u=0}^{u_0-1} \sum_{v=0}^{v_0-1} g(u,v) - \sum_{u=0}^{u_0+w-1} \sum_{v=0}^{v_0-1} g(u,v)$$



- the integral image:

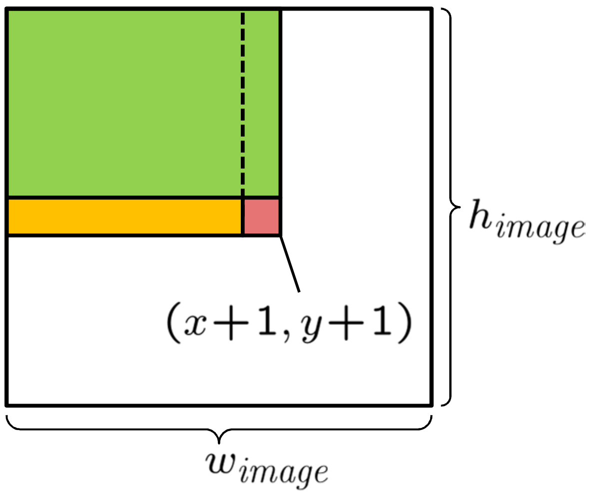
$$I(x,y) := \sum_{u=0}^{x} \sum_{v=0}^{y} g(u,v)$$

- calculating *s* requires 4 operations:

$$\begin{aligned} s =& I(u_0+w-1, v_0+h-1) - I(u_0-1, v_0+h-1) + \\ & I(u_0-1, v_0-1) - I(u_0+w-1, v_0-1) \end{aligned}$$

Digit Recognition cont.

- calculating the integral image



$$I(x,y) := \sum_{u=0}^{x} \sum_{v=0}^{y} g(u,v)$$

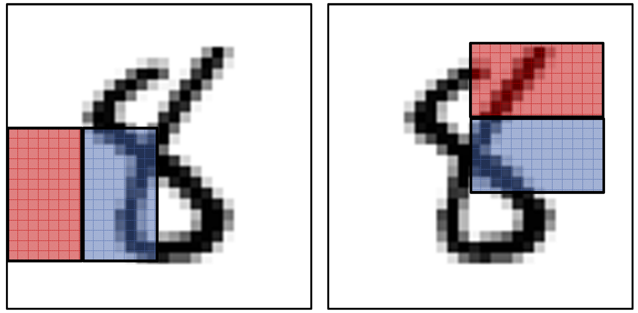
$$\begin{aligned} I(x+1,y+1) &= \sum_{u=0}^{x+1} \sum_{v=0}^{y+1} g(u,v) \\ &= \sum_{u=0}^{x+1} \sum_{v=0}^{y} g(u,v) + \sum_{u=0}^{x} \sum_{v=0}^{y+1} g(u,v) - \sum_{u=0}^{x} \sum_{v=0}^{y} g(u,v) + g(x+1,y+1) \\ &= I(x+1,y) + I(x,y+1) - I(x,y) + g(x+1,y+1) \end{aligned}$$

⇨ yields an iterative algorithm that calculates the whole integral image with $O(w_{image} \cdot h_{image})$ operations

⇨
- naïve approach is superior if you want to calculate one rectangle
- integral image is superior if you want to calculate many rectangles

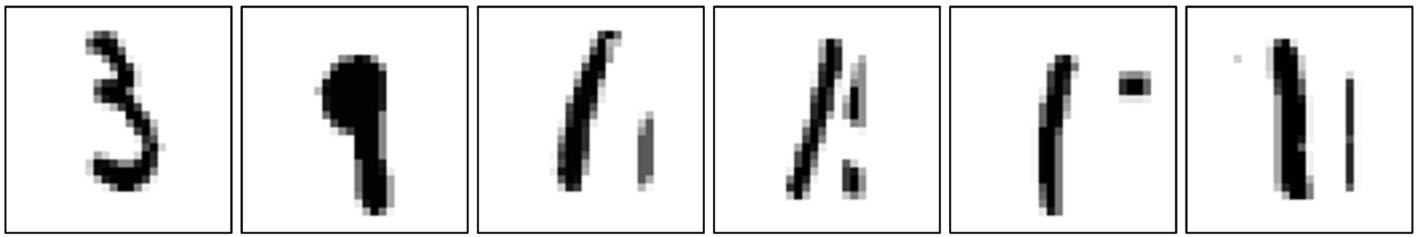
Digit Recognition cont.

- 5^{th} approach:
  - Haar features, use horizontal and vertical edge features at 7x7 positions
    → 98-dimensional patterns



- accuracy:
  - training set: 99.1%
  - test set: 99.4%
- number of support vectors:
  109 (out of 1000)

	is 1	is not 1
classified as 1	496	2
classified as not 1	4	498

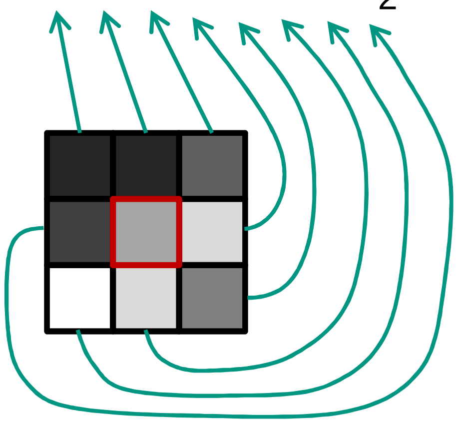


Digit Recognition cont.

- 6^{th} approach:
  - *Local binary patterns (LBP)*
    - *analyze local gray level changes*
    - *perform histograms for multiple areas*

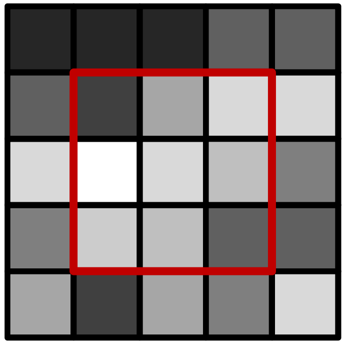
- for each neighboring pixel, check whether neighboring pixel is brighter (1) or darker (0)

$0\ 0\ 0\ 1\ 0\ 1\ 1\ 0_2 = 22_{10}$



Digit Recognition cont.

- 6th approach:
  - *Local binary patterns (LBP)*
    - *analyze local gray level changes*
    - *perform histograms for multiple areas*

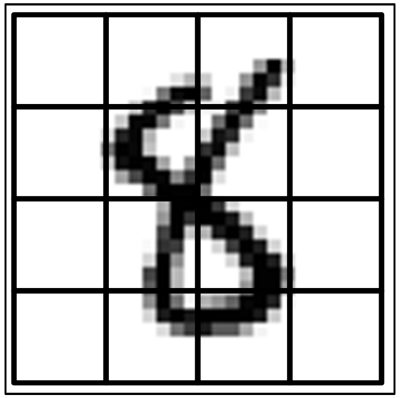


- for each neighboring pixel, check whether neighboring pixel is brighter (1) or darker (0)
- calculate those numbers for all pixels in a block

$00011111_2=31_{10}$
$00010110_2=22_{10}$
$00000000_2=0_{10}$
$00000000_2=0_{10}$
$00100001_2=33_{10}$
$11100001_2=225_{10}$
$11100000_2=224_{10}$
$11100001_2=225_{10}$
$11101111_2=239_{10}$

Digit Recognition cont.

- 6th approach:
  - *Local binary patterns (LBP)*
    - *analyze local gray level changes*
    - *perform histograms for multiple areas*



- for each neighboring pixel, check whether neighboring pixel is brighter (1) or darker (0)
- calculate those numbers for all pixels in a block
- make a histogram
- arrange histograms for all blocks in one vector

31, 22, 0, 0, 33, 225, 224, 225, 239, …

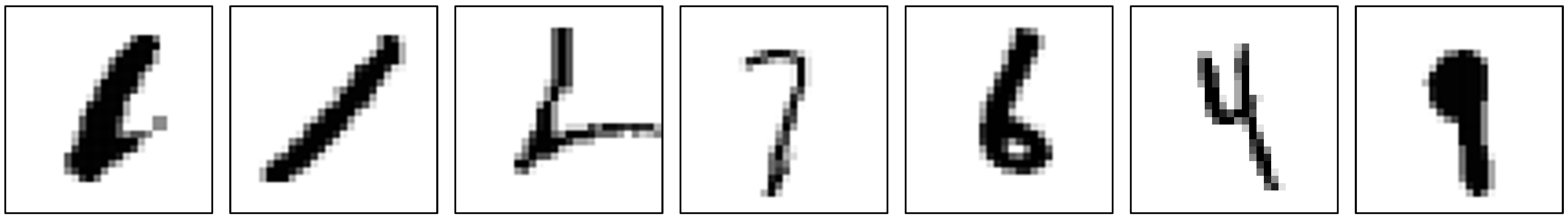
value	0	1	2	3	4	5	6	7	8	9	10	11	…	31	32	33	34	…	255
frequency	2	0	0	0	0	0	0	0	0	0	0	0	…	1	0	1	0	…	0

Digit Recognition cont.

- 6^{th} approach:
  - Local binary patterns
    → 4096-dimensional, sparse patterns

- accuracy:
  - training set: 98.6%
  - test set: 99.3%
- number of support vectors:
  264 (out of 1000)

	is 1	is not 1
classified as 1	495	2
classified as not 1	5	498



Digit Recognition cont.

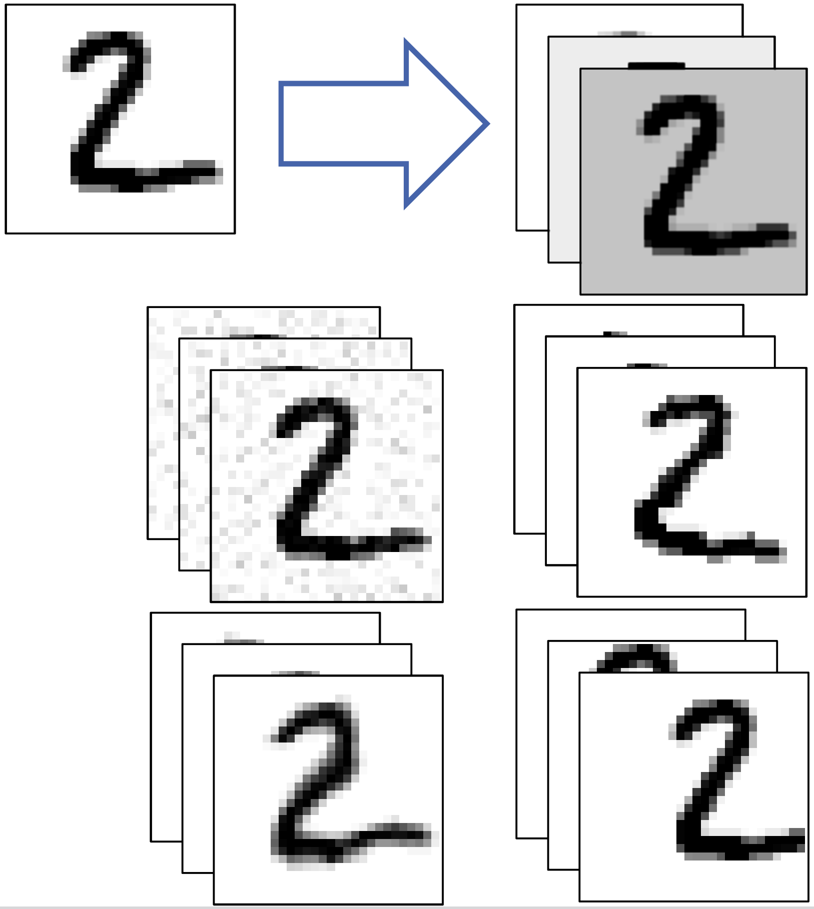
approach	1	2 (line fit)	3 (pixels)	4 (HOG)	5 (Haar)	6 (LBP)
features	2	3	432	288	98	4096
accuracy training set	93.1%	98.5%	99.0%	99.4%	99.1%	98.7%
accuracy test set	80.3%	99.7%	99.4%	99.7%	99.4%	99.3%
number of support vectors	167	95	219	174	109	264

- insights:
  - accuracy on training and test do not vary in the same way
  - “smart” features help a lot
  - more features do not mean higher accuracy
  - different approaches:
    - “smart” features including preprocessing
    - “generic” features (pixel values, HOG, Haar)

Data Tuning

Quality and quantity of training data have high impact on classification results

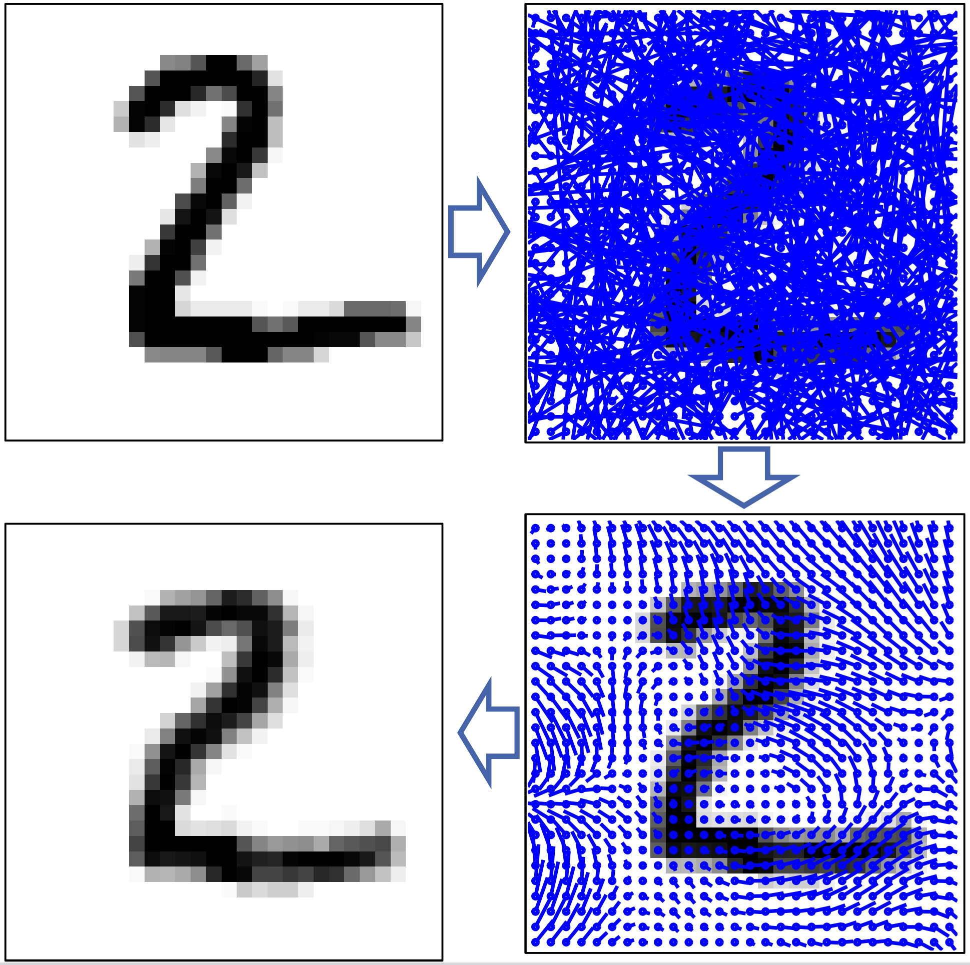
- How can we improve quantity?
  - select and label more images
  - search databases/internet for more training examples (ImageNet, KITTI, CalTech dataset, INRIA dataset, Microsoft COCO, …)
  - vary examples in brightness, contrast, position of object in ROI, rotation
  - add jitter (random noise)
  - mirror examples, if objects are symmetric
  - elastic distortions



Elastic Distortion

- Goal: get distorted example images to extend data set

Elastic distortion
1. for each pixel: sample random shift from a Gaussian distribution
2. smooth shift values by convolution with Gaussian filter
3. for each pixel: shift pixel to new position



Data Tuning cont.

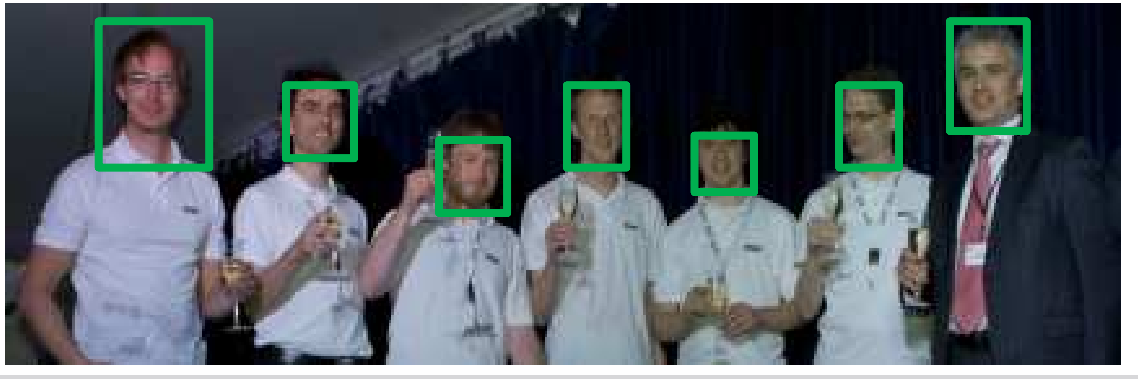
Quality and quantity of training data have high impact on classification results

$$x_i' = \frac{x_i - \bar{x}}{s_x}$$

$$with\ \bar{x} = \frac{1}{n}\sum_i x_i$$

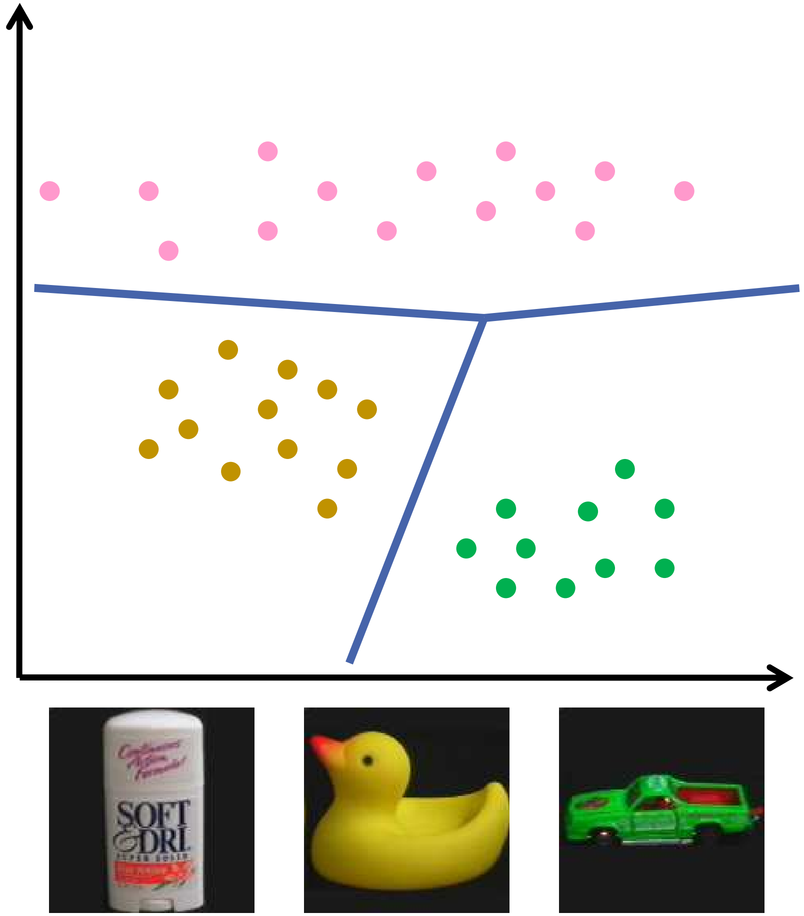
$$and\ s_x = \sqrt{\frac{1}{n}\sum_i (x_i - \bar{x})^2}$$

- How can we improve quality?
  - check consistency of labels
  - standardize/normalize patterns
  - take data from various sources/from various image sequences with varying conditions → increase variation within pattern set
  - check whether ROI is consistent



Multi-Class-Classification

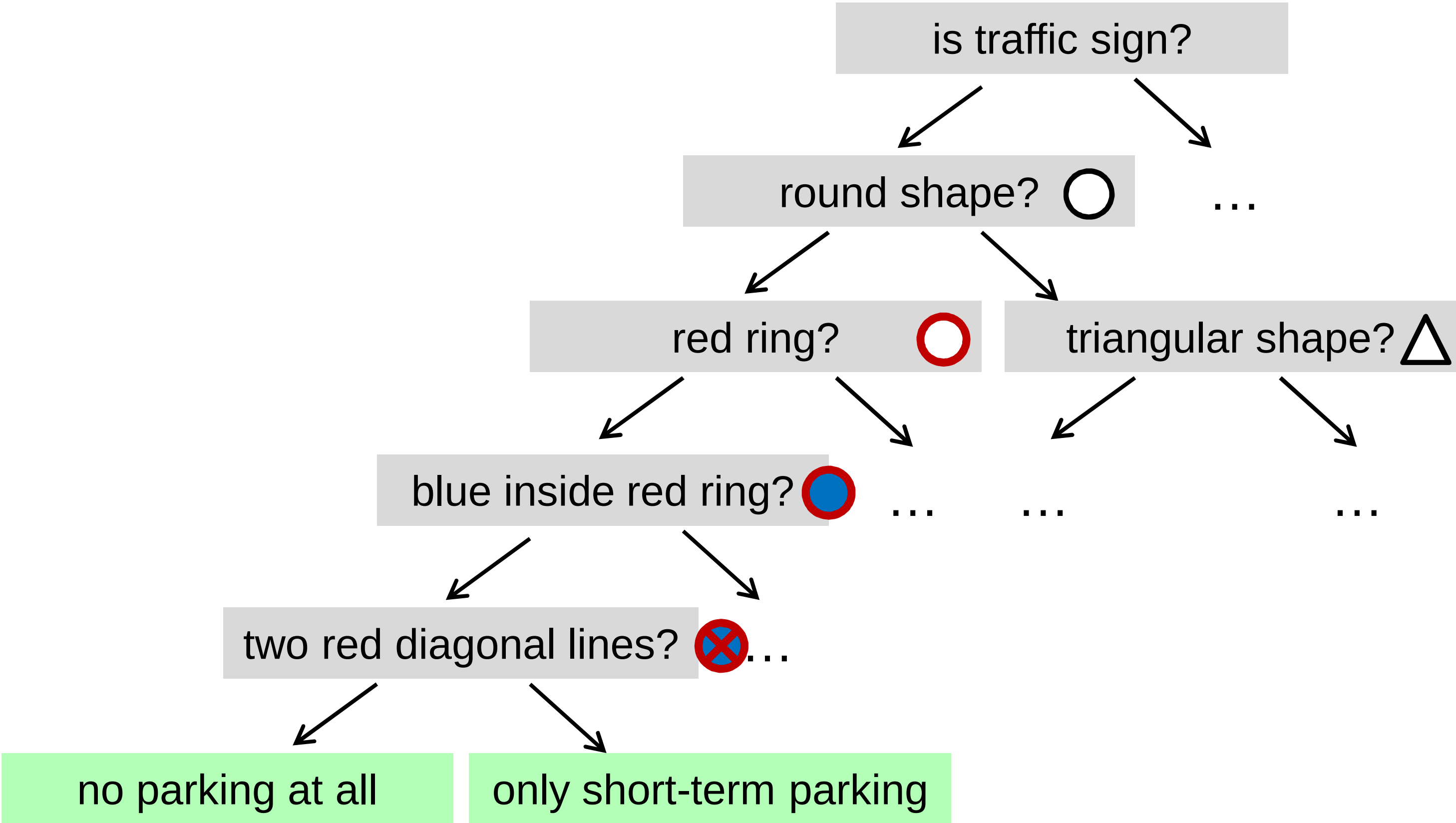
- classification with more than two classes



Multi-Class-Classification cont.

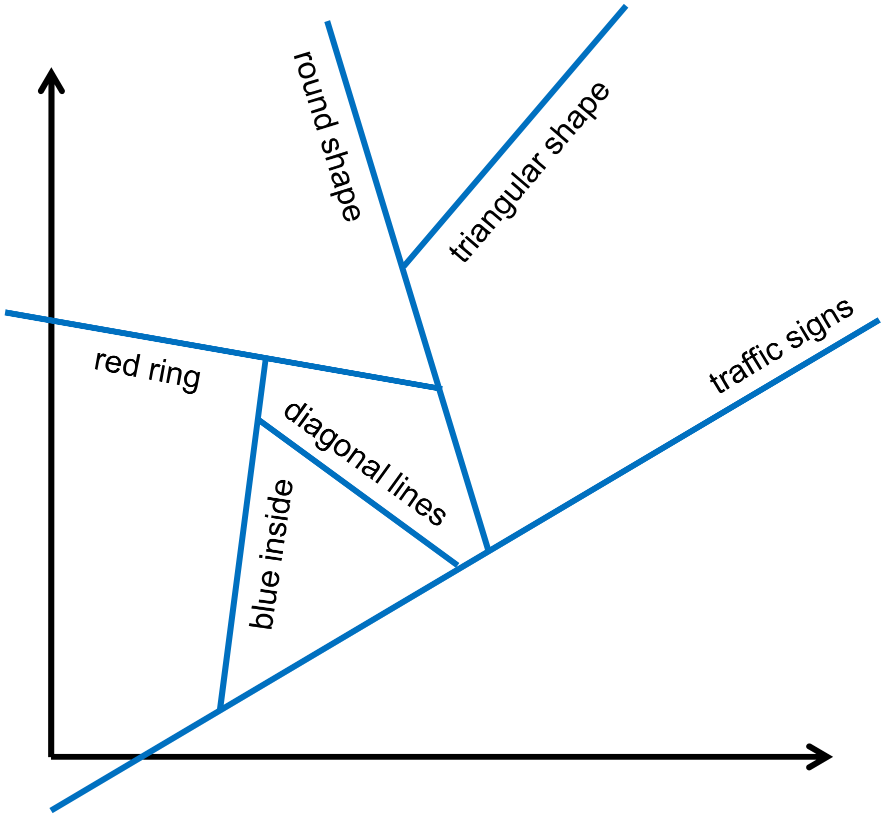
- hierarchical approach:

  build decision tree of binary classification



Multi-Class-Classification cont.

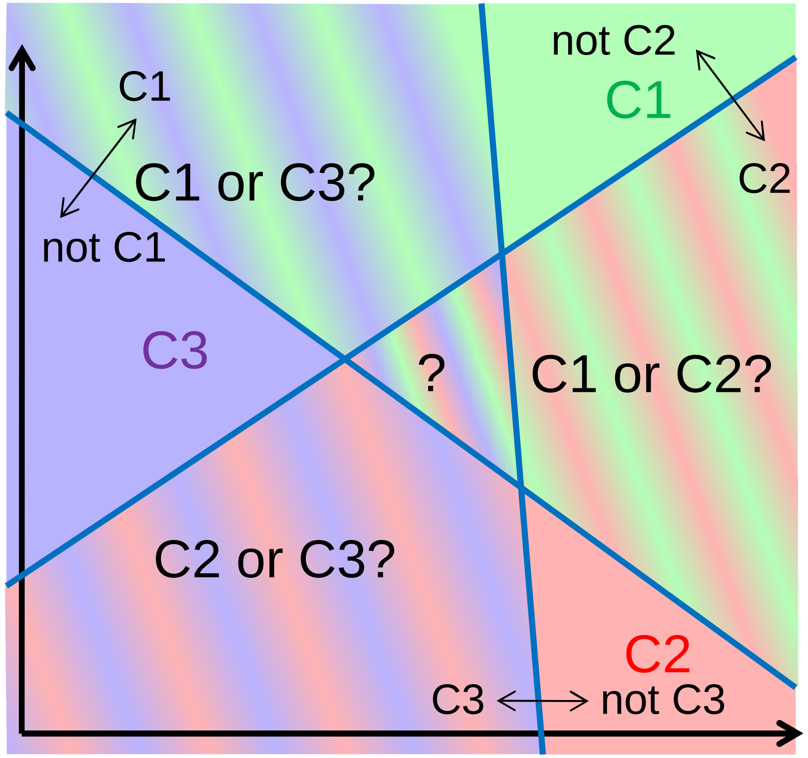
- disadvantage:
  - decision tree must be designed by hand
  - difficult to find suitable structure



Multi-Class-Classification cont.

- *one-versus-the-rest* approach:

  build one classifier for each class that classifies class elements versus non-class elements

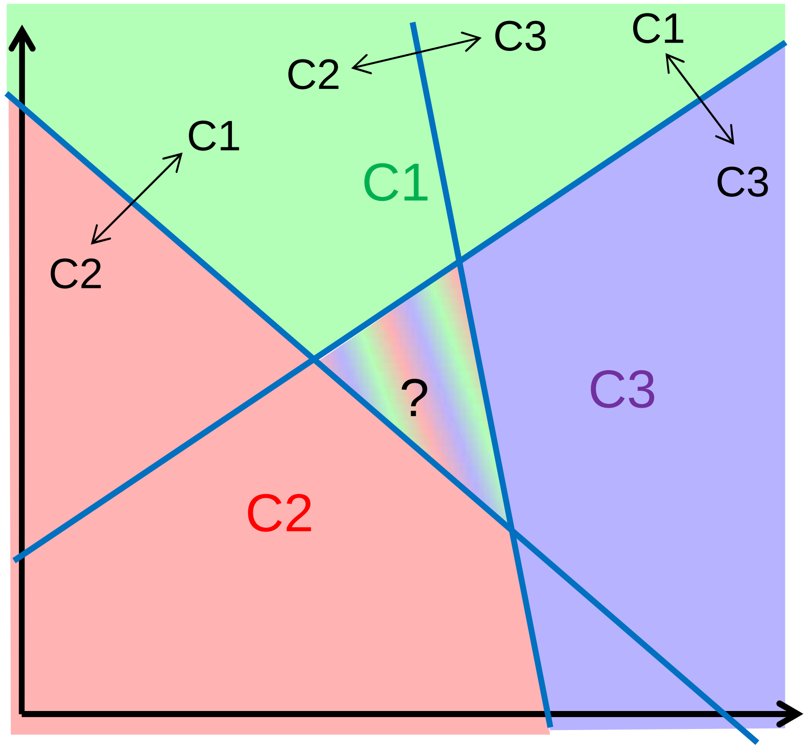


Multi-Class-Classification cont.

- *one-versus-one* approach:

  build one classifier for each pair of classes and use voting of classifiers

  – overcoming ambiguities



ENSEMBLE METHODS

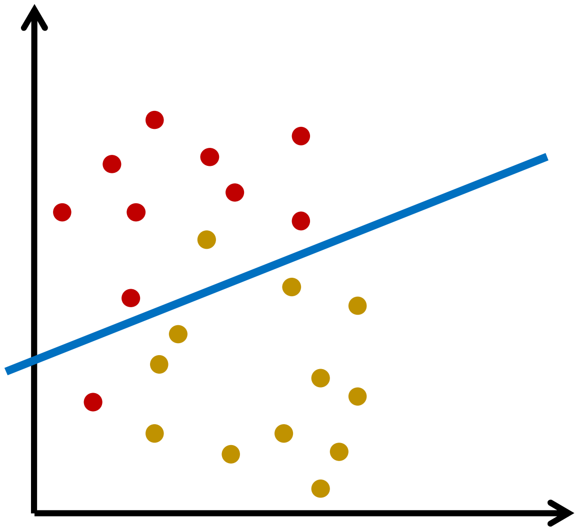
Ensembles

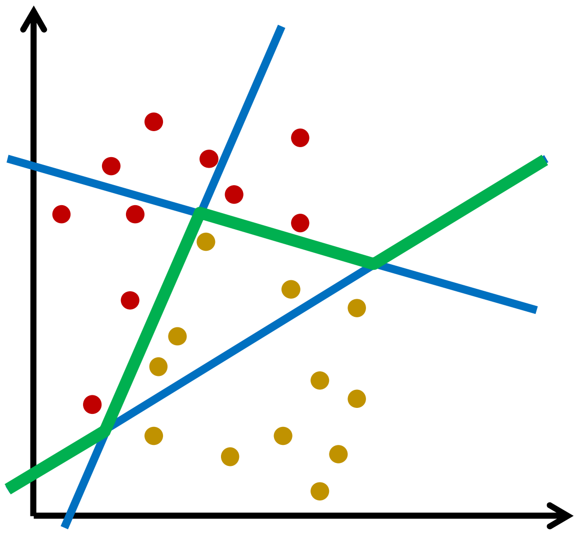
- what do politicians do when they don't know how to decide?
- ask an expert:
  - answer depends on expert's expertise
  - answer depends on expert's point of view

- ask many experts
  - quality: experts must have big expertise
  - fairness & independence: experts must not be selected depending on their point of view
  - decision of committees of experts might be better than every individual expert

Ensembles

- What do you do if you want to solve a classification problem?
  - *create an expert: train a classifier*
  - *train several classifiers*
    → and build an ensemble





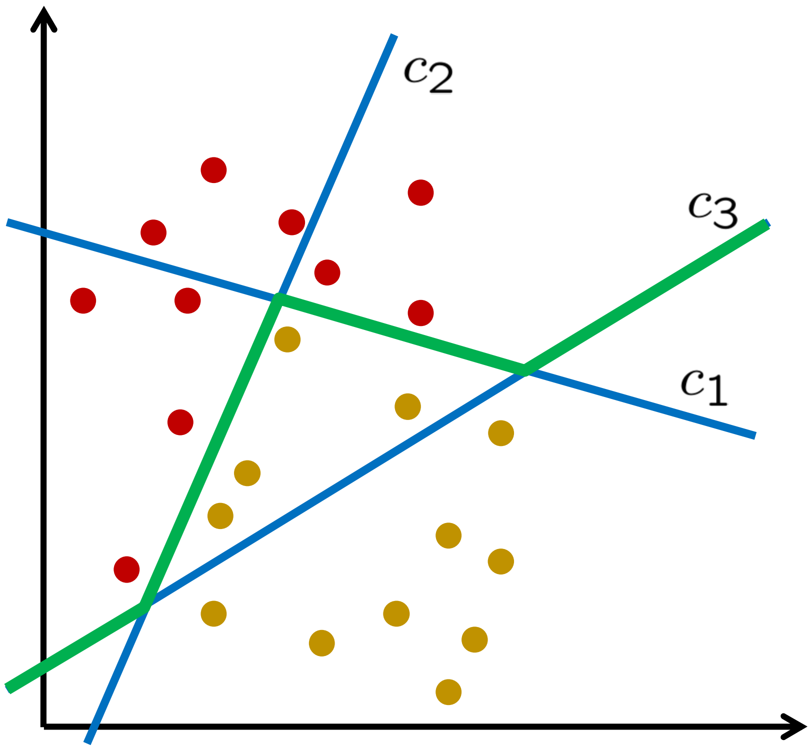
Ensembles cont.

- How do ensembles work?
  - *k* classifiers $c_1, c_2, \ldots, c_k$
  - applying same pattern to all classifiers → *k* predictions

$$c_1(\vec{x}) \in \{-1, +1\},$$
$$c_2(\vec{x}) \in \{-1, +1\},$$
$$\ldots,$$
$$c_k(\vec{x}) \in \{-1, +1\}$$

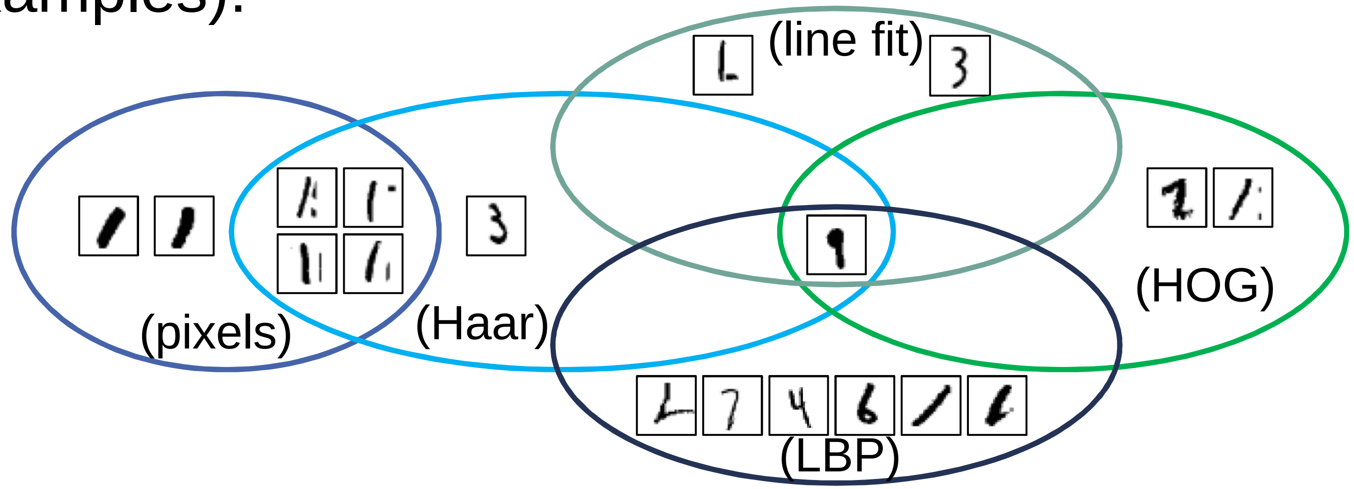
  - sum up all predictions and compare with zero:

$$ensemble(\vec{x}) = sign\Big(\sum_{j=1}^{k} c_i(\vec{x})\Big)$$



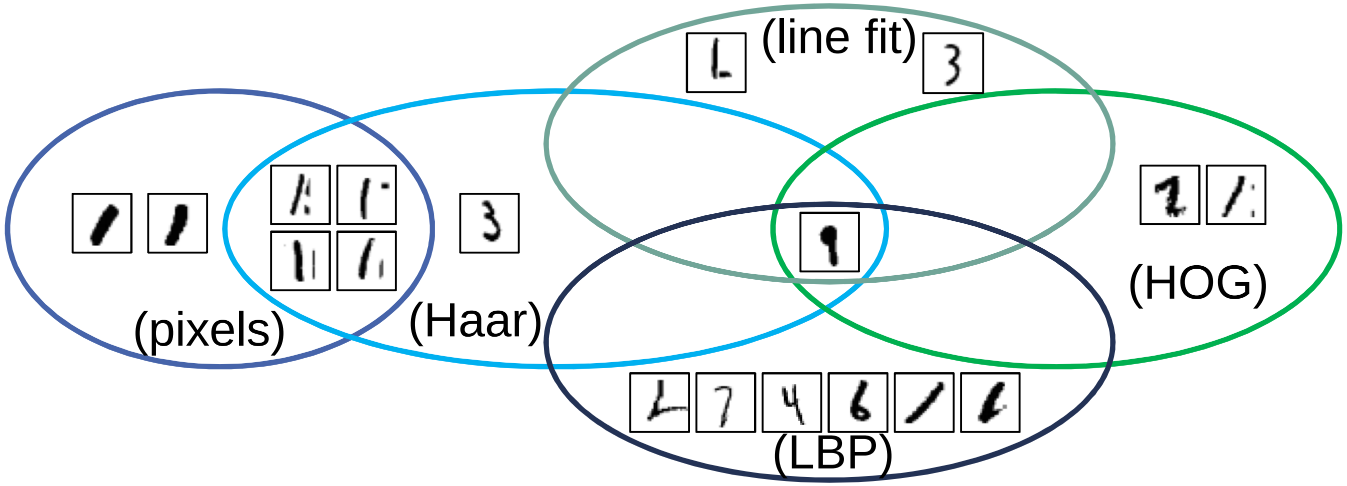
Example: Digit Recognition

- Best four approaches of digit recognition (cf. slide 62)
  - (2) „line fit“ feature: 99.7%
  - (3) pixel values: 99.4%
  - (4) HOG features: 99.7%
  - (5) Haar features: 99.4%
  - (6) LBP features: 99.3%
- Ensemble of these approaches, shared errors (out of 1000 test examples):



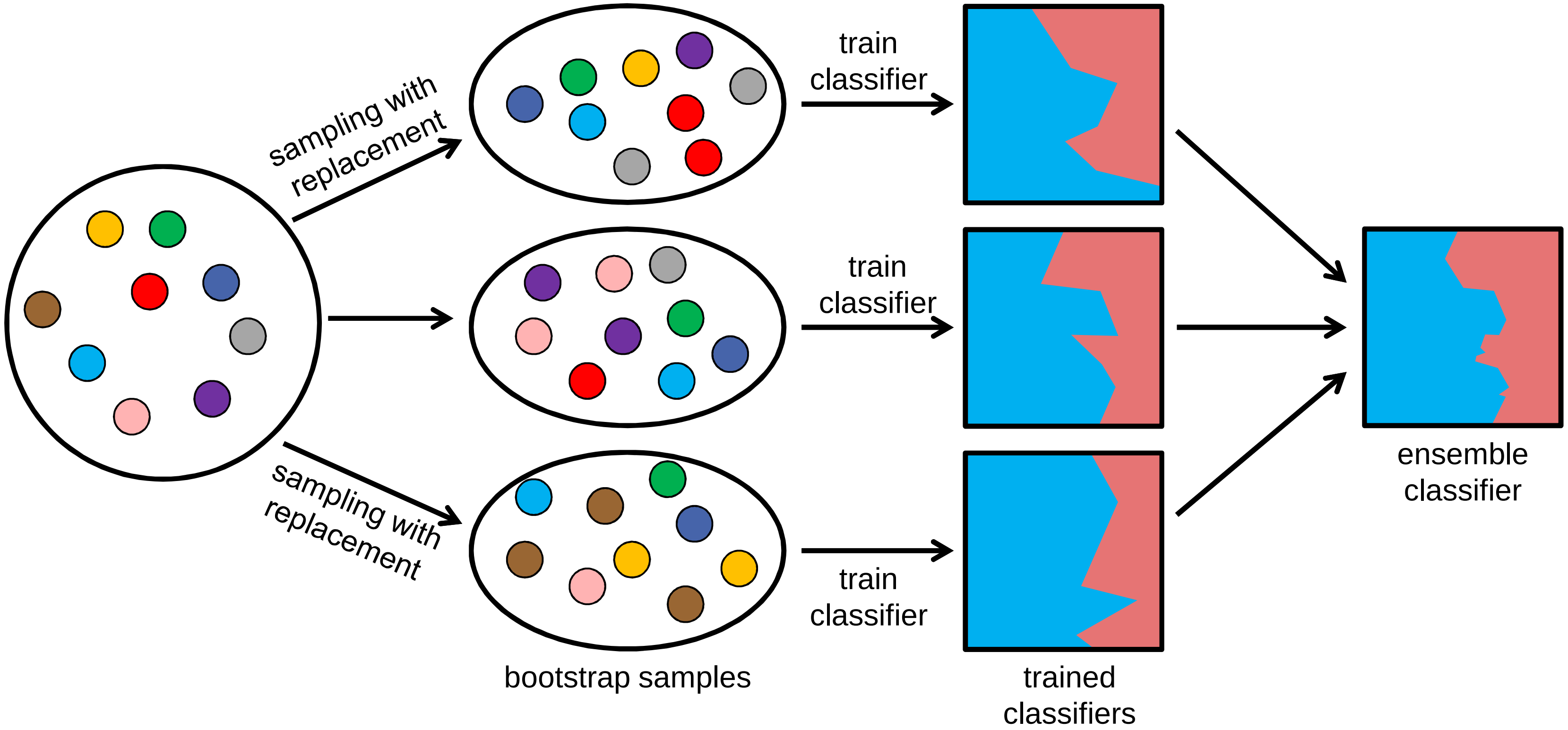
Example: Digit Recognition cont.

- Ensemble of these approaches:
  - *ensemble: line fit, pixels, Haar*
    - error rate: 5/1000
    - error of members: 3, 6, 6/1000
  - *ensemble: line fit, HOG, Haar*
    - error rate: 1/1000
    - error of members: 3, 3, 6/1000
    - error of SVM with joint features: 2/1000
  - *ensemble: line fit, HOG, LBP*
    - error rate 1/1000
    - error of members: 3, 3, 7/1000
    - error of SVM with joint features: 5/1000
  - *ensemle: all together*
    - error rate 1/1000



Bagging

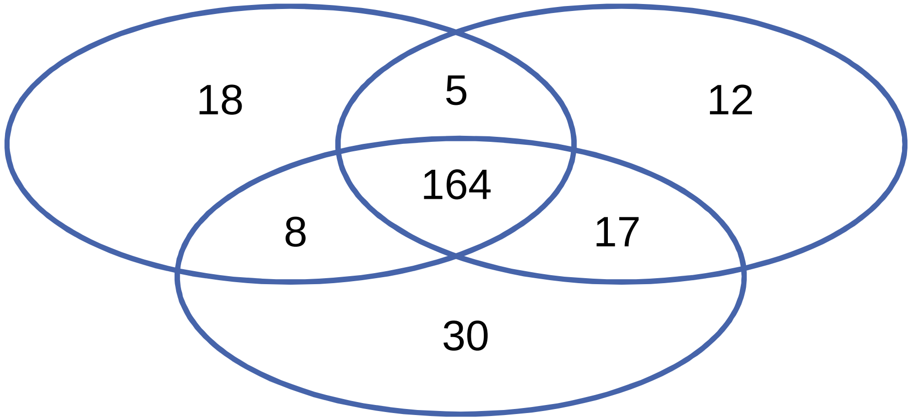
- Bagging (Breimann, 1996) = bootstrap aggregation = learning from bootstrap samples (Efron, 1982)



- varying the training set by sampling with replacement

Bagging cont.

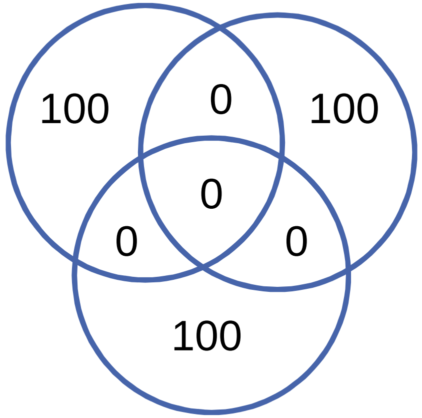
- Example: Bagging for digit recognition
  - using 2 features modeling (approach 1, cf. slide 41)
  - bagging with 3 bootstrap samples
  - shared test errors:



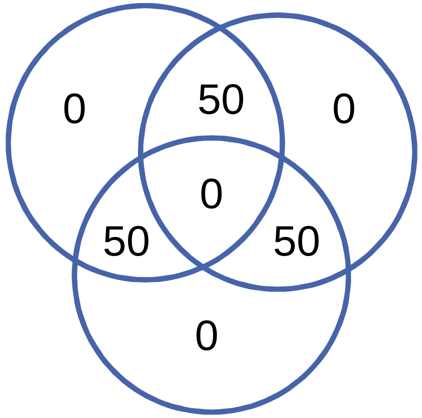
  - total test error:
    - non-ensemble approach: 197/1000
    - ensemble approach with 3 bootstrap samples: 194/1000

Boosting

- when is an ensemble beneficial?
  - *best case: classifiers don't share errors*
    - ensemble errors: 0
    - errors of each classifier: 100



  - *worst case: classifiers share all errors*
    - ensemble errors: 150
    - errors of each classifier: 100



- key idea to improve ensembles:
  - avoid classifiers sharing errors
- Boosting:
  - train classifiers depending on each other
  - classifier No. *n+1* should focus on the examples misclassified by classifiers No. *1,…,n*

Boosting cont.

- implementation ideas:
  - *weighted training patterns*

    introduce weight $\gamma_i \geq 0$ for each *training pattern* to model its importance

    → modification of training algorithms necessary, e.g. soft margin SVM:

$$\underset{\vec{w},b}{minimise} \quad \frac{1}{2}||\vec{w}||^2 + C\sum_i (\gamma_i \cdot \xi_i)$$

$$subject\ to \quad d^{(i)} \cdot (\langle \vec{x}^{(i)}, \vec{w} \rangle + b) \geq 1 - \xi_i \qquad \text{for all } i$$

$$\xi_i \geq 0 \qquad \text{for all } i$$

  - *how to determine pattern weights?*

    recalculate weights after training a classifier:

    - increase weight of misclassified patterns
    - decrease weight of well classified patterns

Boosting cont.

- implementation ideas:
  - *weighted voting*

    introduce weight $\beta_k \geq 0$ for each *classifier* to model its reliability

    $\rightarrow$ modification of voting scheme:

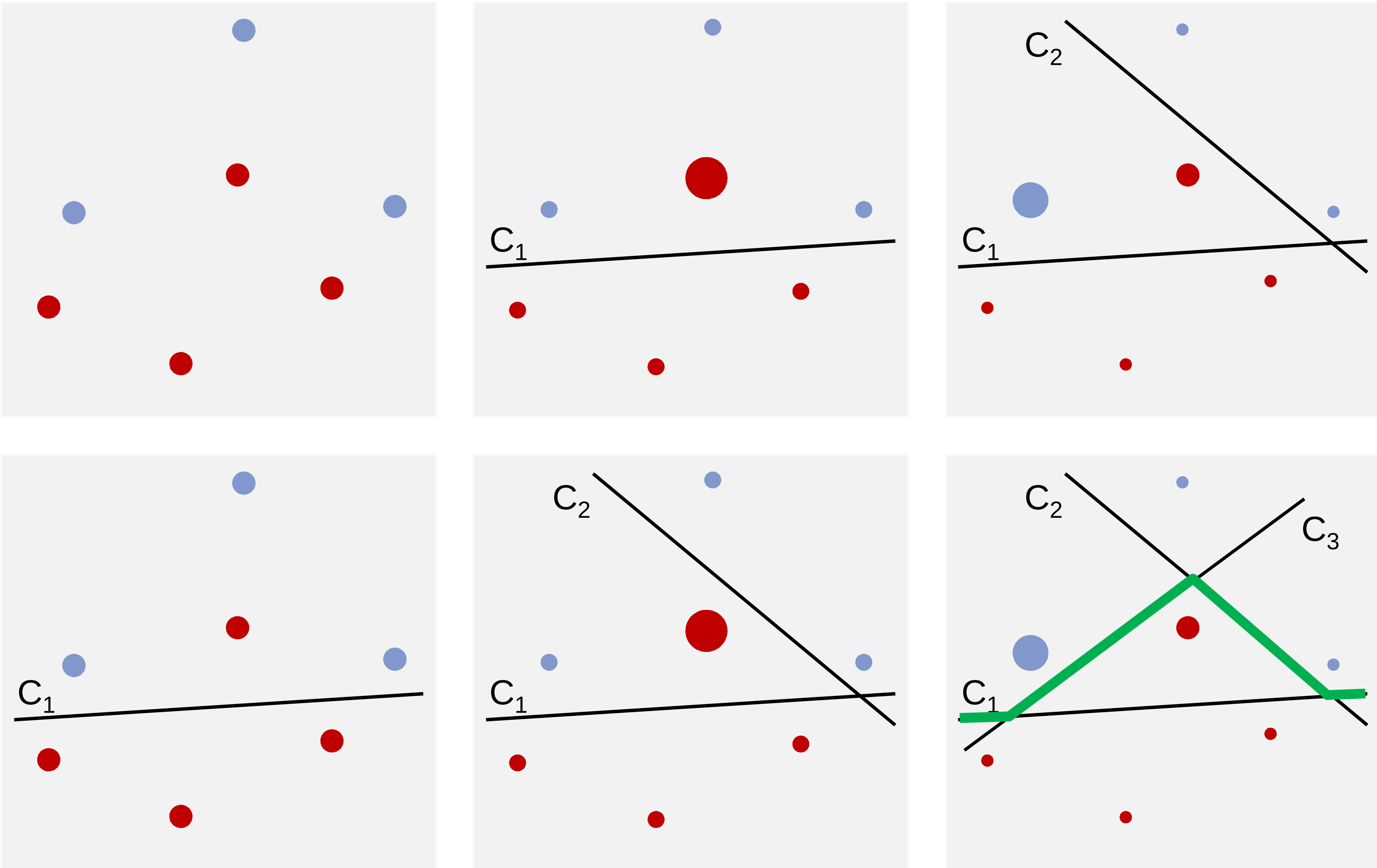
$$ensemble(\vec{x}) = sign\left(\sum_k \beta_k \cdot vote_k\right)$$

  - *how to determine the voting weights?*

    choose weight according to performance of classifier:

    - large weight for classifier with high accuracy
    - small weight for classifier with low accuracy

Boosting cont.



AdaBoost [Freund & Schapire, 1994]

initialize pattern weights $\gamma_i \leftarrow \frac{1}{n} \quad i \in \{1, \dots, n\}$

for $k = 1, \dots, T$ **do**

  train new classifier c_k from weighted training patterns

  calculate weighted training error

  $\epsilon_k := \sum_{i=1}^{n} \begin{cases} 0 & \text{if } c_k(\vec{x}^{(i)}) = d^{(i)} \\ \gamma_i & \text{otherwise} \end{cases}$

  set $\beta_k \leftarrow \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k}$

  for $i = 1, \dots, n$ **do**

    $\gamma_i \leftarrow \gamma_i \cdot \begin{cases} e^{-\beta_k} & \text{if } c_k(\vec{x}^{(i)}) = d^{(i)} \\ e^{\beta_k} & \text{if } c_k(\vec{x}^{(i)}) \neq d^{(i)} \end{cases}$

  end for

  renormalize weights $\gamma_1, \dots, \gamma_n$

end for

create ensemble $\vec{x} \mapsto \sum_{k=1}^{T} \frac{\beta_k}{\sum_{j=1}^{T} \beta_j} c_k(\vec{x})$

AdaBoost cont.

- properties of AdaBoost:
  - the training error of the ensemble is bounded above by:

$$\prod_{t=1}^{T}\left(2\sqrt{\epsilon_t(1-\epsilon_t)}\right) \leq \exp\left\{-2\sum_{t=1}^{T}(\frac{1}{2}-\epsilon_t)^2\right\}$$

  - if all $\epsilon_t \leq \lambda < \frac{1}{2}$ and $T \rightarrow \infty$ AdaBoost yields a perfect classifier

Haar Classifier

- Haar feature:

$$s = \frac{1}{N_{red}} \sum_{(u,v)\in red\ area} g(u,v) - \frac{1}{N_{blue}} \sum_{(u,v)\in blue\ area} g(u,v)$$

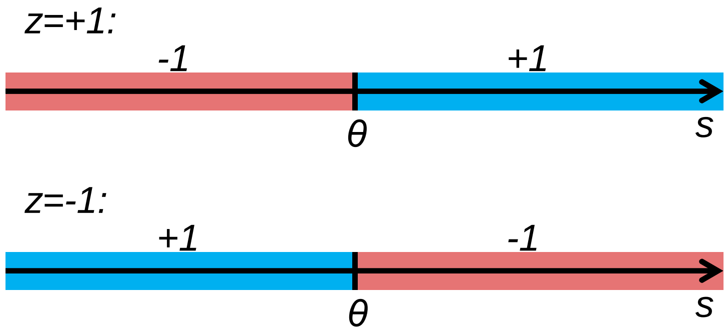
- make it a classifier:

$$c(s) = sign(z \cdot (s - \theta))$$

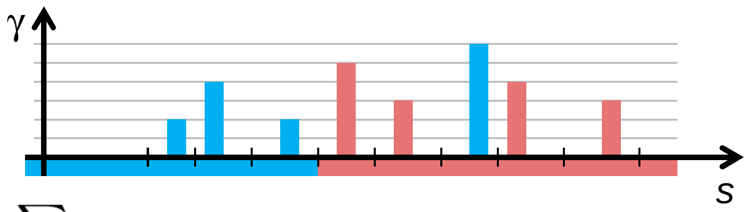
  - parameters:

    $\theta \in \mathbb{R}$ threshold

    $z \in \{+1, -1\}$ orientation

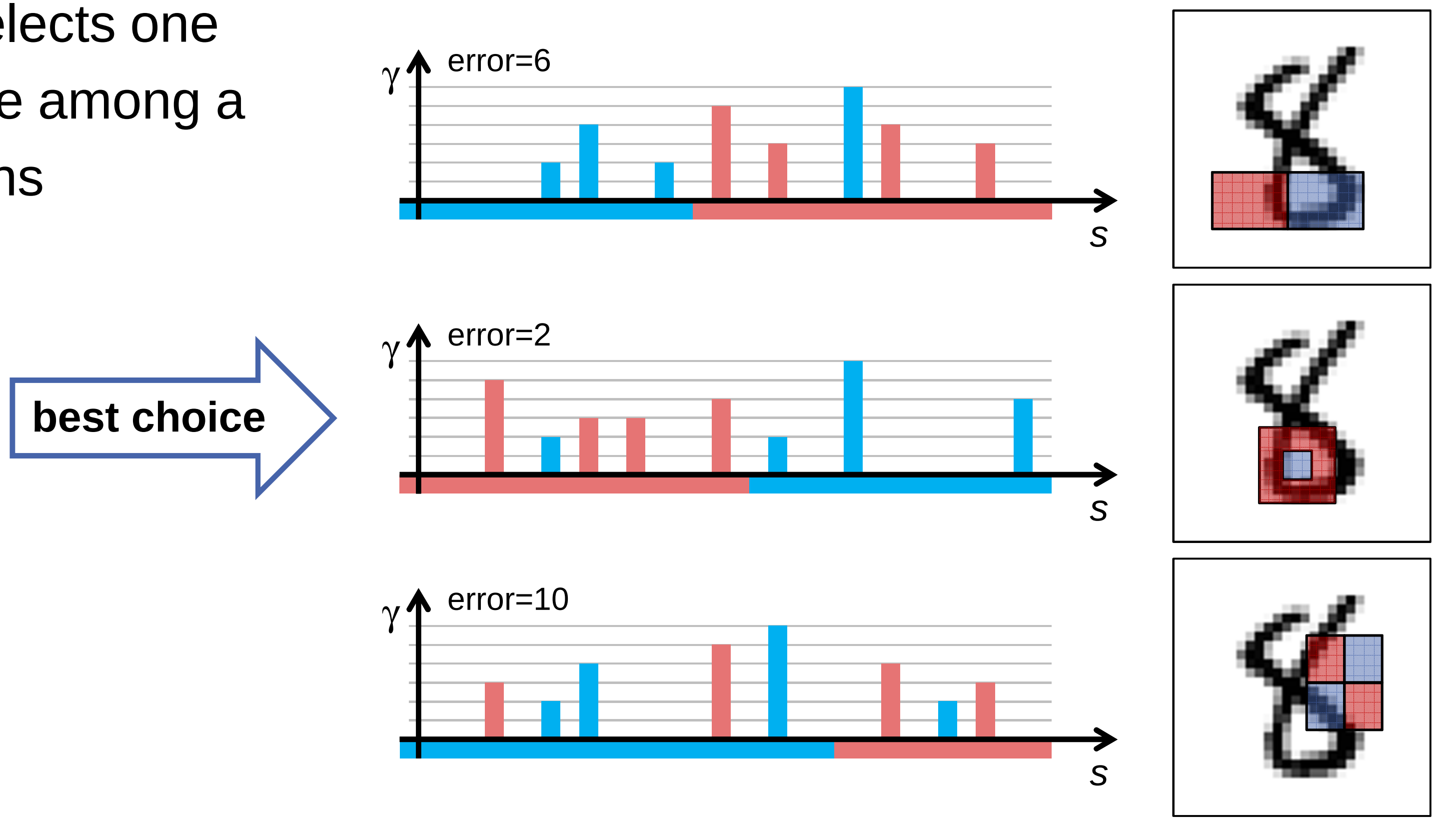


- train the classifier from weighted examples:
  - try all possible values for θ and z
  - select values that minimize weighted error $\sum_{s_i<\theta, d^{(i)}=z} \gamma_i + \sum_{s_i>\theta, d^{(i)}=-z} \gamma_i$



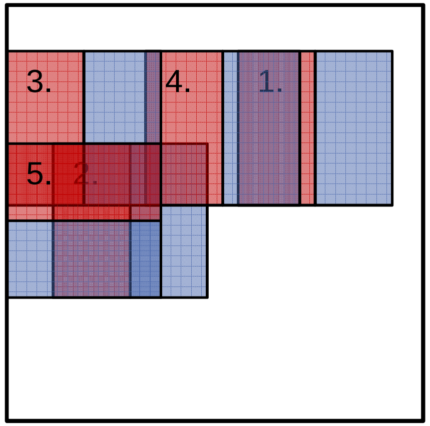
Haar Classifier cont.

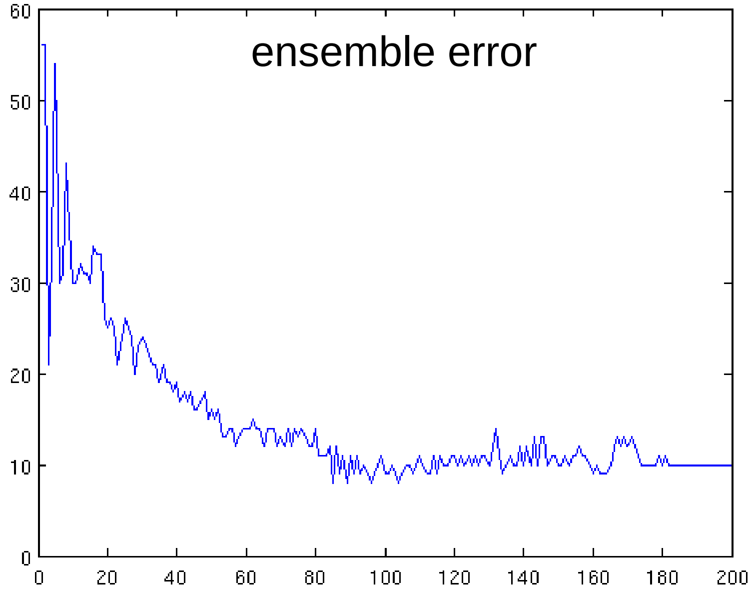
- Haar Classifier with multiple features:
  - classifier selects one Haar feature among a set of options

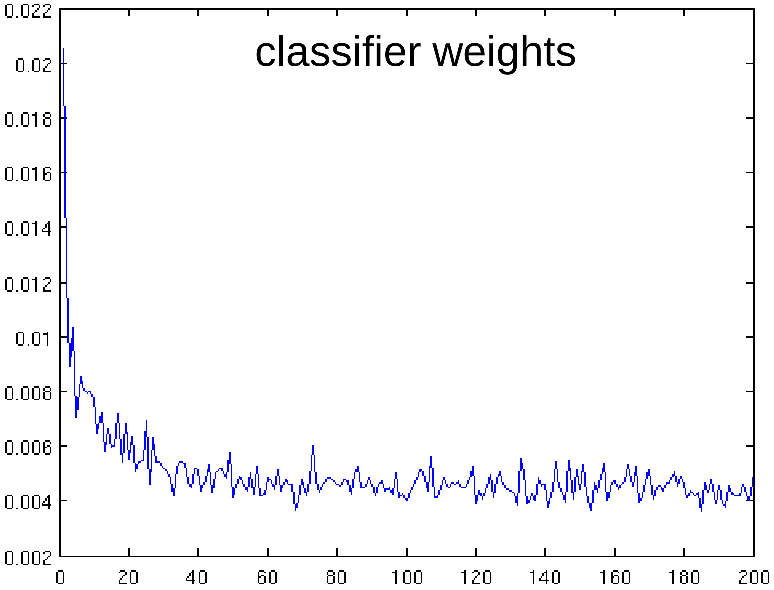


Boosting with Haar Classifiers

- Idea: use AdaBoost with Haar Classifiers
  - test error on digit recognition task:
    - first classifier: 56/1000
    - ensemble size 5: 54/1000
    - ensemble size 50: 16/1000
    - ensemble size 200: 10/1000

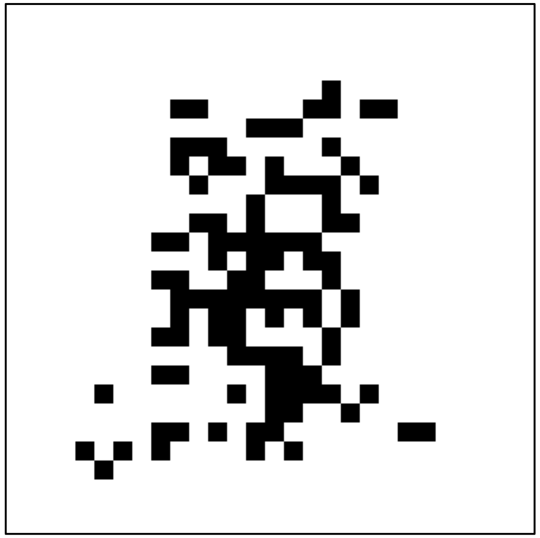


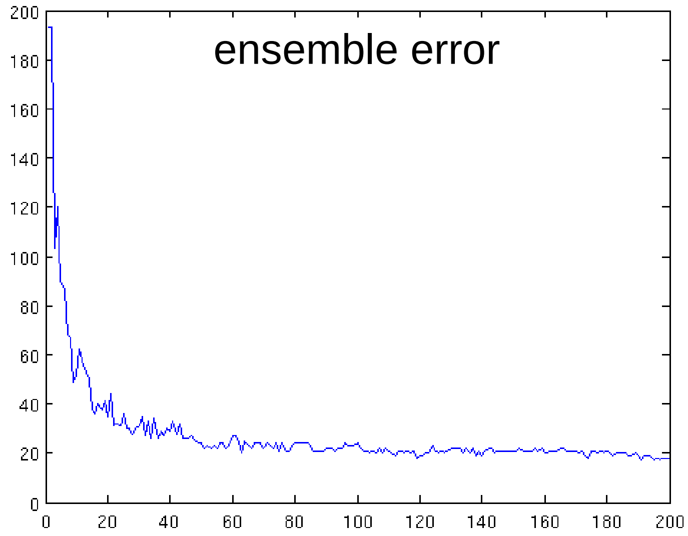


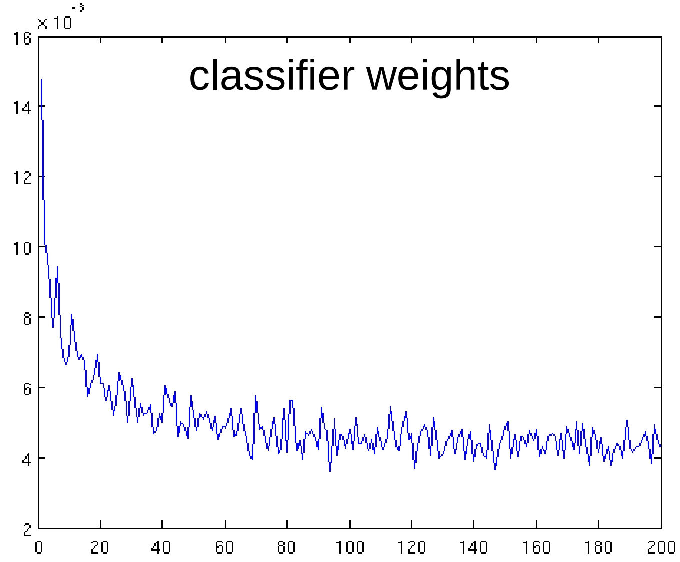


Boosting Other Classifiers

- Every feature induces a classifier
  - e.g. pixel gray levels
  - test error on digit recognition task:
    - first classifier: 193/1000
    - ensemble size 5: 90/1000
    - ensemble size 50: 24/1000
    - ensemble size 200: 18/1000







Balancing Errors

- Ensemble classifies:

$$\sum_k (\beta_k \cdot c_k(\vec{x})) \lesseqgtr 0$$

- Extension:

$$\sum_k (\beta_k \cdot c_k(\vec{x})) \lesseqgtr z$$

  - $z > 0$: classify positive only if you are very sure
  - $z < 0$: classify positive even if you are not that sure

Balancing Errors cont.

- Example
  - AdaBoost with Haar features
  - ensemble size 5 (c.f. slide 81)

$$\sum_k (\beta_k \cdot c_k(\vec{x})) \lessgtr z$$

z=0	is 1	is not 1
classified as 1	474	28
classified as not 1	26	472

high recall, small precision →

z=-0.5	is 1	is not 1
classified as 1	499	129
classified as not 1	1	371

high precision, small recall →

z=0.5	is 1	is not 1
classified as 1	387	5
classified as not 1	113	495

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

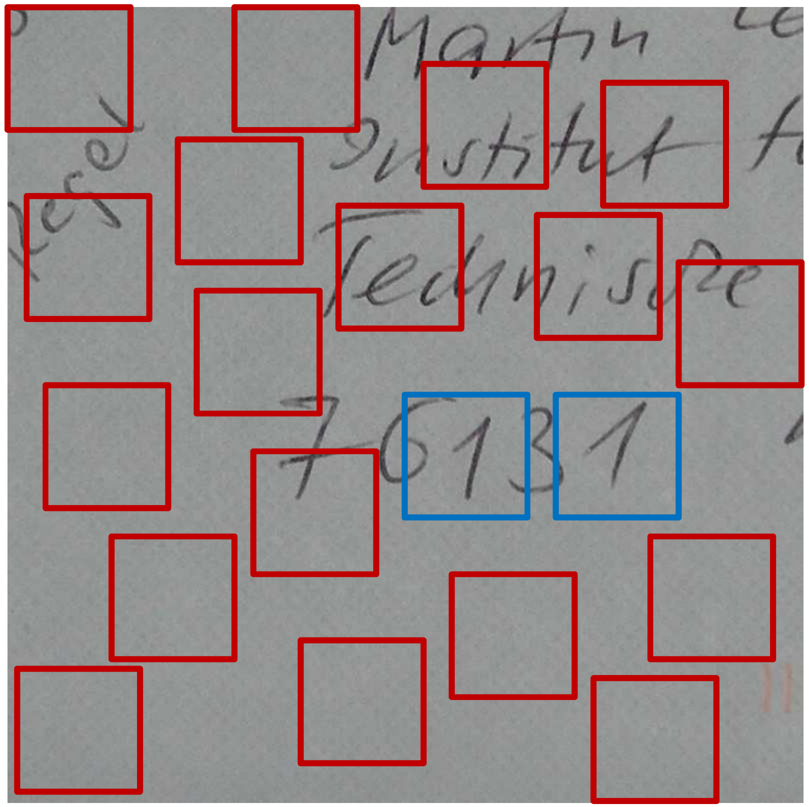
$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Searching for Objects

- how can we find objects in an image with a classifier?
  - e.g. finding the digit “1” on a letter
  - using a classifier for “1”
- idea:
  - apply classifier to all possible areas in the image
    - vary position of area
    - vary size of area
    - vary orientation of area (optionally)

  → requires millions of trials

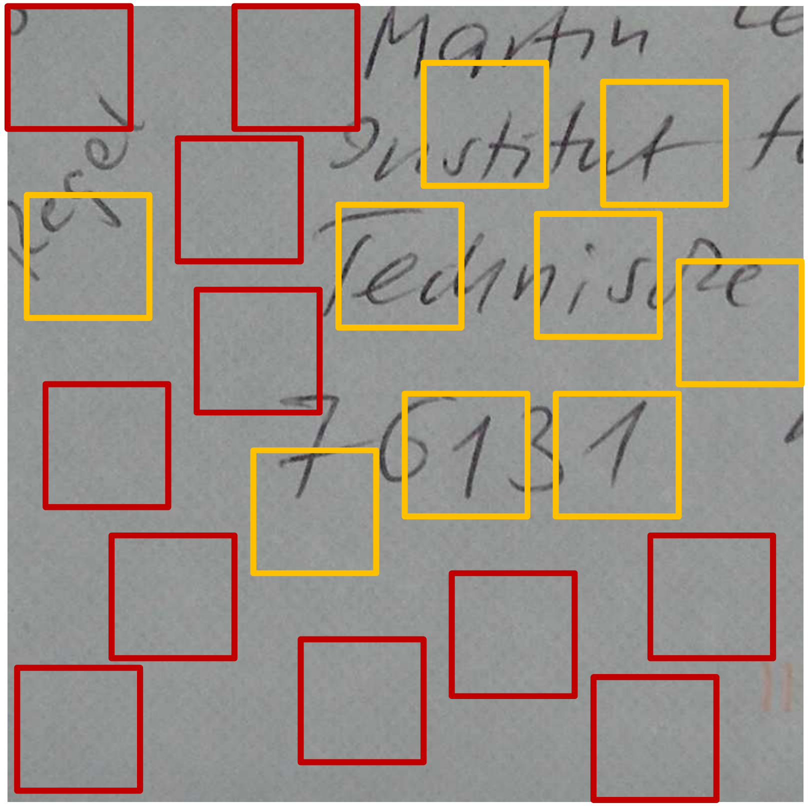
  efficiency?



Searching for Objects cont.

- improved idea:

  use two classifiers
  - classifier 1
    - efficient
    - inaccurate
    - high recall
    - low precision
    - applied to all areas
  - classifier 2
    - inefficient
    - accurate
    - high recall
    - high precision
    - applied to areas which are found by classifier 1

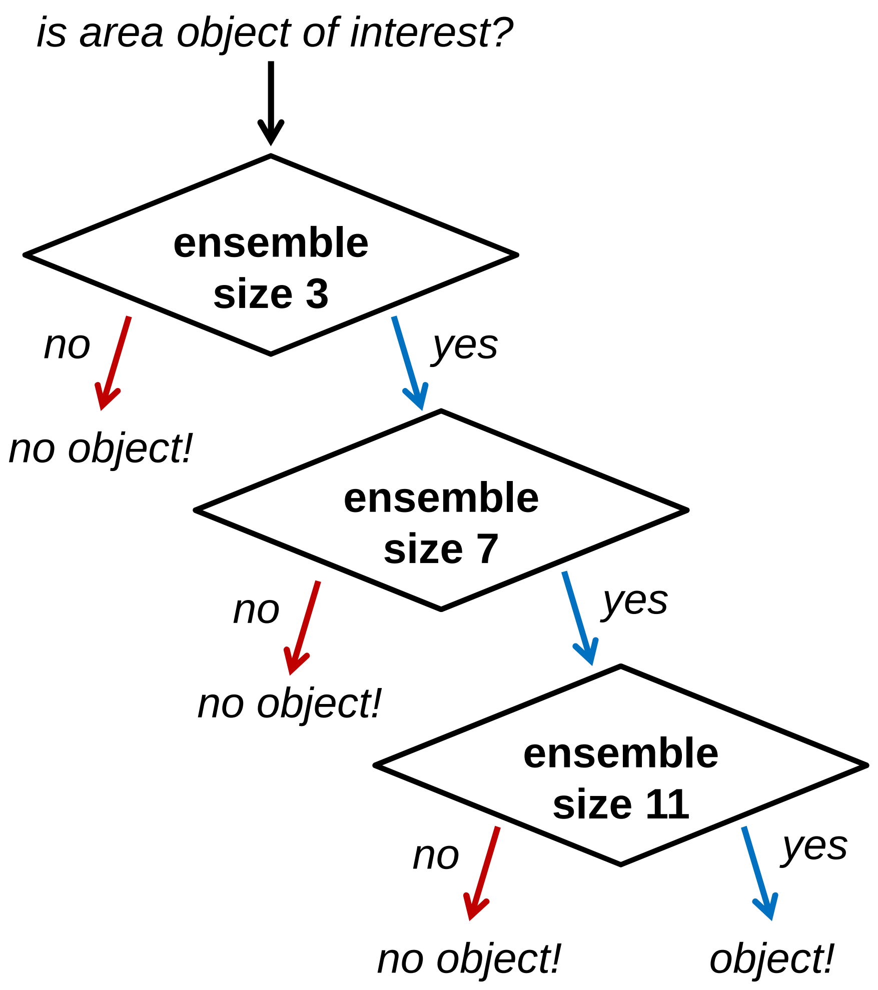


- idea can be extended to a series of many classifiers

  → approach of Viola/Jones

Viola/Jones Approach

- combines:
  - Haar classifiers
  - AdaBoost
  - series ("cascade") of classifiers of increasing ensemble size
  - tuning ensembles to maximize recall
  - search over the whole image with varying area position and size



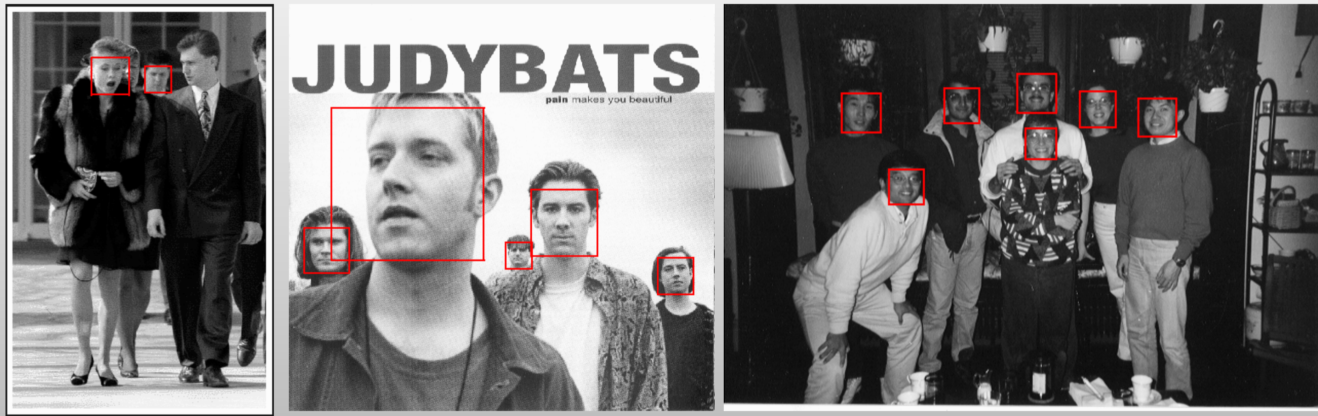
Example by Viola/Jones

- face recognition

most important Haar features

positive training examples

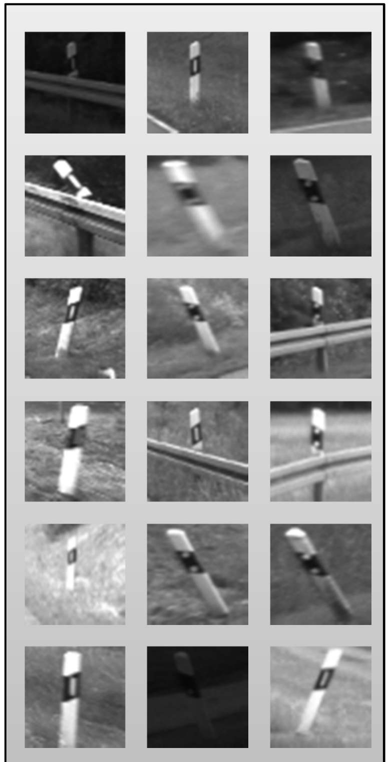
results



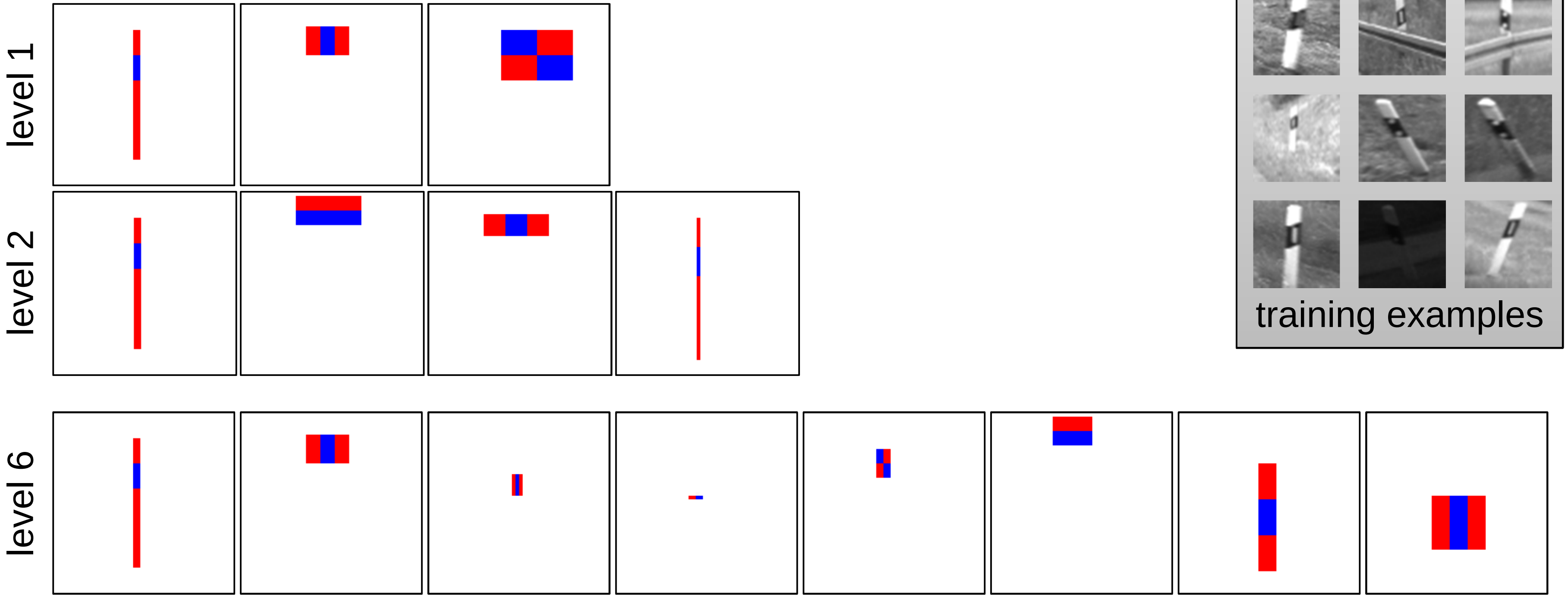
images source: P. Viola, M. Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, CVPR 2001

Example: Detection of Delineators

- detection of delineators as visual landmarks
  - using Viola/Jones approach to find regions of interest
  - ensemble sizes: 3, 4, 3, 5, 9, 8, 11, 14, 20, 23
  - classification of regions of interest with SVM and HOG features



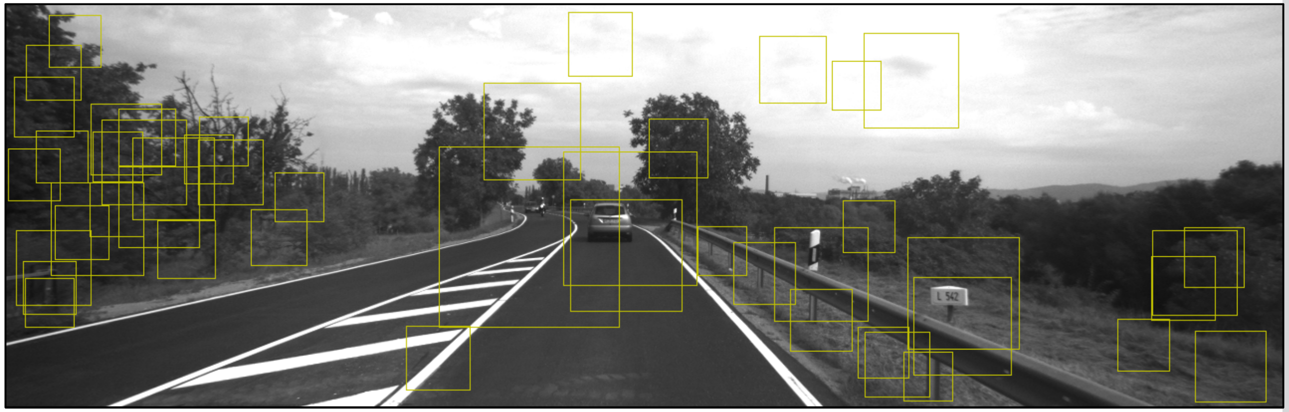
training examples



Example: Detection of Delineators cont.

depth of
cascade = 4

average:
27 roi/frame



depth of
cascade = 7

average:
10 roi/frame

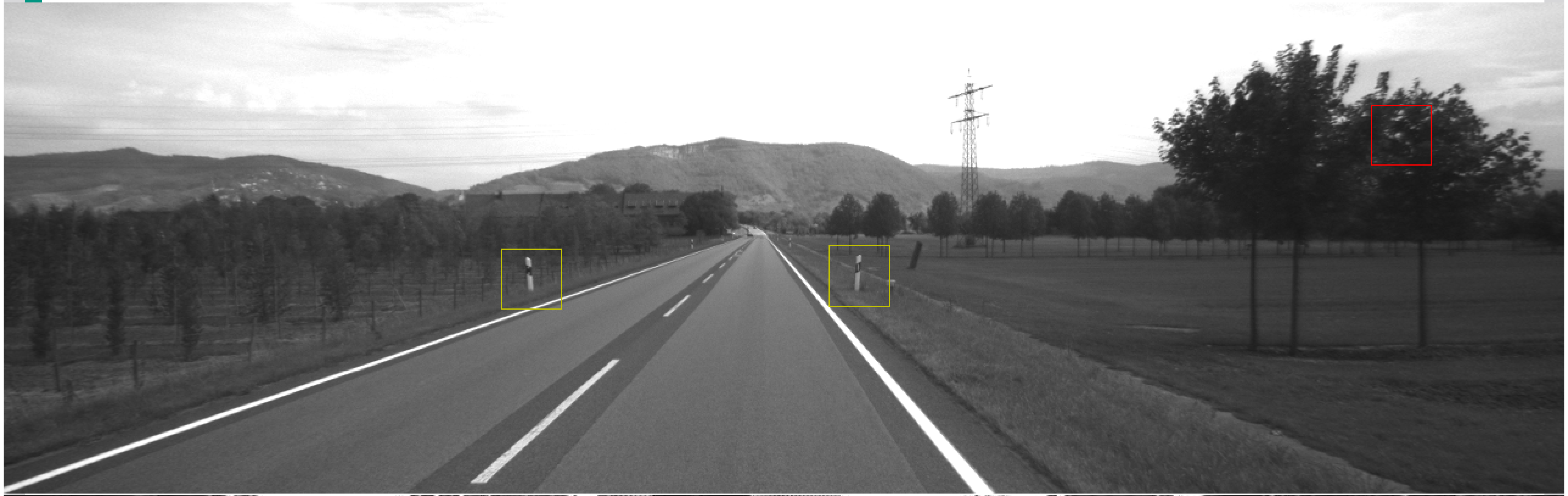
depth of
cascade = 10

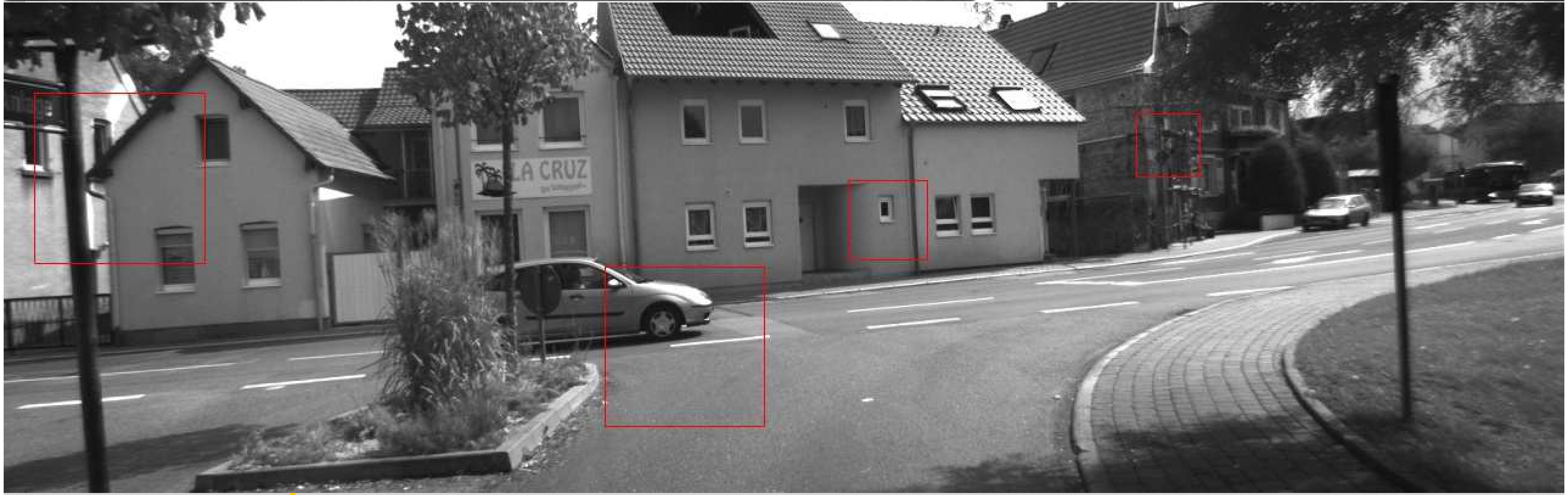
average:
1.6 roi/frame

Example: Detection of Delineators cont.

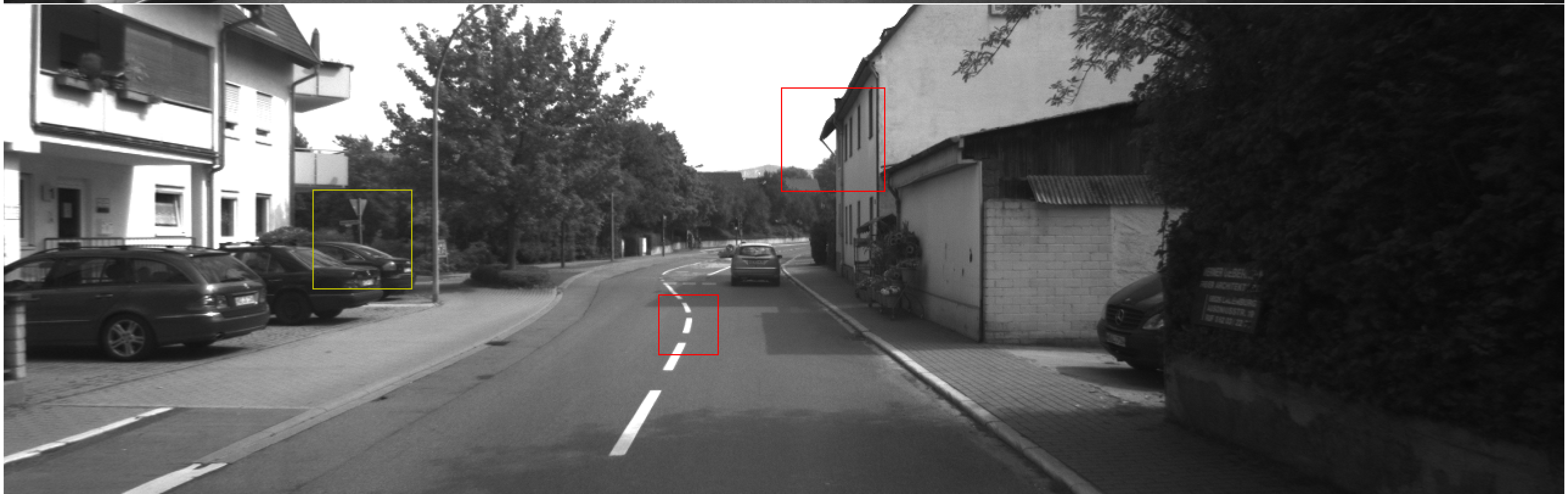
- performance on 1283x403 images
  - runtime approx. 55 ms/frame for Haar-classifier/detector
  - 0-10 roi per image found
  - SVM/HOG requires ≈ 1 ms/roi
  - test set accuracy of SVM >99%

Example: Detection of Delineators cont.





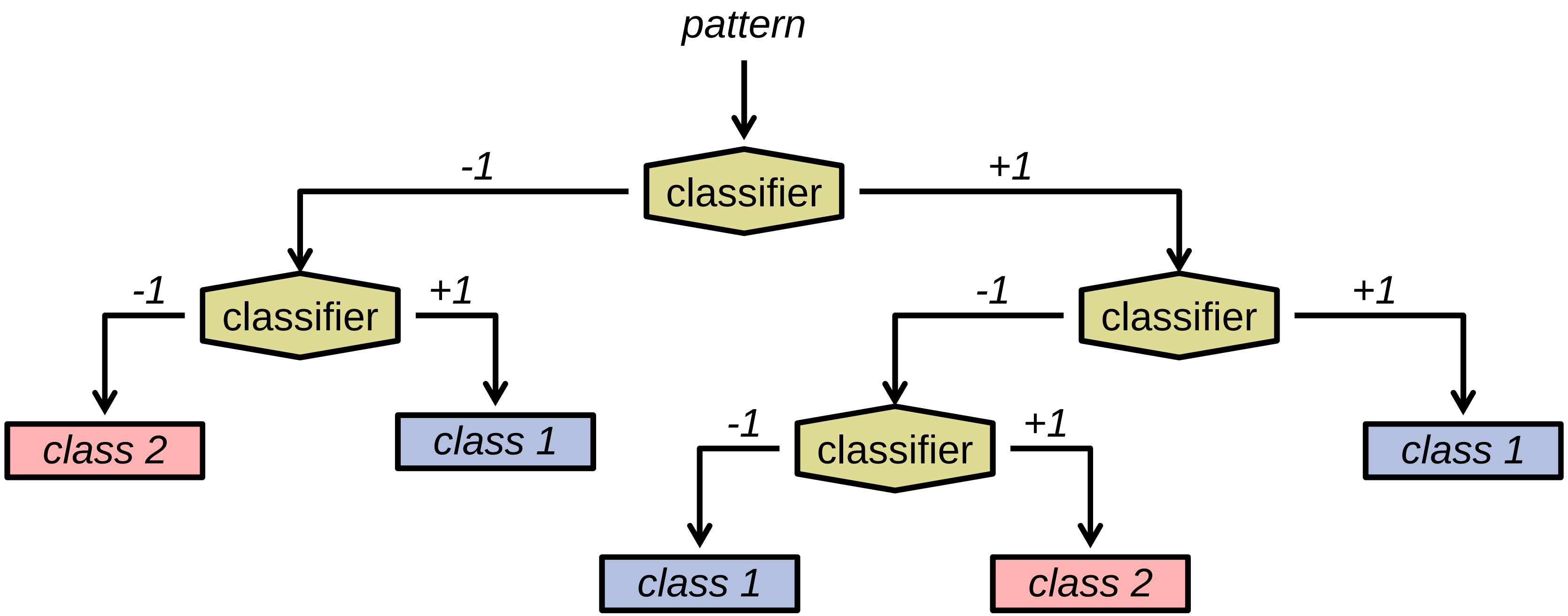
Example: Detection of Delineators cont.



DECISION TREES

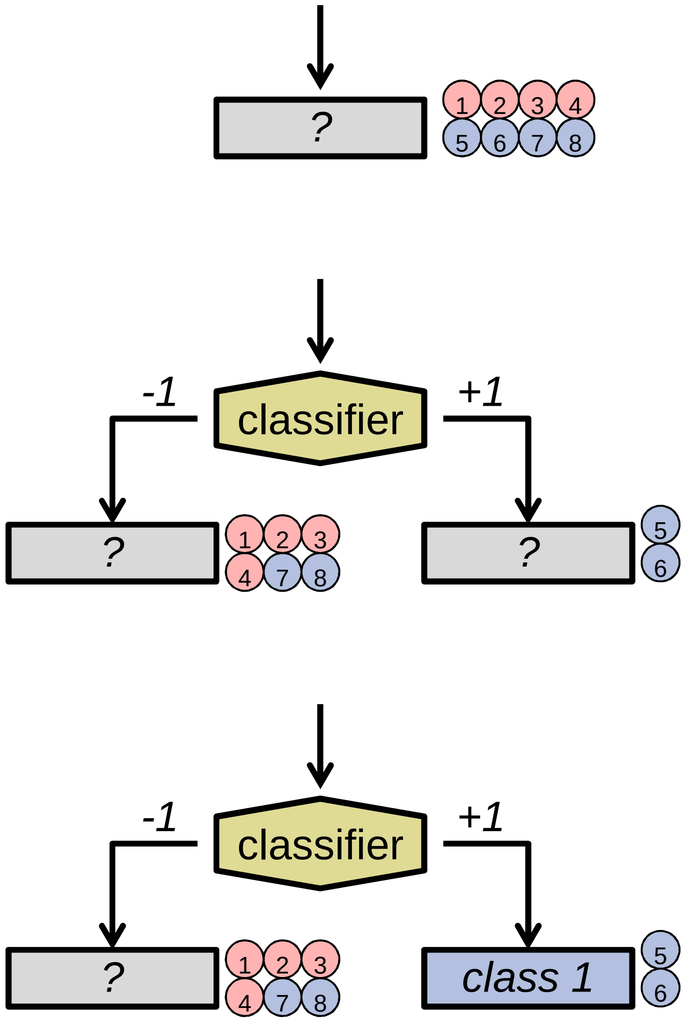
Decision Trees

- Decision Tree:
  - tree structure, branching factor 2
  - interior nodes: binary classifiers
  - leaf nodes: class labels



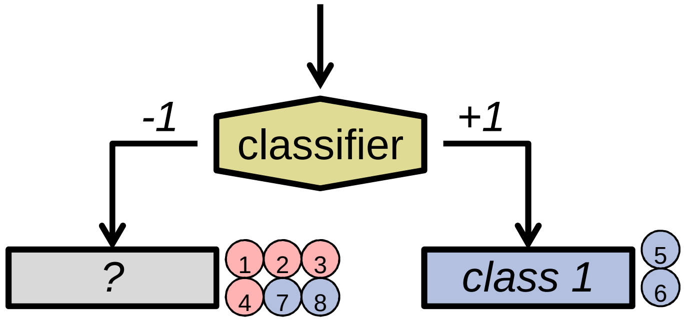
Decision Trees Training

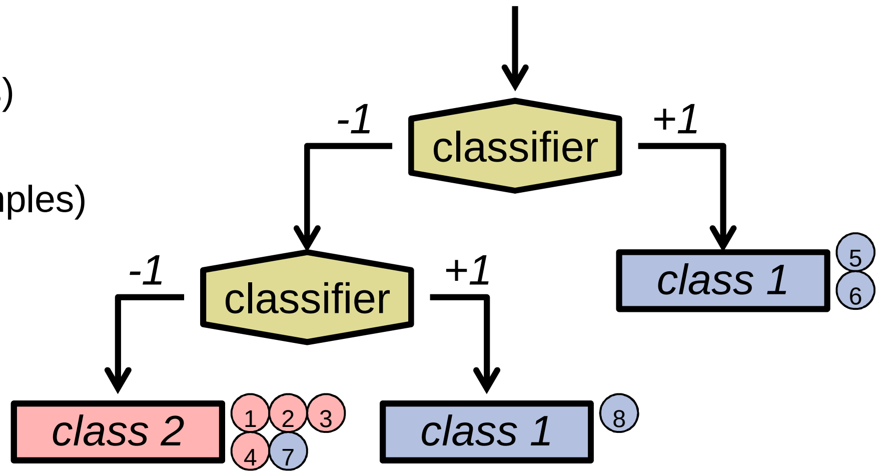
- creating decision trees from training examples
  - create leaf node with unknown class label as root node. Assign all training examples to it
  - **while** (unlabeled leaf nodes exist)
  - select unlabeled leaf node *n*
  - **if** (num. pos. examples >> num. neg. examples)
  - assign pos. label to node *n*
  - **elseif (**num. pos. examples << num. neg. examples)
  - assign neg. label to node *n*
  - **else**
  - train new classifier
  - replace leaf node *n* by classifier node
  - create two unlabeled leaf nodes
  - classify examples and assign them to the leaf nodes
  - **endif**
  - **endwhile**



Decision Trees Training cont.

- creating decision trees from training examples
  - create leaf node with unknown class label as root node. Assign all training examples to it
  - **while** (unlabeled leaf nodes exist)
  - select unlabeled leaf node *n*
  - **if** (num. pos. examples >> num. neg. examples)
  - assign pos. label to node *n*
  - **elseif (**num. pos. examples << num. neg. examples)
  - assign neg. label to node *n*
  - **else**
  - train new classifier
  - replace leaf node *n* by classifier node
  - create two unlabeled leaf nodes
  - classify examples and assign them to the leaf nodes
  - **endif**
  - **endwhile**





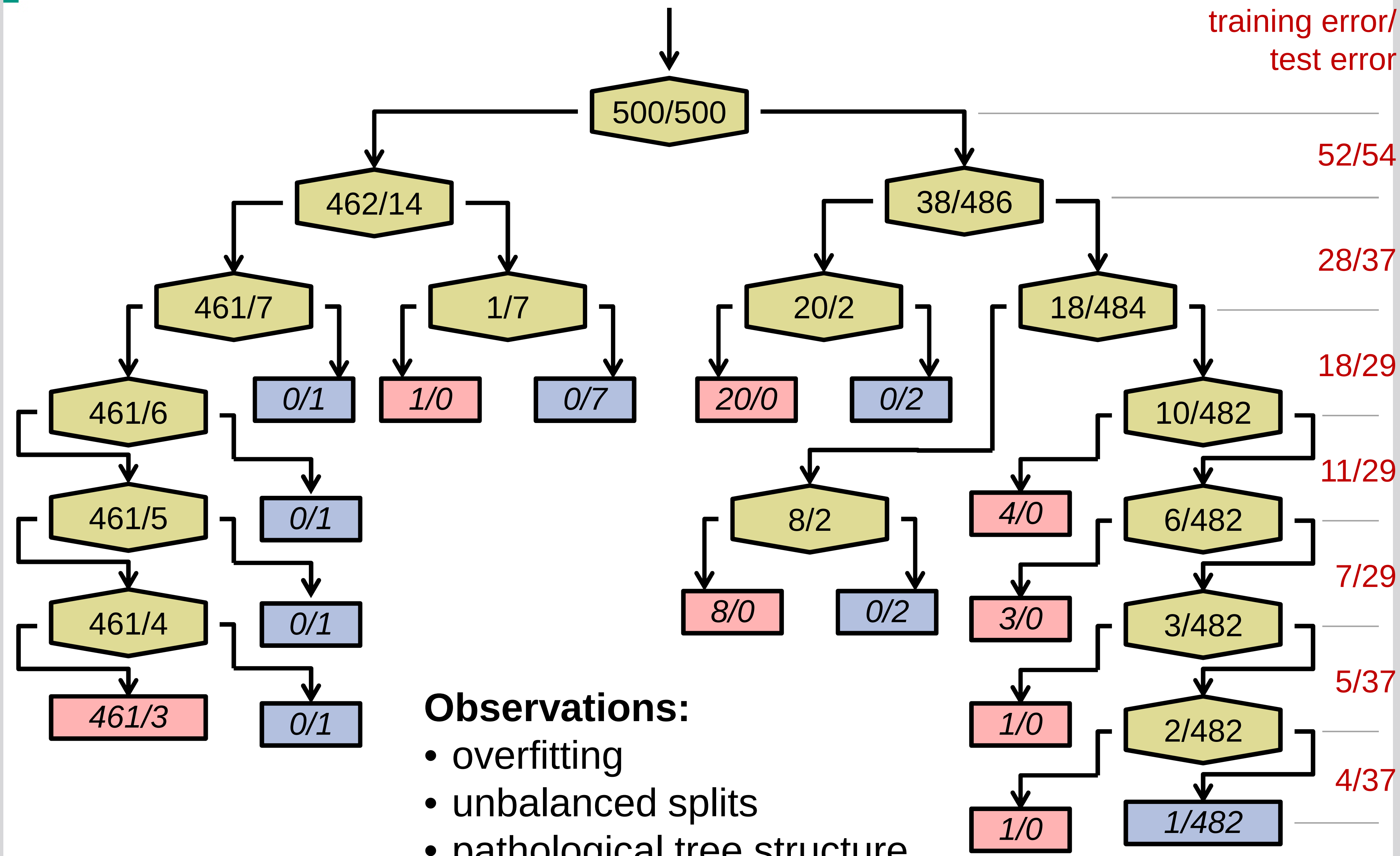
Decision Tree with Threshold Classifier

- Which classifiers are appropriate?
  - in general: all
  - similar idea like boosting:

    create a complex classifier by combining simple classifiers, i.e. threshold classifiers
- Example: digit recognition with Haar classifiers

  (next slide)

Example: Decision Tree for Digit Recognition



Observations:
- overfitting
- unbalanced splits
- pathological tree structure

Techniques to Improve Decision Trees

- Regularization techniques:
  - *Early Stopping*

    use a validation set while building the tree. Stop splitting nodes when you observe non-decreasing error on the validation set.

    Example: the validation error for digit recognition was:

    54 for tree of depth 1
    37 for tree of depth 2
    29 for tree of depth 3
    29 for tree of depth 4
    29 for tree of depth 5
    37 for tree of depth 6
    37 for tree of depth 7

  → take tree with depth 3

Techniques to Improve Decision Trees cont.

– *Pruning*

create complete decision tree first. Remove unbalanced or pathological branches afterwards.

- several pruning criteria
- implemented e.g. in decision tree algorithm C4.5

Techniques to Improve Decision Trees cont.

– *Randomized Decision Trees and Forests*

Randomize decision tree creation by:

- randomly selecting a subset of the training data (i.e. similar to bagging)
- randomly selecting a subset of features which serve as options for the next split
- randomly selecting the threshold for discrimination

Build an ensemble of many randomized trees

→ *Random Decision Forest*

Example: Decision Forest for Digit Recognition

- Building decision forests:
  - using Haar features
  - randomly selecting feature and threshold (best among *k* trials)
  - no variation of training set
  - allowing deep trees
  - varying the ensemble size *n*

- Error on test set:

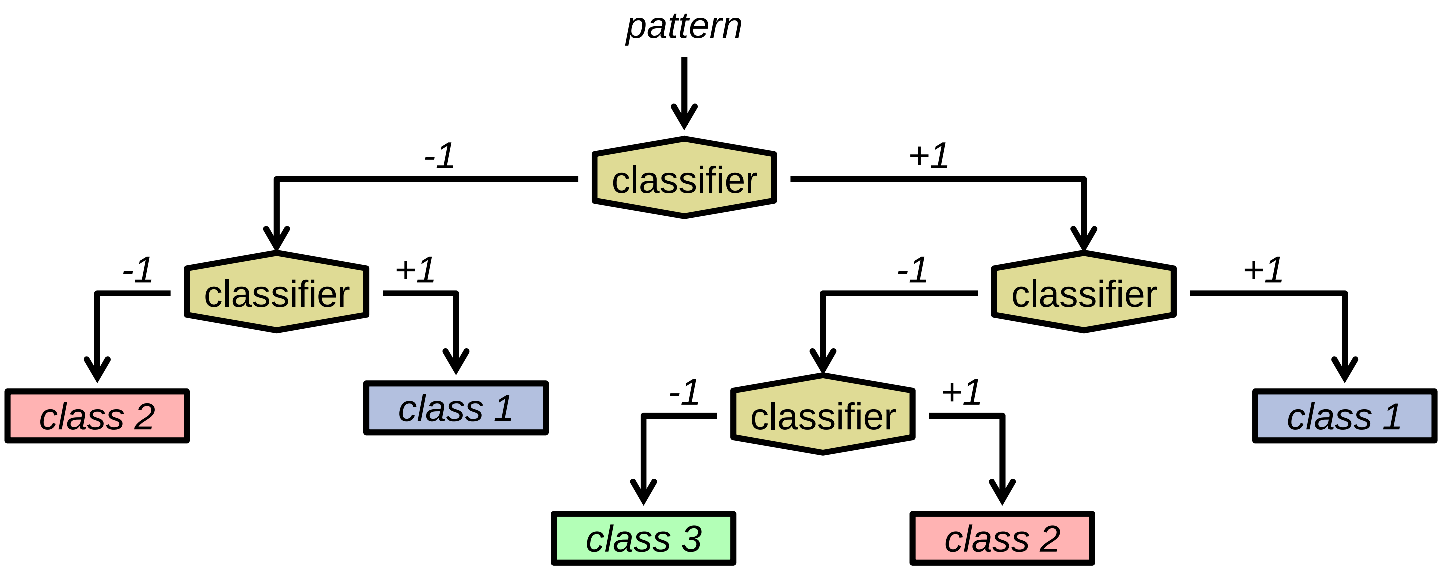
	n=1	n=10	n=30	n=80
k=1	336	398	374	392
k=10	90	53	36	32
k=30	43	29	20	17
k=80	41	25	15	11

comparison:

- decision tree trained with early stopping: 29
- AdaBoost ensembles
  - size 5: 54
  - size 50: 16
  - size 200: 10
- SVM: 6

Multi-Class-Classification with Decision Trees

- extension to more than two classes



  - classifiers are trained to minimize Shannon entropy in leaf nodes

$$-\sum_{leaf\ \ node\ n} \left(\frac{N_n}{N} \sum_{class\ c} \left(\frac{N_n^c}{N_n} \log_2 \frac{N_n^c}{N_n}\right)\right)$$

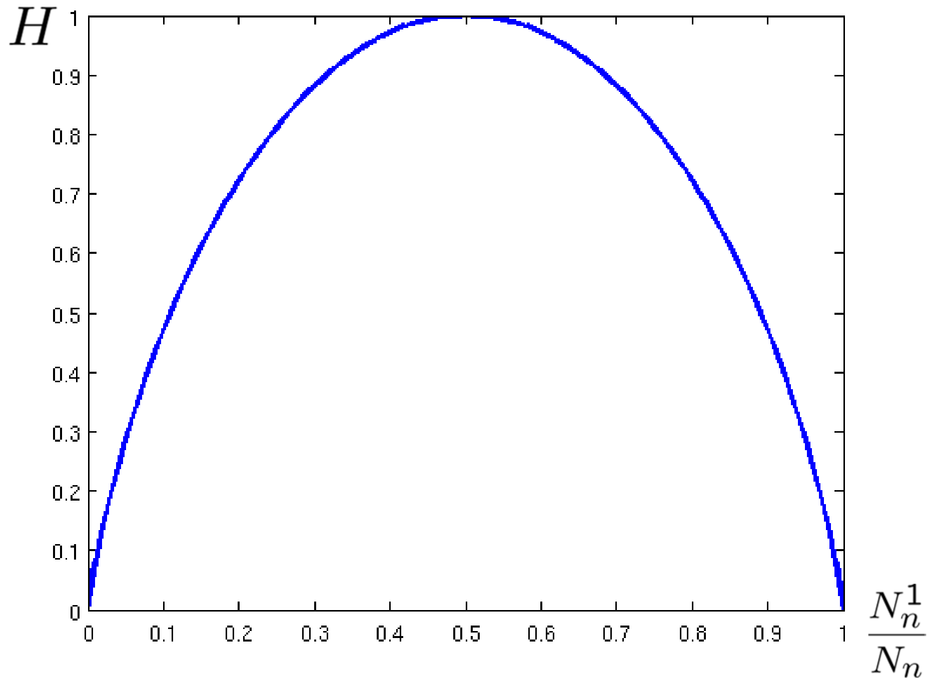
ratio of training patterns assigned to leaf node *n*

ratio of training patterns belonging to class *c* among those assigned to leaf node *n*

Multi-Class-Classification with Decision Trees cont.

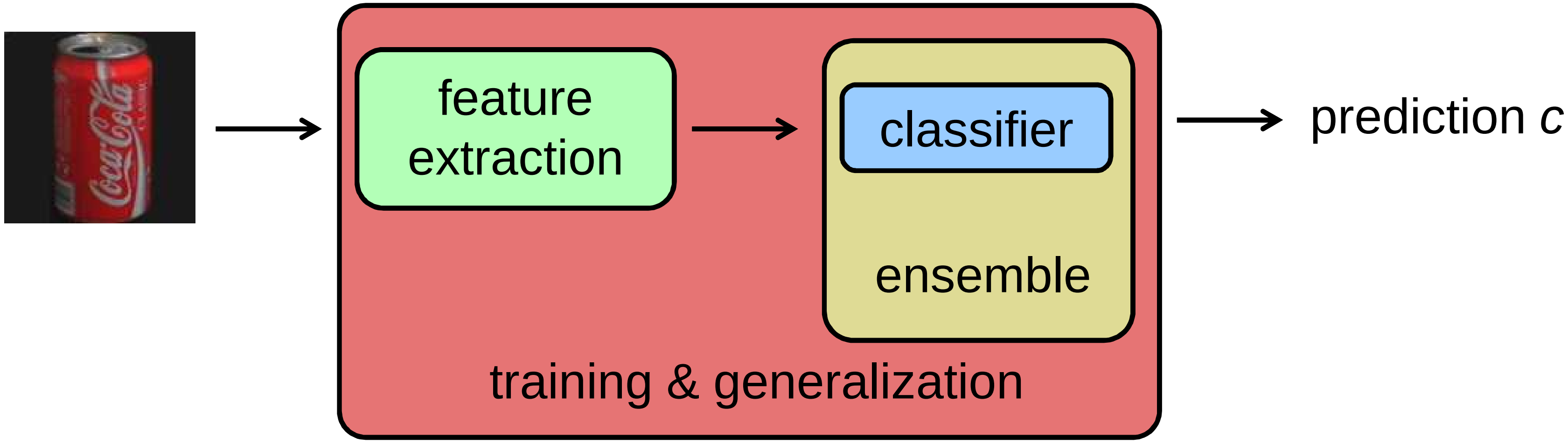
- Entropy measures the homogeneity of a pattern set
  - all patterns belong to the same class: entropy minimal (0)
  - same amount of patterns belongs to each class: entropy maximal

entropy in the 2-class case



SUMMARY: PATTERN RECOGNITION

Pattern Recognition: the Complete Picture

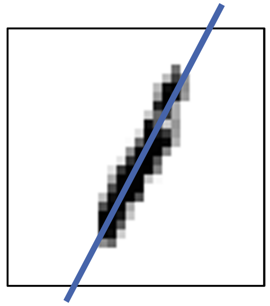


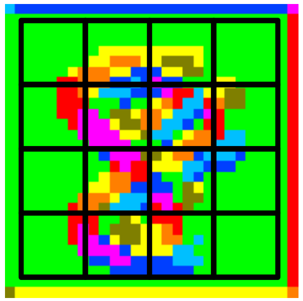
features	classifiers	ensembles	generalization
smart (specific)	SVM	free ensemble	validation
HOG	threshold	bagging	cross-validation
Haar	decision tree	boosting	data tuning
LBP	cascade	decision forest	early stopping
neural features	neural network	multi-class	randomization
…	…	…	…

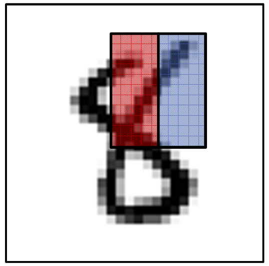
Pattern Recognition: the Complete Picture cont.

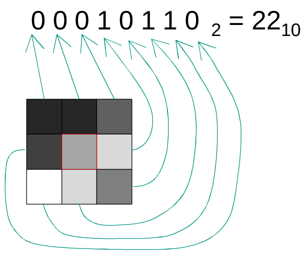
features
▪ smart (specific) ▪ HOG ▪ Haar ▪ LBP ▪ neural features ▪ …

- problem-specific features
  - pre processing (segmentation, filtering, …)
  - describe shape, color, template similarity
- gray level features
  - image subareas
- HOG
  - orientation histograms
  - accurate results
- Haar
  - easy and efficient calculation, integral images
  - edge, line, center-surround, chessboard
- LBP
  - local graylevel structure
  - easy and efficient calculation
- neural features
  - cf. chapter 12







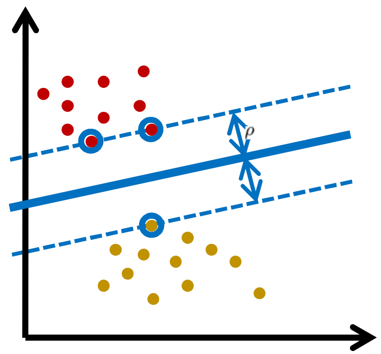


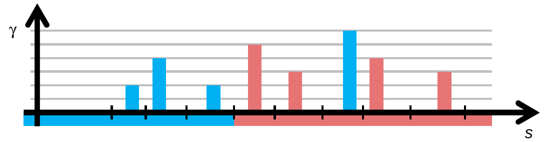
Pattern Recognition: the Complete Picture cont.

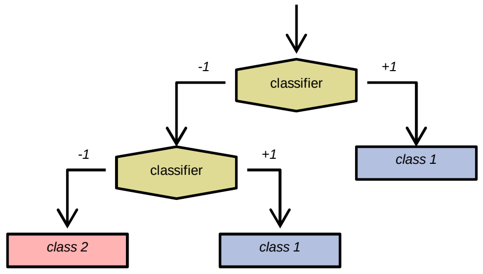
classifiers

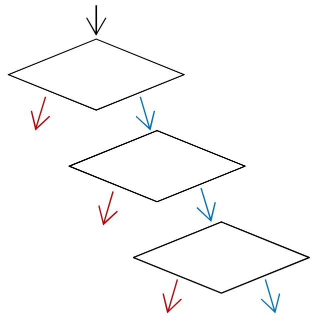
- SVM
- threshold
- decision tree
- cascade
- neural network
- …

- support vector machines
  - maximal margin classifier
  - soft margin approach to allow errors
  - kernel approach to model non-linearity
- threshold classifier
  - little expressive power, but can be combined by boosting, decision trees
- decision trees
  - powerful classifier combining simple classifiers in tree structure
  - tends to overfit, requires regularization
- cascade
  - set of classifiers with increasing complexity and specivity
- artificial neural network
  - cf. chapter 12







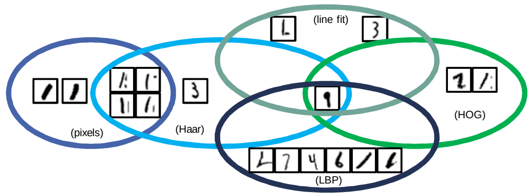


Pattern Recognition: the Complete Picture cont.

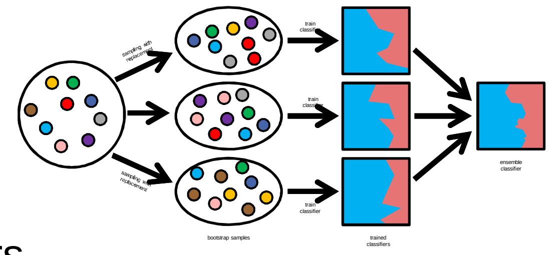
ensembles

- free ensemble
- bagging
- boosting
- multi-class
- …

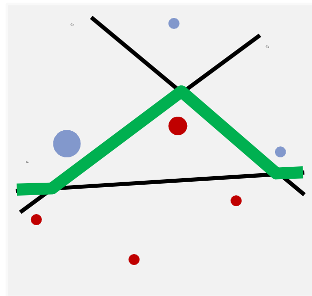
- Ensembles
  - combine “weak” classifiers to form a strong classifier by voting
  - independence & accuracy of members



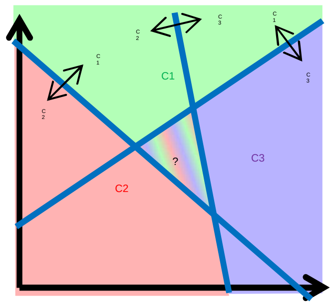
- Bagging
  - vary training set by bootstrapping



- Boosting
  - train classifiers that compensate errors of other ensemble members
  - weighted training examples
  - weighted voting
  - AdaBoost



- Multi-class classification
  - multi class decision trees
  - 1-versus-the rest approach
  - 1-versus-1 approach

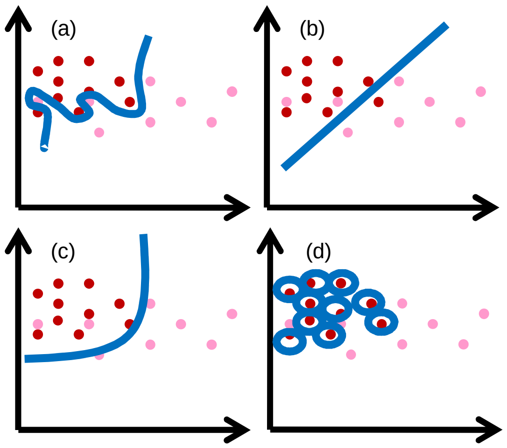


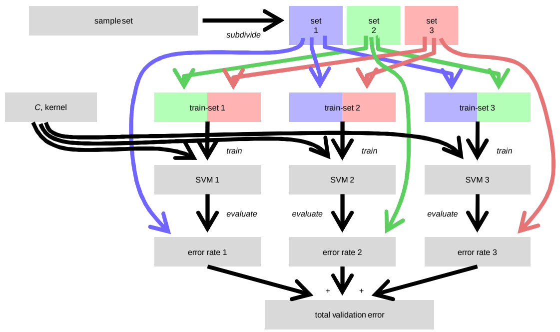
Pattern Recognition: the Complete Picture cont.

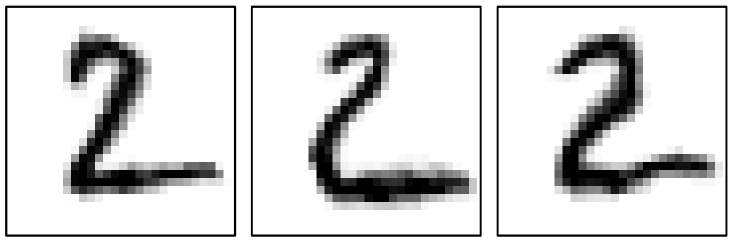
generalization

- validation
- cross-validation
- data tuning
- early stopping
- randomization
- …

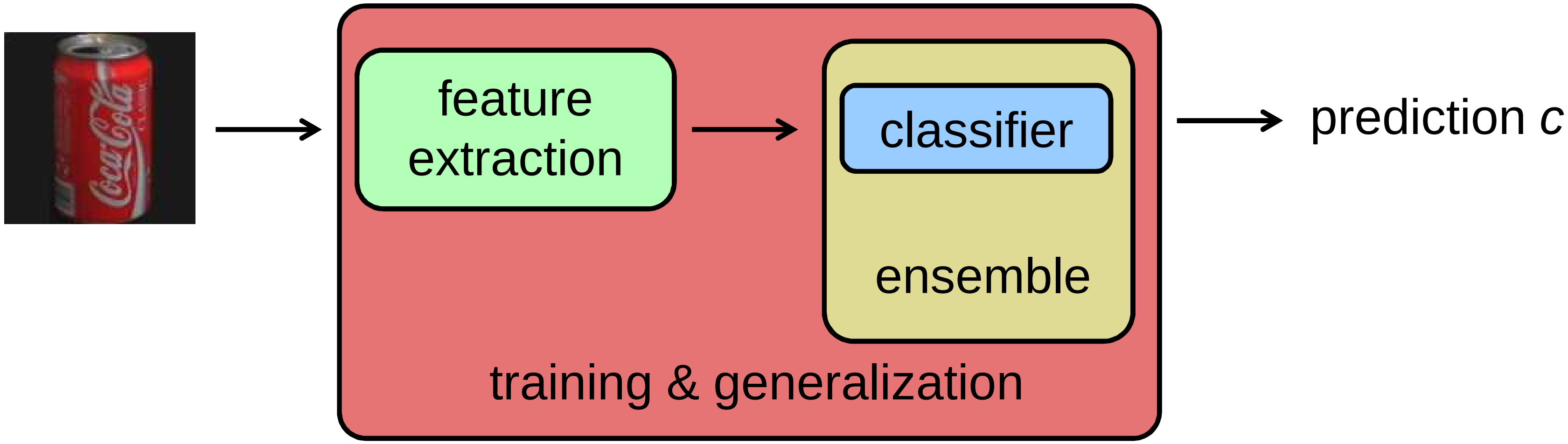
- generalization
  - don’t memorize training examples, learn the basic concepts
  - minimization of test error
  - overfitting, underfitting
- validation techniques
  - standard validation
  - cross-validation
- regularization techniques
  - early stopping
  - randomization & ensembles
  - more techniques in chapter 12
- data tuning
  - extend data set
  - modify/distort examples
  - improve quality of data





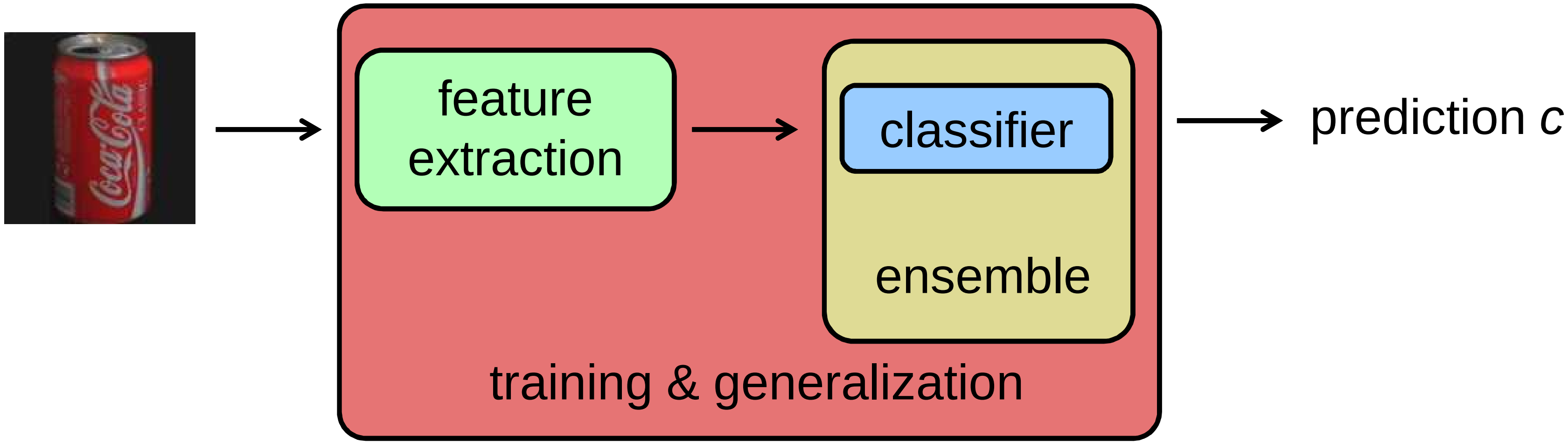


Pattern Recognition: the Complete Picture



- What matters most?
  1. good features
  2. as many training examples as possible
  3. consequent engineering of classification problem
  4. being aware of the generalization pitfalls
  5. type of classifier does not matter that much

Pattern Recognition: the Complete Picture



- Other techniques beyond the scope of the lecture
  - deformable part models
  - nearest neighbors classifiers
  - learning vector quantization
  - …