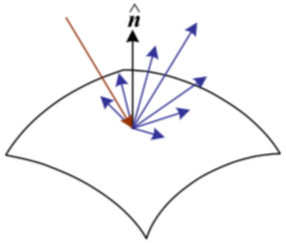
Object Detection: Perceptrons, Boosting, and SVMs

Frank Dellaert

CS 4476 Computer Vision at Georgia Tech

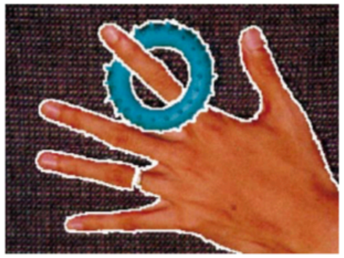
Several Slides by Lana Lazebnik and others



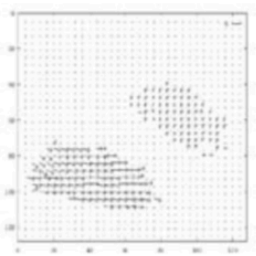
2. Image Formation

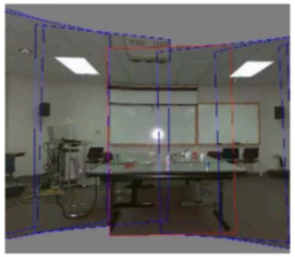
3. Image Processing

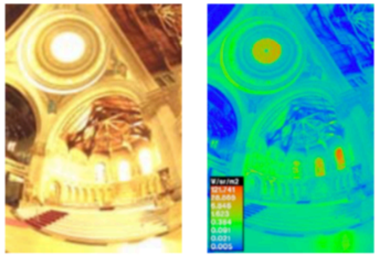
4. Features

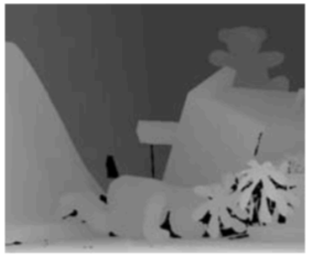

5. Segmentation

6-7. Structure from Motion

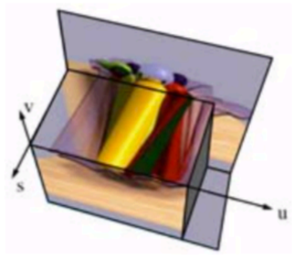

8. Motion


9. Stitching


10. Computational Photography


11. Stereo

12. 3D Shape



13. Image-based Rendering

14. Recognition

Recap: Classification

Face Detection

Linear Classifiers

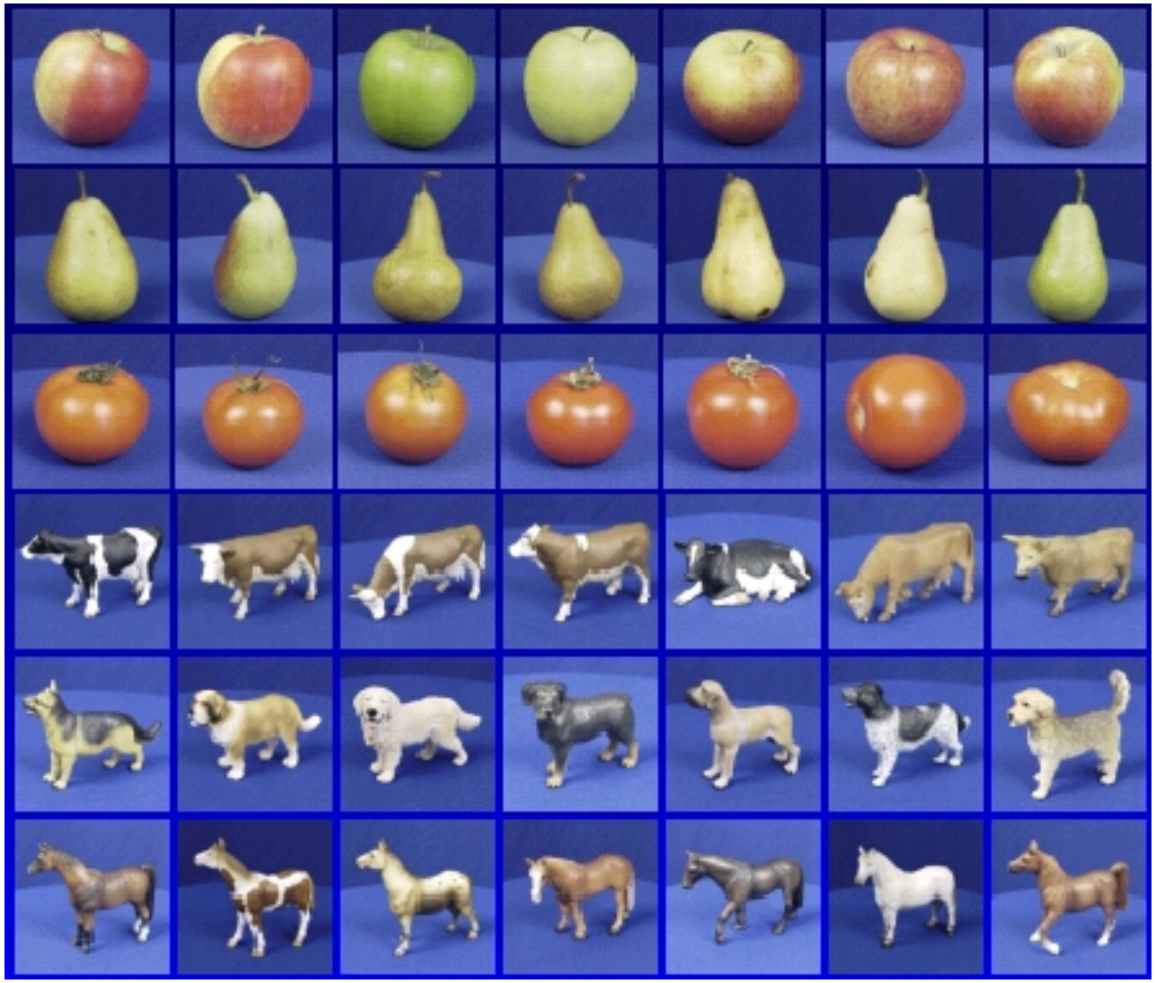
Boosting

Viola-Jones

Pedestrian Detection

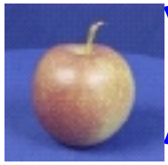
Support Vector Machines

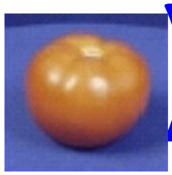
Image classification

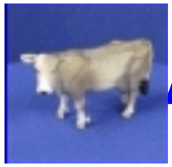


The statistical learning framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = "apple"

f() = "tomato"

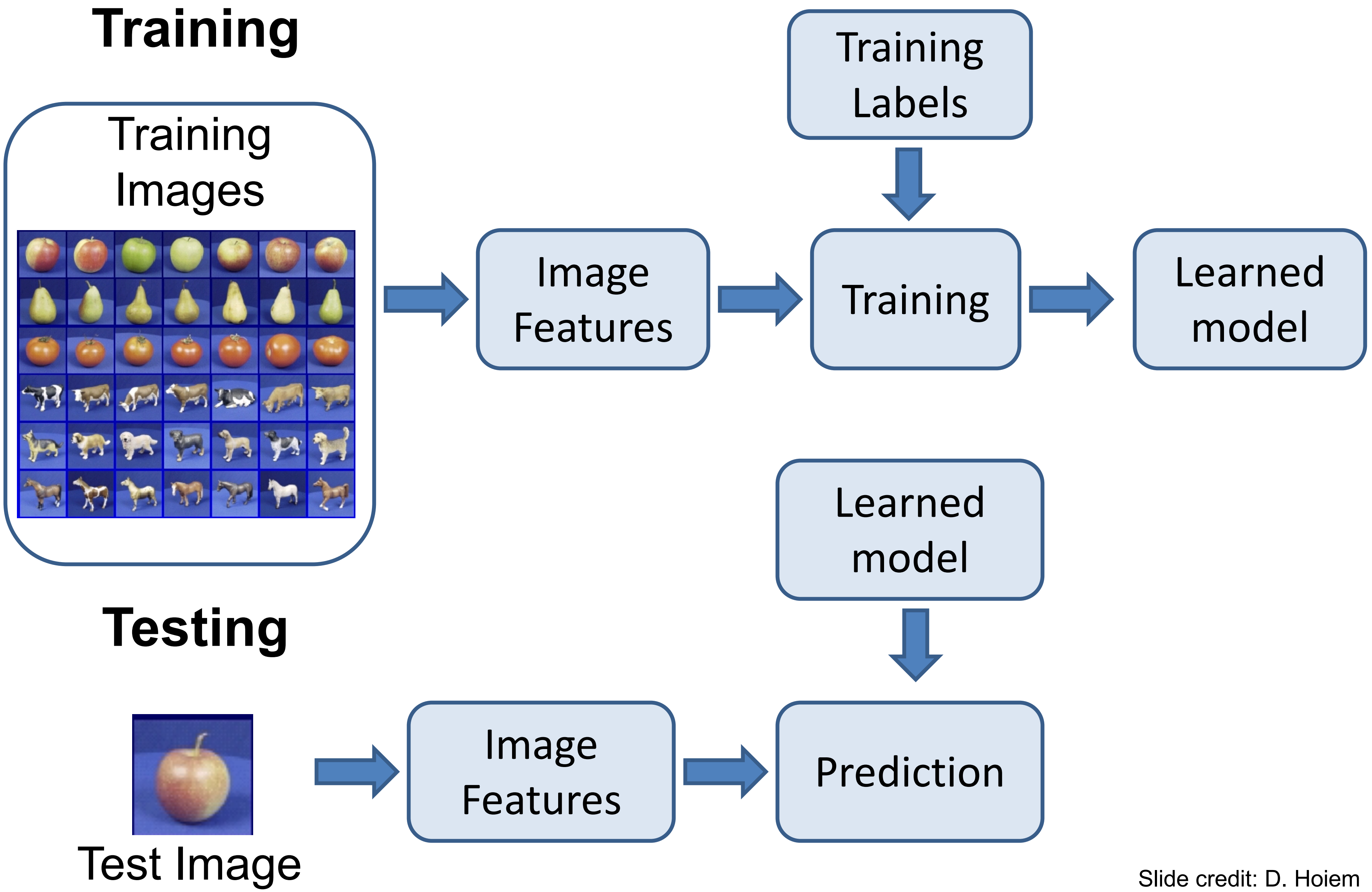
f() = "cow"

The statistical learning framework

$$\underset{\text{output}}{\underset{\uparrow}{y}} = \underset{\text{prediction function}}{\underset{\uparrow}{f}}(\underset{\text{feature representation}}{\underset{\nwarrow}{\mathbf{x}}})$$

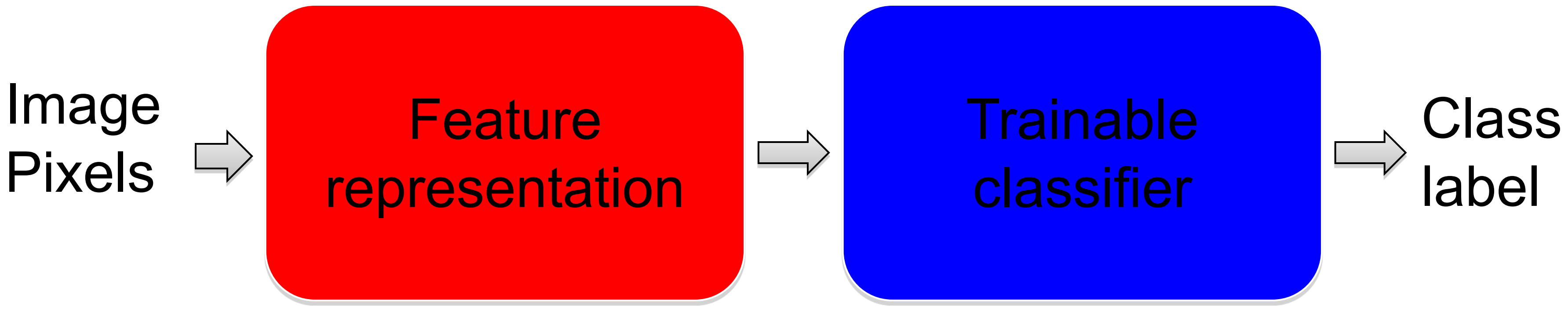
- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

Steps



Slide credit: D. Hoiem

“Classic” recognition pipeline

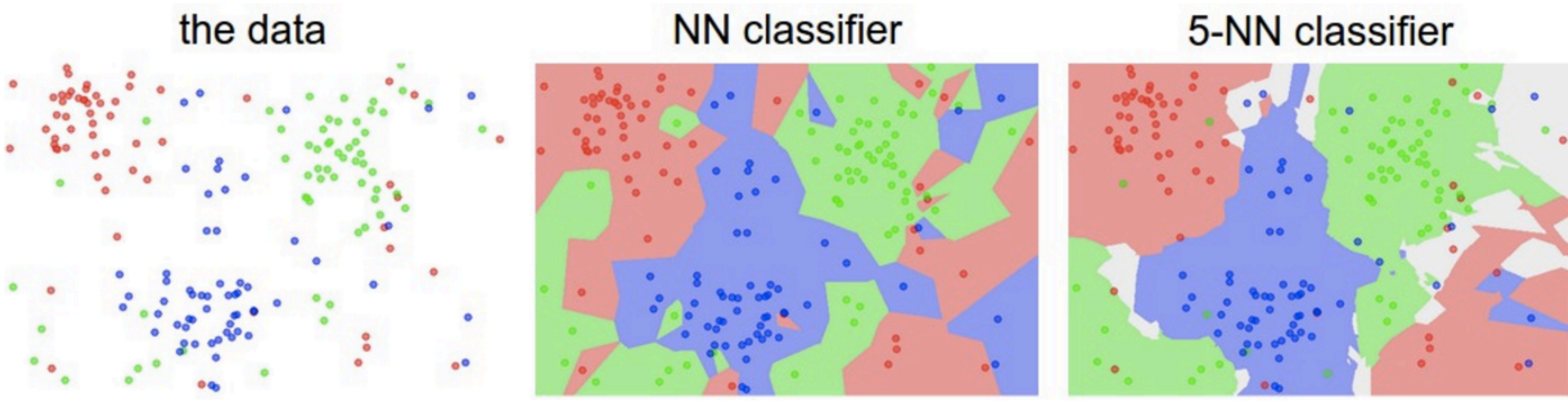


- Hand-crafted feature representation
- Off-the-shelf trainable classifier

Very brief tour of some classifiers

- **K-nearest neighbor**
- **Perceptron**
- **Boosting**
- **SVM**
- Naïve Bayes
- Bayesian network
- Randomized Forests
- **Convolutional Neural networks**

K-nearest neighbor classifier



Credit: Andrej Karpathy, http://cs231n.github.io/classification/

Recap: Classification

Face Detection

Linear Classifiers

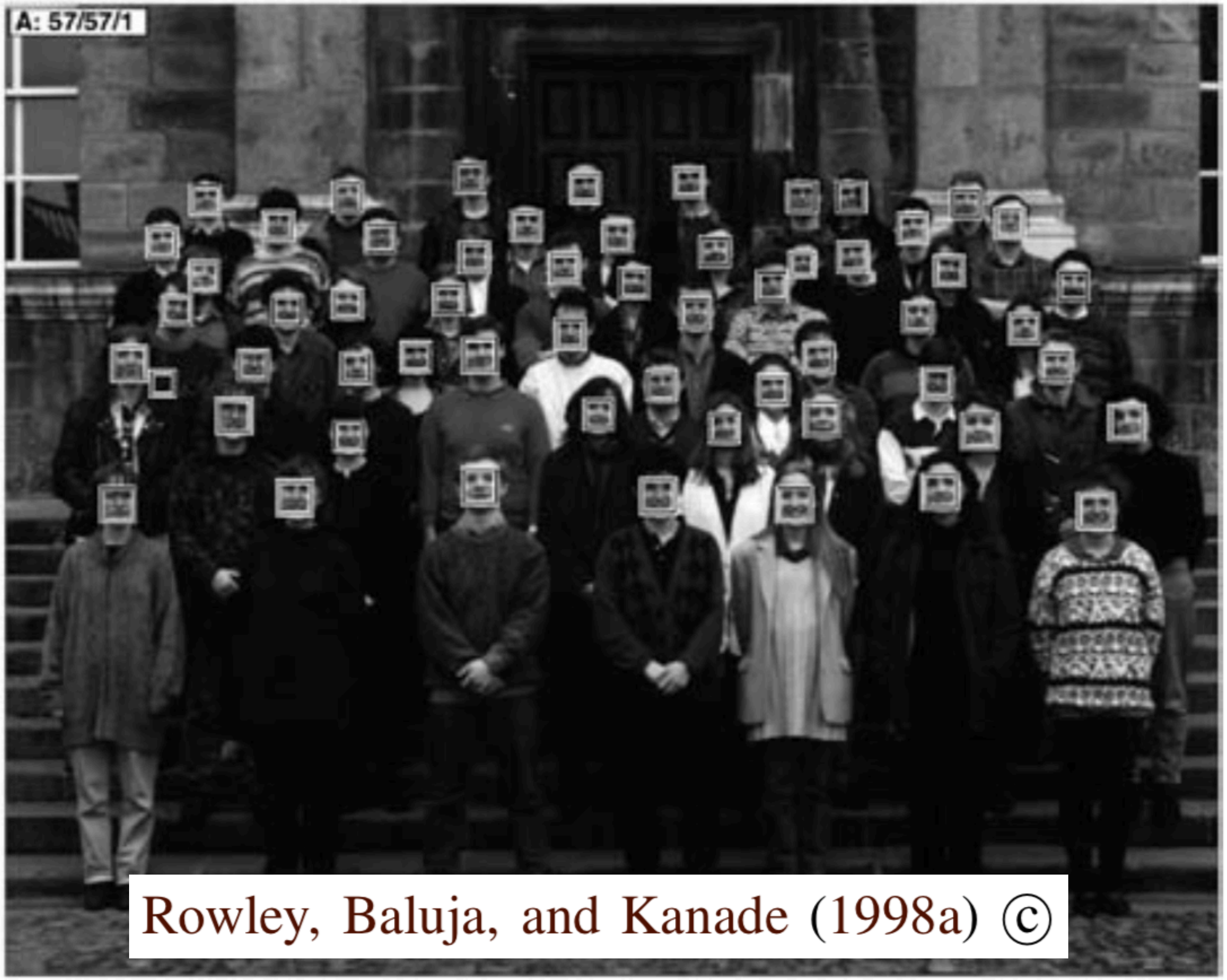
Boosting

Viola-Jones

Pedestrian Detection

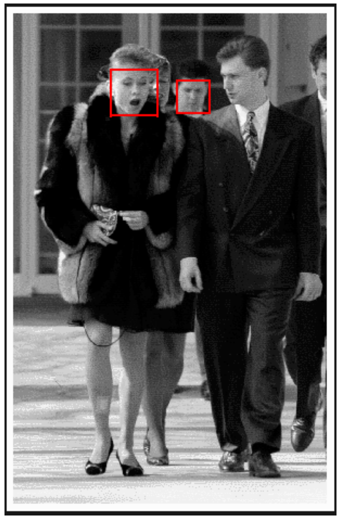
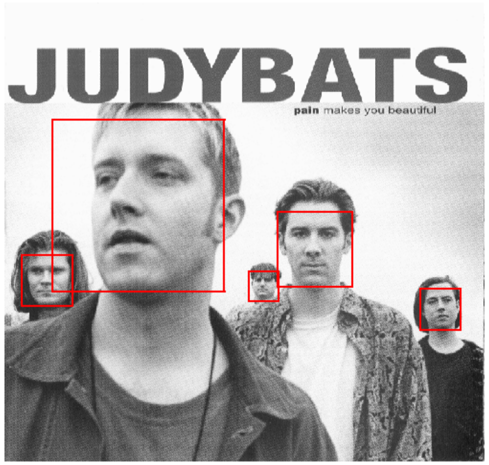
Support Vector Machines

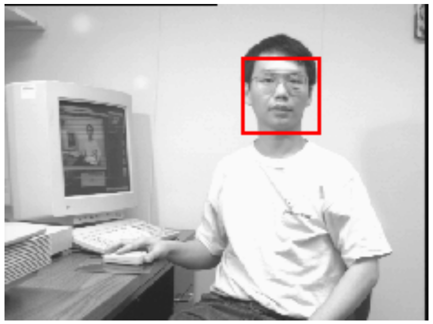
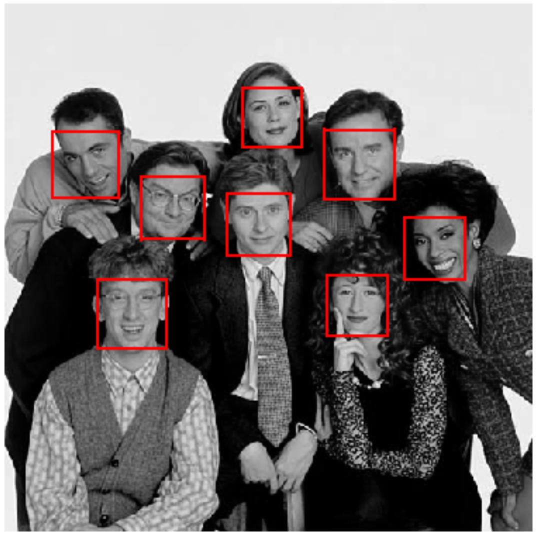
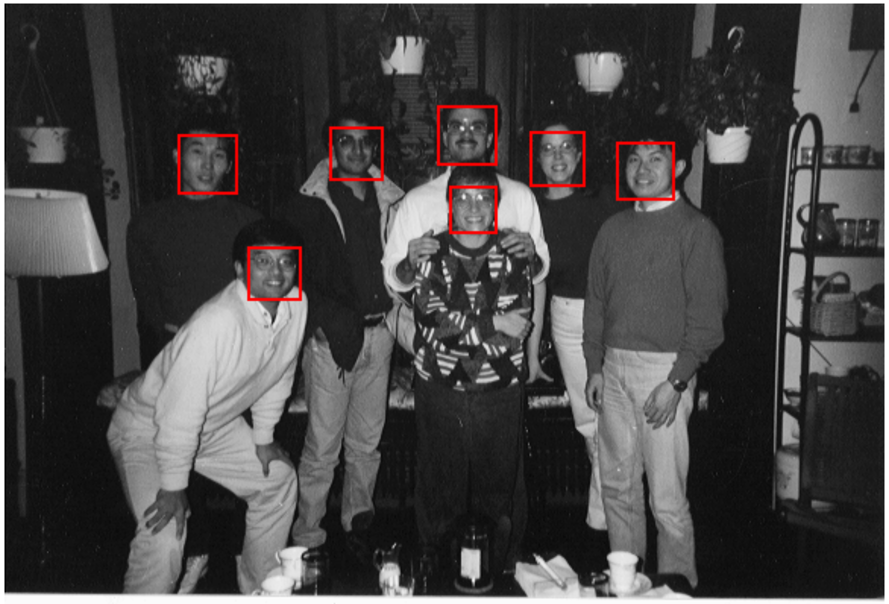
Face Detection: the problem



Rowley, Baluja, and Kanade (1998a) ©

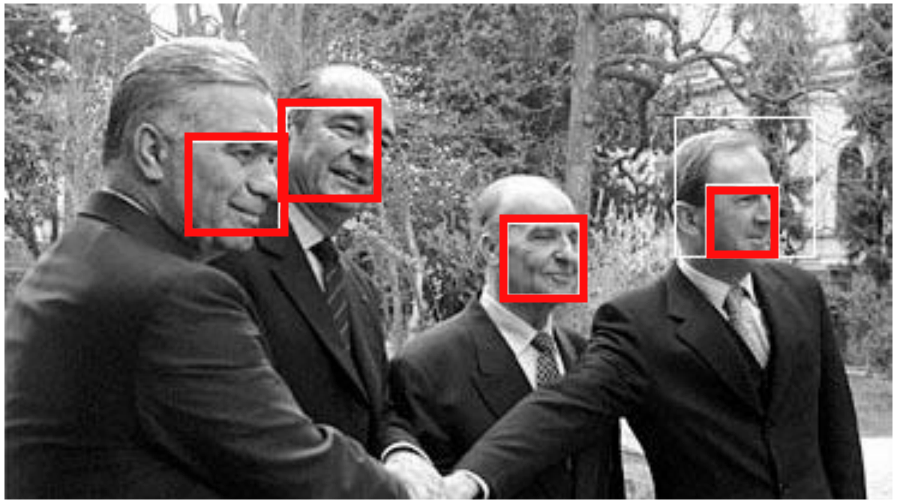
Output of Face Detector on Test Images

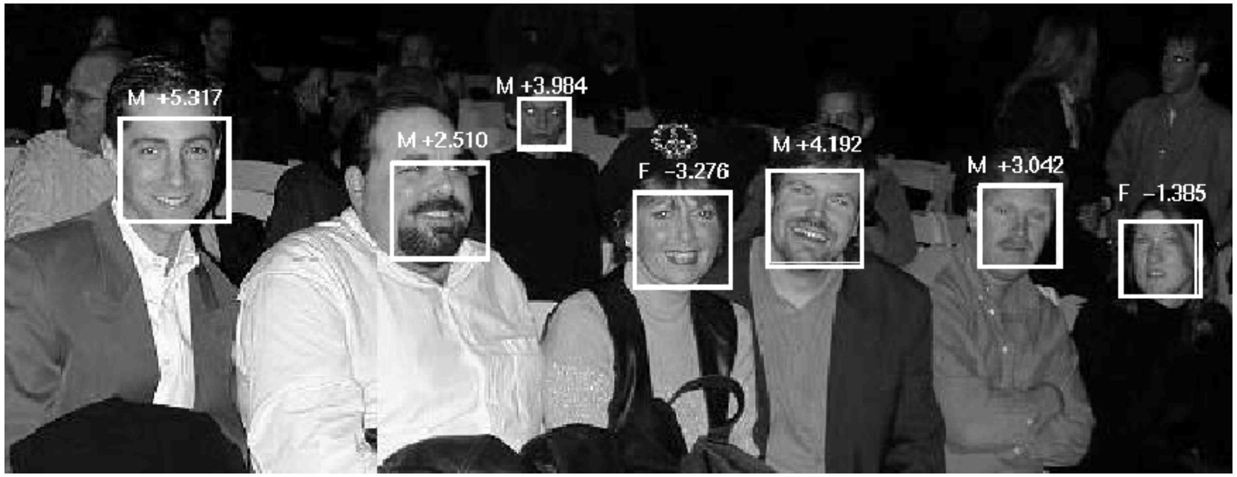
Solving other "Face" Tasks

Facial Feature Localization



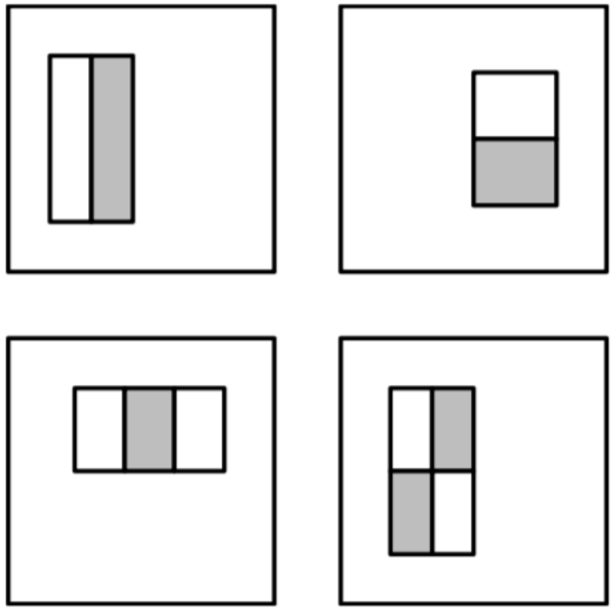
Profile Detection

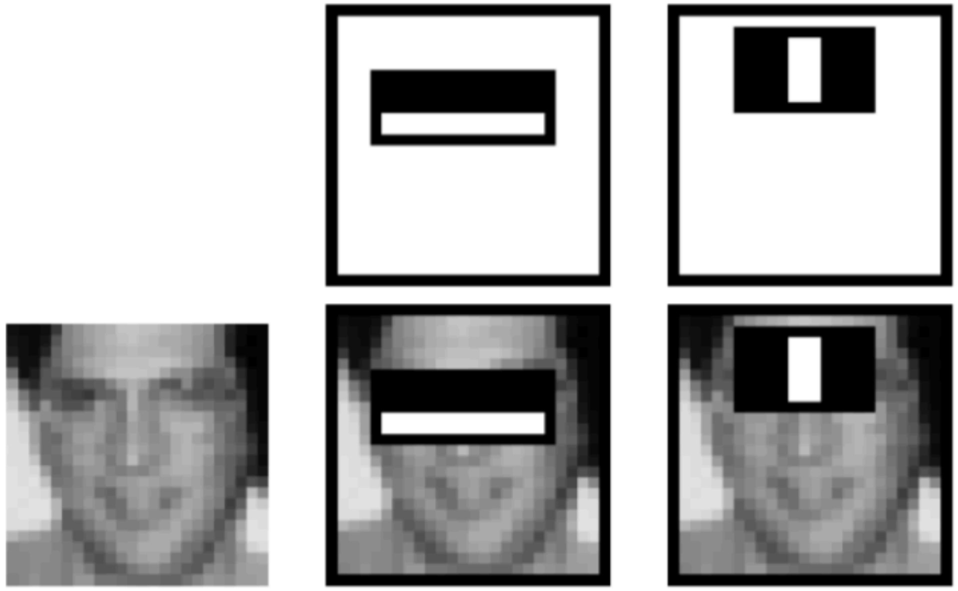
Demographic Analysis



Haar Wavelet Features

Can be computed very efficiently using “integral transform”.



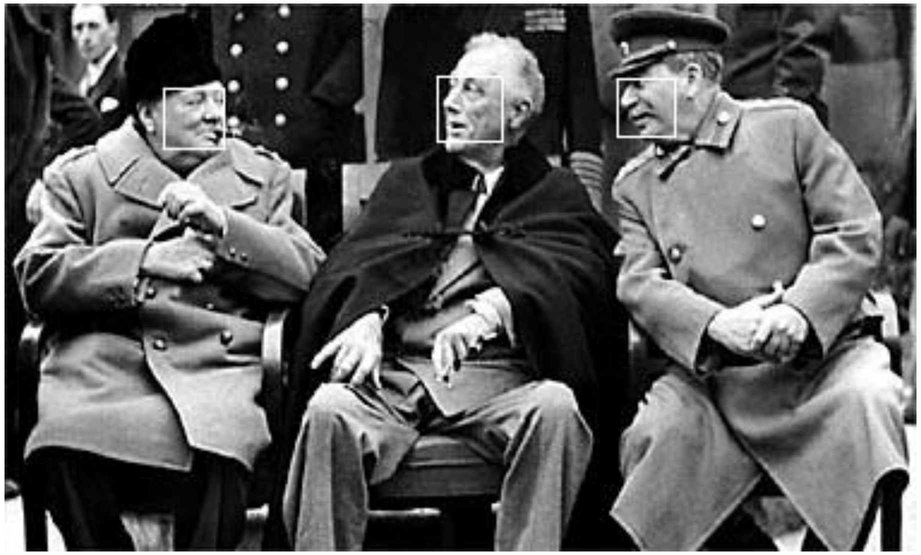


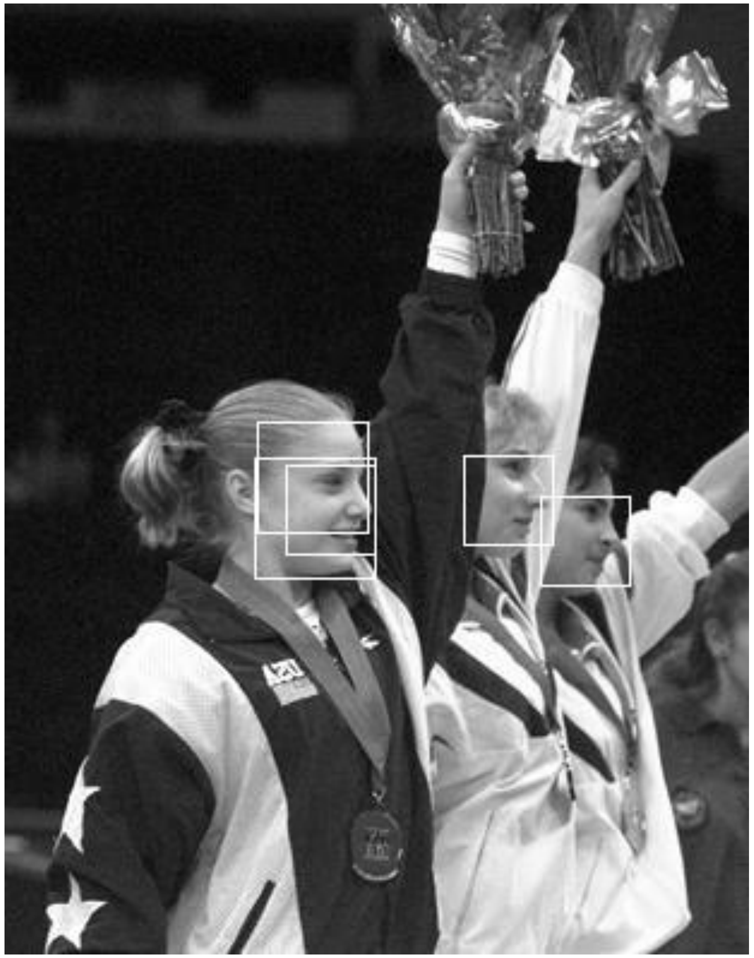
Introduced by Viola & Jones, CVPR 2001
“Rapid object detection using a boosted cascade of simple features”

- One of the most influential CV papers
- 25K citations (SIFT: 54K)

- 384 by 288 pixel images
- 15 fps on 700 MHz Intel Pentium III ☺
- 24x24 detector windows
- 180K possible features
- Scanned at multiple locations/scales

Profile Detection





Recap: Classification

Face Detection

Linear Classifiers

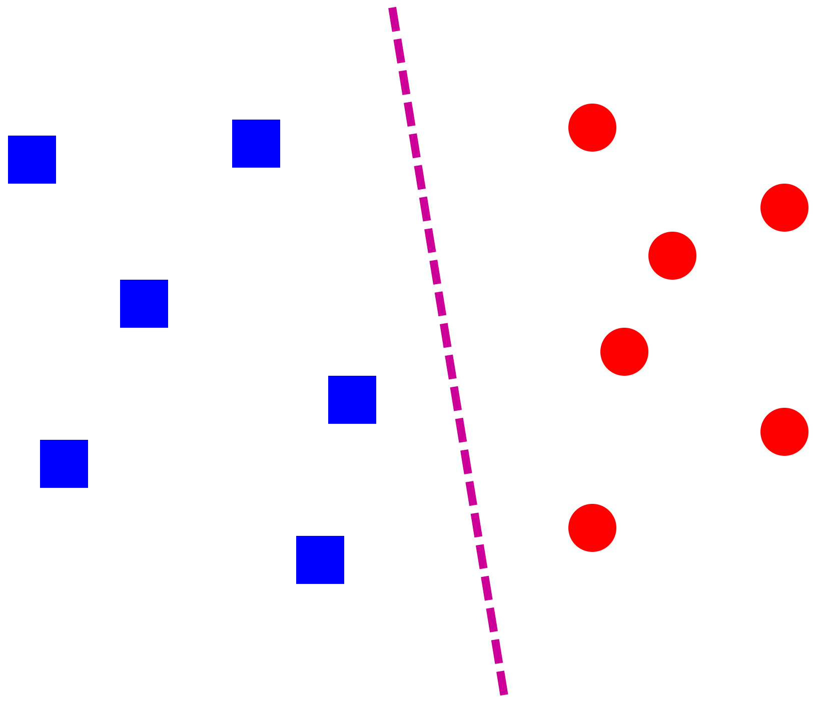
Boosting

Viola-Jones

Pedestrian Detection

Support Vector Machines

Linear classifiers aka Perceptrons

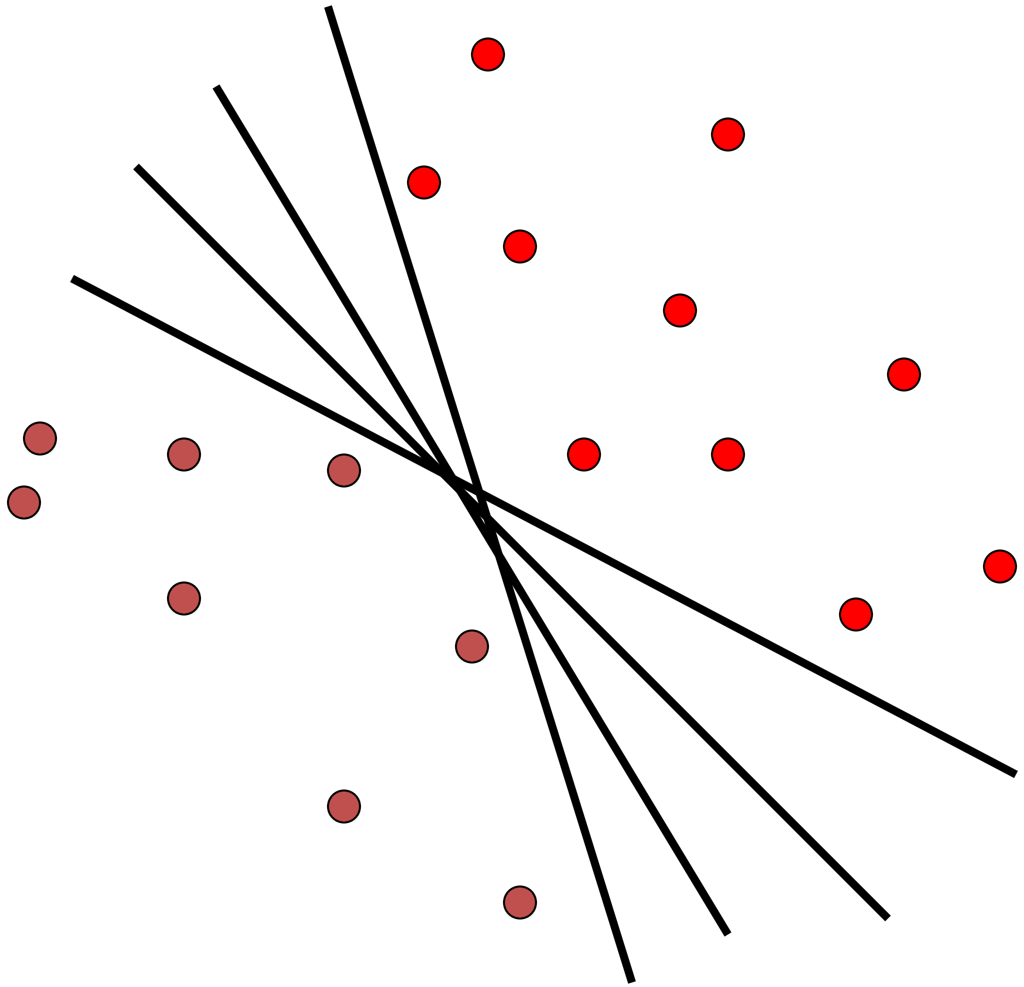


- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x} + b)$$

Linear classifiers aka Perceptrons

- When the data is linearly separable, there may be more than one separator (hyperplane)



Which separator is best?

Perceptron Learning

```
1. Start with w=0, b=0
2. R=max |$x_i$|
3. For i=1:N
       if $y_i$(<w$x_i$>+b)<=0
           w += f $y_i$ $x_i$
           b += f $y_i$ $R^2$
4. Terminate if all examples correctly classified
```

Nearest neighbor vs. linear classifiers

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- NN cons:
  - Need good distance function
  - Slow at test time
- Linear pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time
- Linear cons:
  - Works for two classes
  - How to train the linear function?
  - What if data is not linearly separable?

Recap: Classification

Face Detection

Linear Classifiers

Boosting

Viola-Jones

Pedestrian Detection

Support Vector Machines

Szeliski, Chapter 14, Section 14.1.1

Weak learners

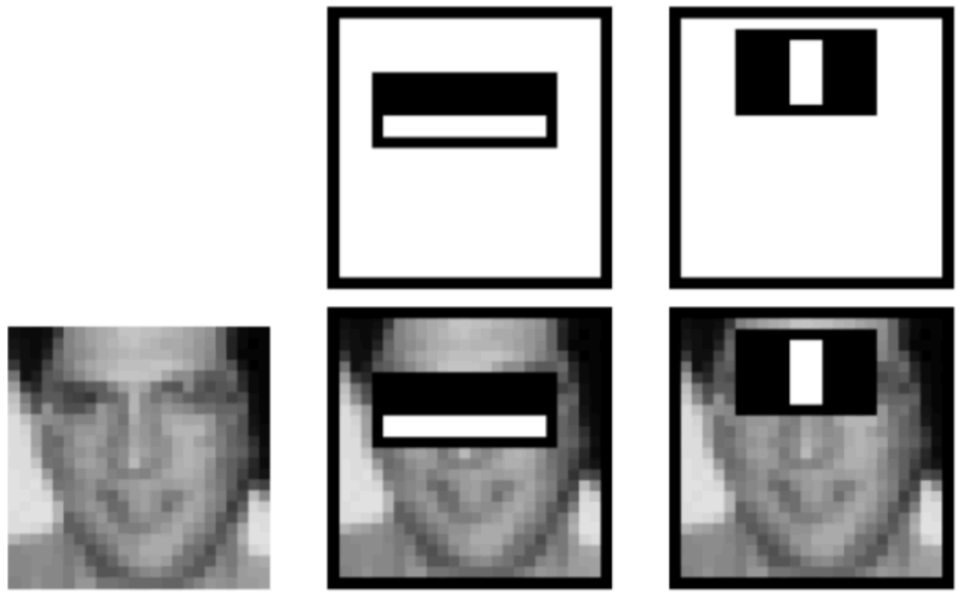
- Accurate learners = hard
- “so-so” learners = easy

- Example:
  - if “buy” occurs in email, classify as SPAM

- Weak learner = “rule of thumb”

More Weak Learners

- Perceptron
- Decision stumps
- Haar wavelets:



Adaboost Algorithm

A Super Efficient Feature Selector

- Features = Weak Classifiers
- Each round selects the optimal feature given:
  - Previous selected features
  - Exponential Loss

AdaBoost

- Given a set of weak classifiers

$$h_j(\boldsymbol{x}) = a_j[f_j < \theta_j] + b_j[f_j \geq \theta_j] = \begin{cases} a_j & \text{if } f_j < \theta_j \\ b_j & \text{otherwise,} \end{cases}$$

  – None much better than random

- Iteratively combine classifiers
  – Form a linear combination

$$h(\boldsymbol{x}) = \operatorname{sign}\left[\sum_{j=0}^{m-1} \alpha_j h_j(\boldsymbol{x})\right]$$

  – Training error converges to 0 quickly
  – Test error is related to training margin

Adaboost Algorithm

For each training stage $j = 1 \ldots M$:

(a) Renormalize the weights so that they sum up to 1 (divide them by their sum).

(b) Select the best classifier $h_j(\boldsymbol{x}; f_j, \theta_j, s_j)$ by finding the one that minimizes the weighted classification error

$$\boxed{\begin{aligned} e_j &= \sum_{i=0}^{N-1} w_{i,j} e_{i,j}, \\ e_{i,j} &= 1 - \delta(y_i, h_j(\boldsymbol{x}_i; f_j, \theta_j, s_j)). \end{aligned}} \quad \begin{aligned} &(14.3) \\ &(14.4) \end{aligned}$$

For any given f_j function, the optimal values of (θ_j, s_j) can be found in linear time using a variant of weighted median computation (Exercise 14.2).

(c) Compute the modified error rate β_j and classifier weight α_j,

$$\boxed{\beta_j = \frac{e_j}{1 - e_j} \quad \text{and} \quad \alpha_j = -\log \beta_j.} \qquad (14.5)$$

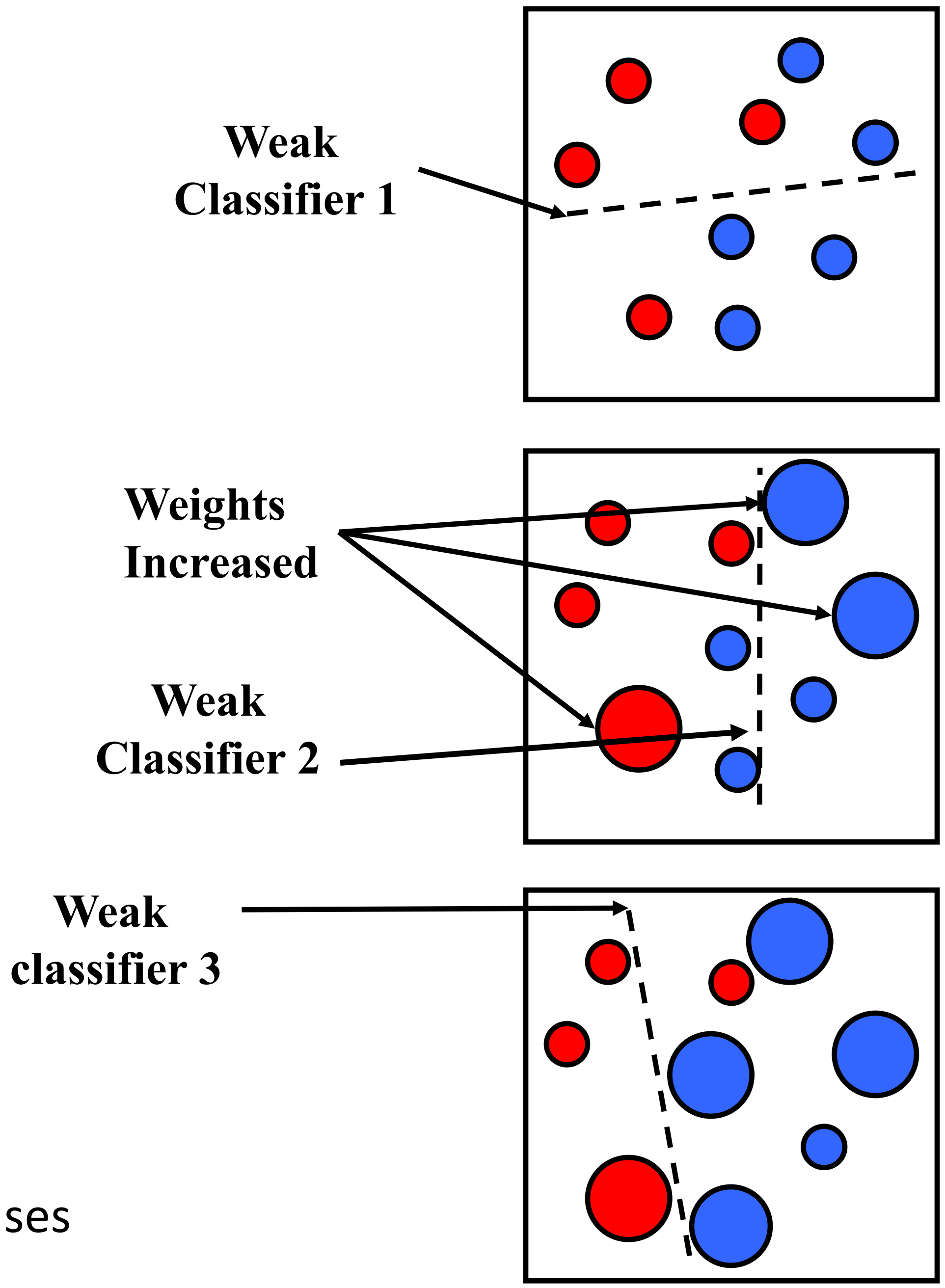
(d) Update the weights according to the classification errors $e_{i,j}$

$$\boxed{w_{i,j+1} \leftarrow w_{i,j} \beta_j^{1-e_{i,j}},} \qquad (14.6)$$

AdaBoost at work…

Final classifier is linear combination of weak classifiers:

$$h(\boldsymbol{x}) = \mathrm{sign}\left[\sum_{j=0}^{m-1} \alpha_j h_j(\boldsymbol{x})\right]$$



Surprising Phenomenon:

- After all examples are classified:
- Test error keeps decreasing !
- Connection with SVM: Margin increases

Recap: Classification

Face Detection

Linear Classifiers

Boosting

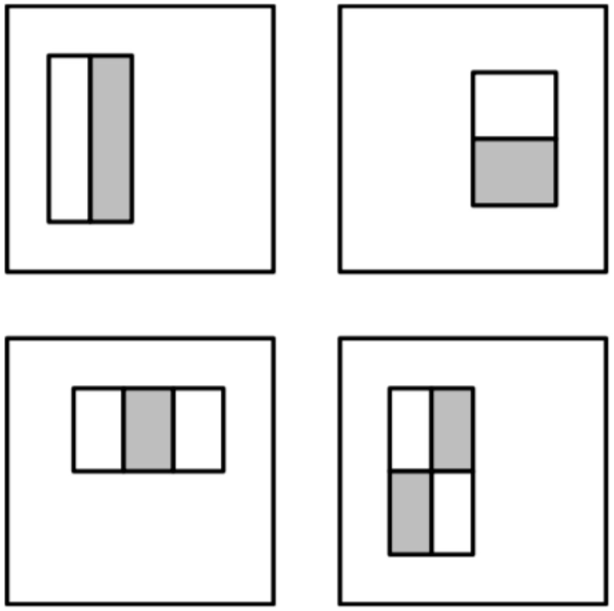
Viola-Jones

Pedestrian Detection

Support Vector Machines

Recap: Haar Wavelet Features

"Rectangle filters"



$$h(\boldsymbol{x}) = \text{sign}\left[\sum_{j=0}^{m-1} \alpha_j h_j(\boldsymbol{x})\right]$$

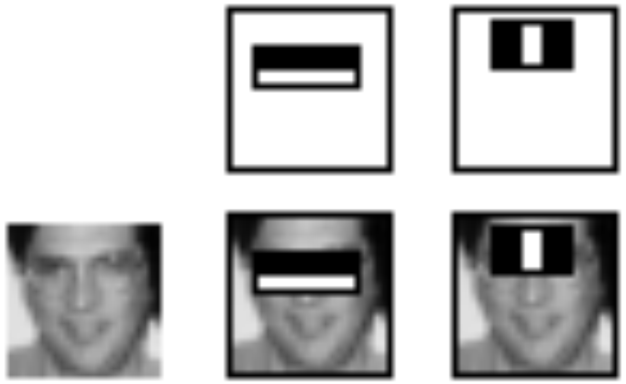
$$h_j(\boldsymbol{x}) = a_j[f_j < \theta_j] + b_j[f_j \geq \theta_j] = \begin{cases} a_j & \text{if } f_j < \theta_j \\ b_j & \text{otherwise,} \end{cases}$$

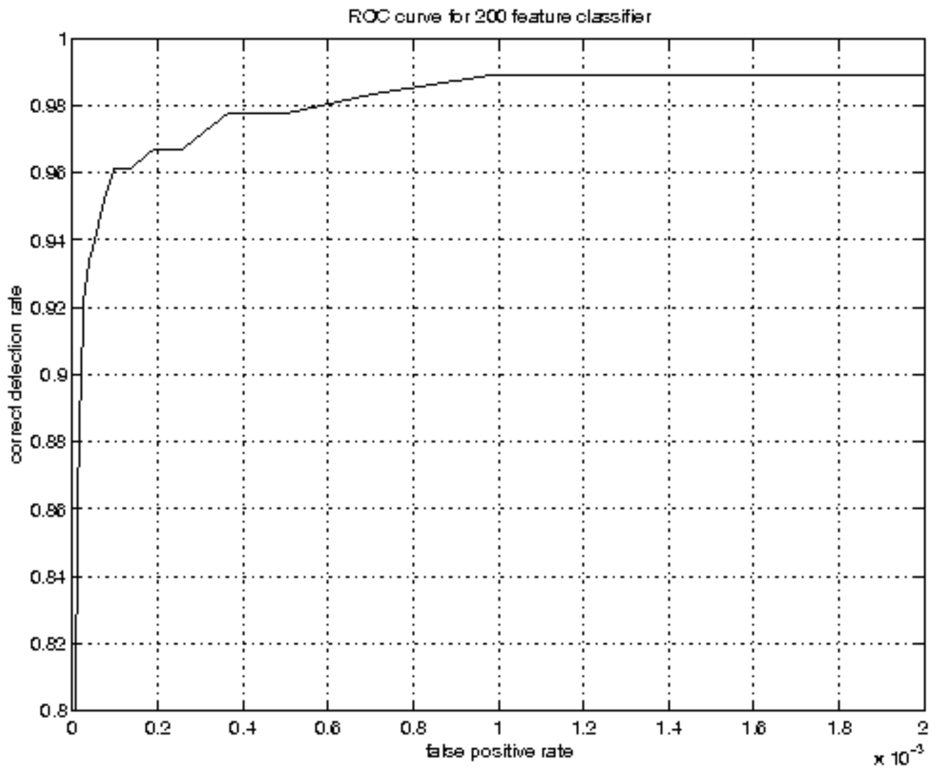
Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost

95% correct detection on test set with 1 in 14084 false positives.

Not quite competitive...

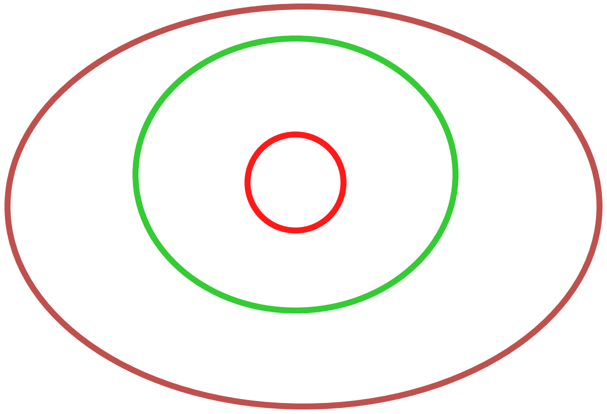


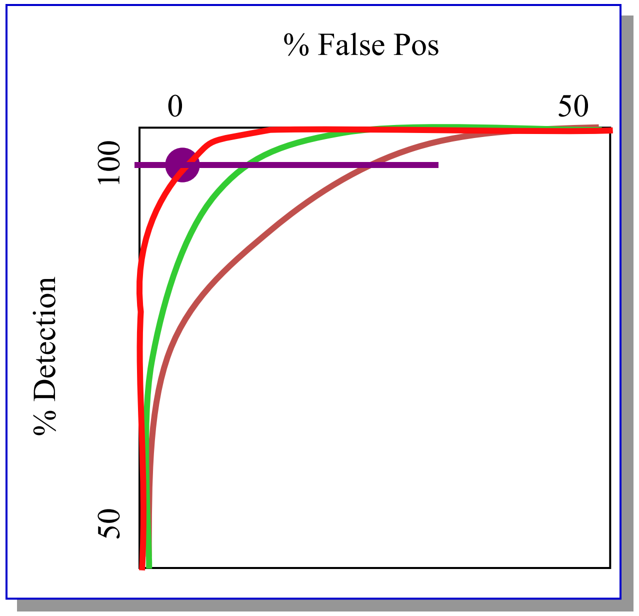


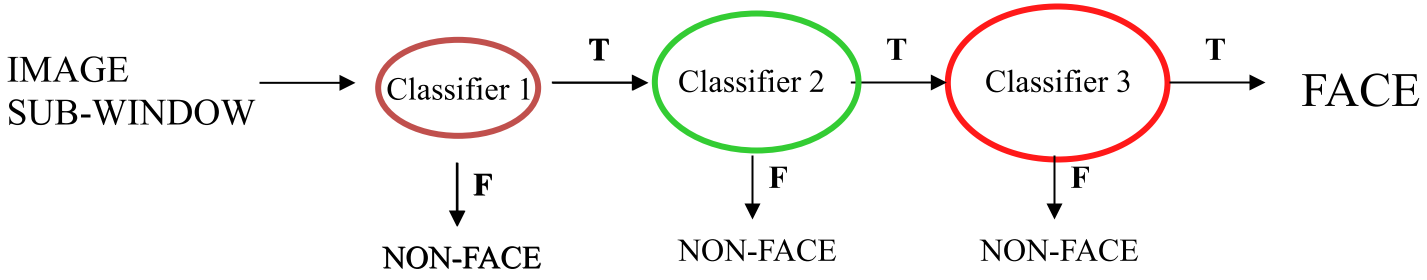
ROC curve for 200 feature classifier

Building better (and faster) classifiers

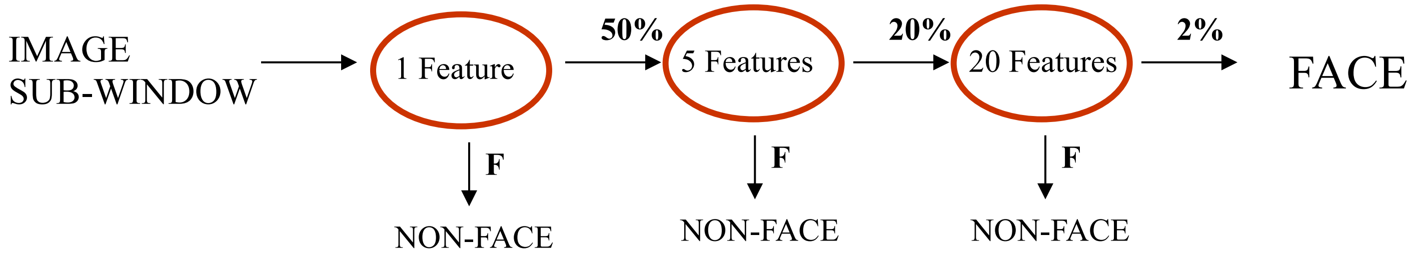
- Given a nested set of classifier hypothesis classes





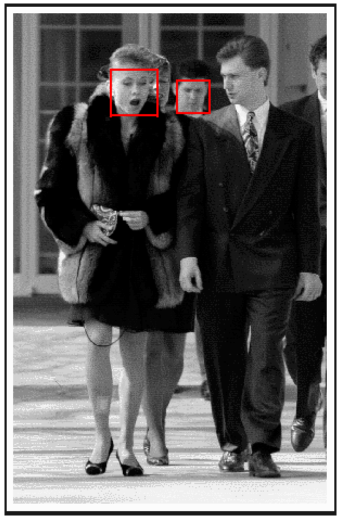


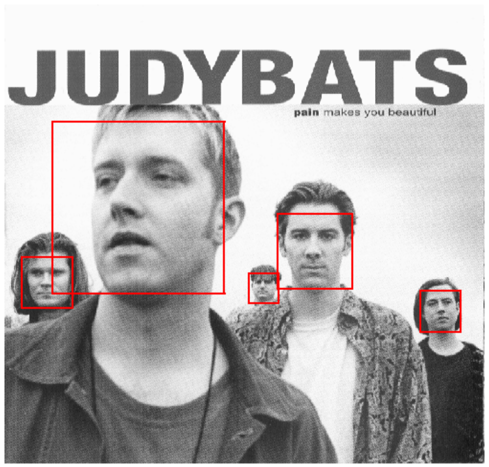
Cascaded Classifier

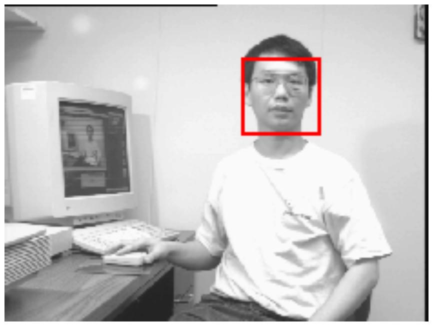


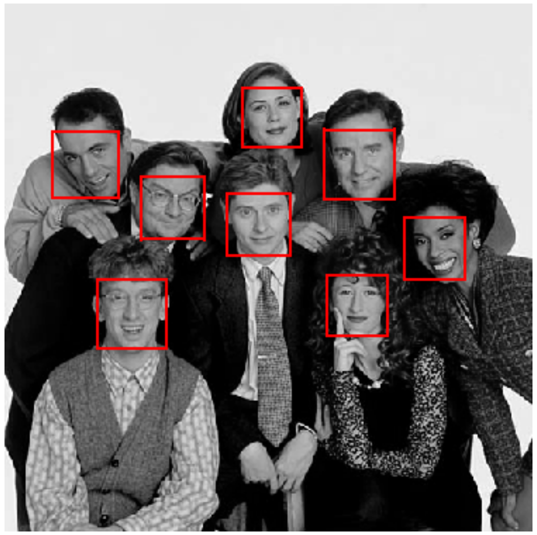
- A 1 feature classifier achieves 100% detection rate and about 50% false positive rate.
- A 5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative)
  - using data from previous stage.
- A 20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)

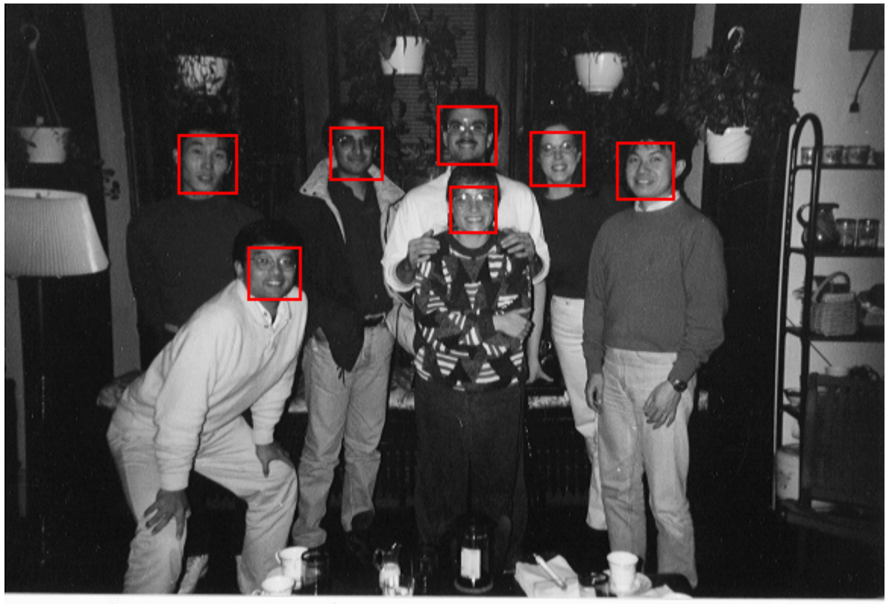
Output of Face Detector on Test Images





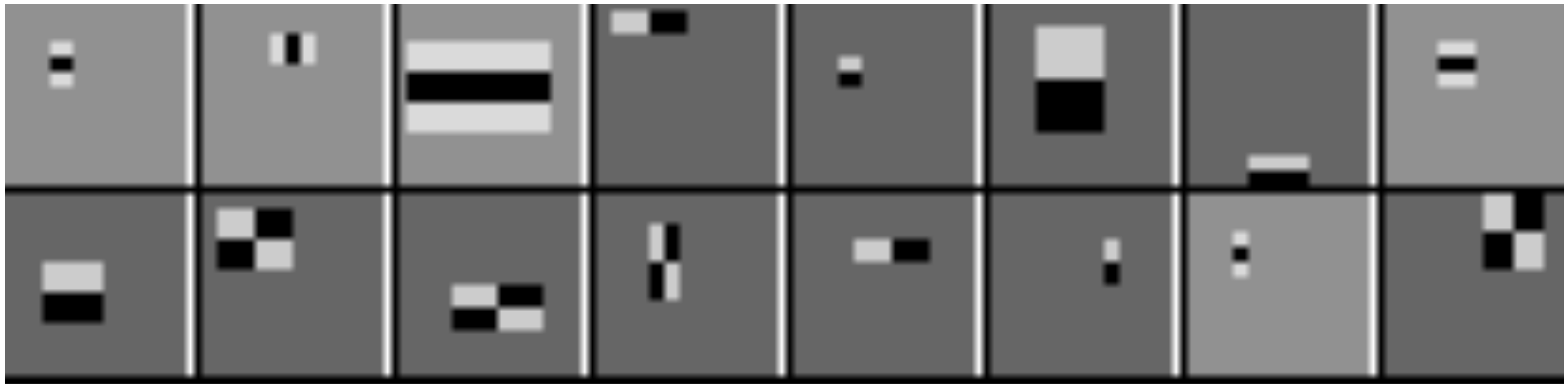




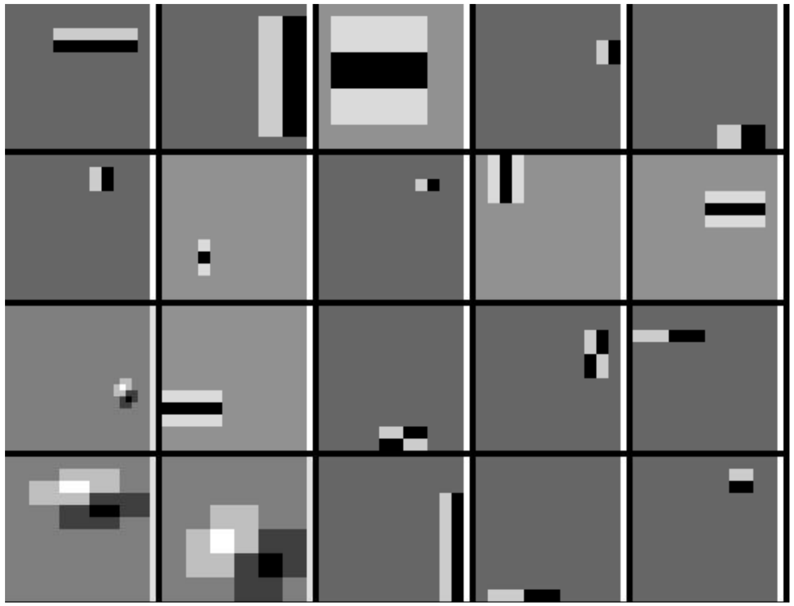


Feature Localization Features

- Learned features reflect the task



Profile Features



Recap: Classification

Face Detection

Linear Classifiers

Boosting

Viola-Jones

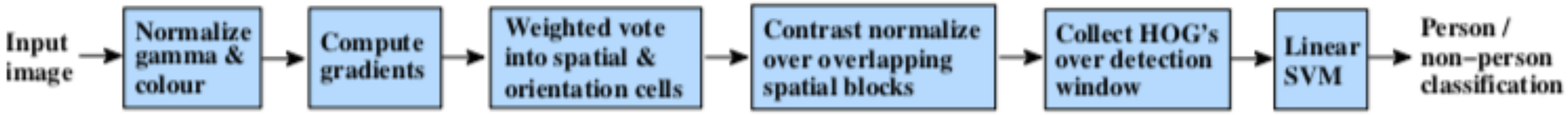
Pedestrian Detection

Support Vector Machines

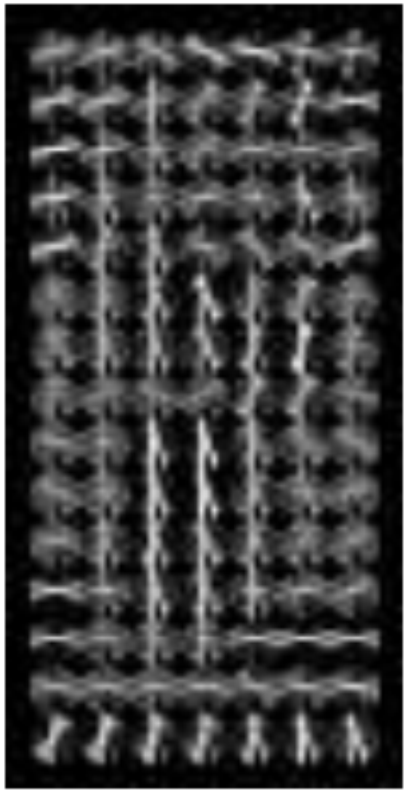
The problem

- Seminal paper for machine learning in CV:
  - Dalal and Triggs, CVPR 2005
  - Found MIT pedestrian test set too easy
  - Introduces new ‘INRIA’ dataset
  - Tiny datasets by today’s standards
    - MIT: 500 train, 200 test
    - New: 1239 train, 566 test

Hand-coded Features: HOG


Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non–person classification

- HOG = histogram of oriented gradients
- Very similar to SIFT
- Evaluated on regular (overlapping) grid
- Single scale
- 8x8 pixel cells
- 9 orientation bins



See also: https://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

Recap: Classification

Face Detection

Linear Classifiers

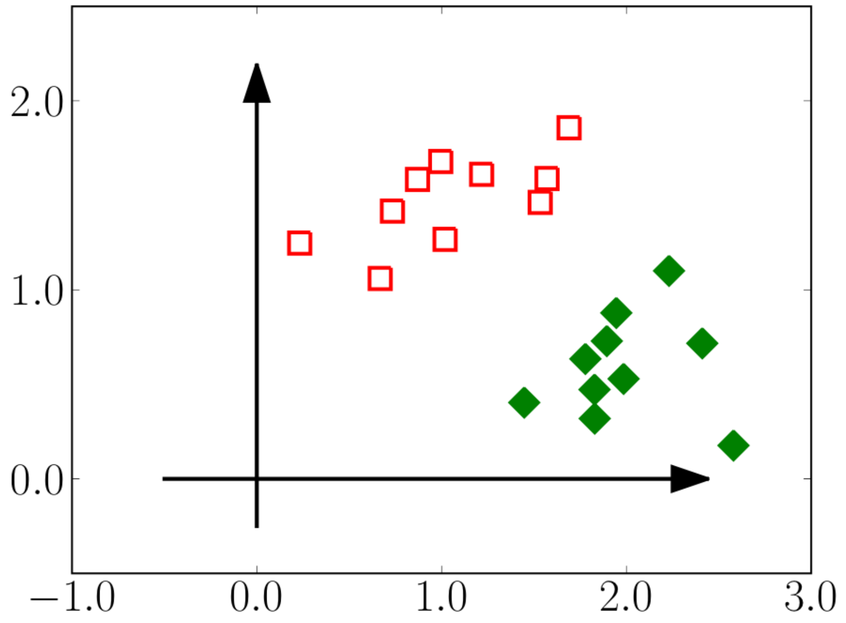
Boosting

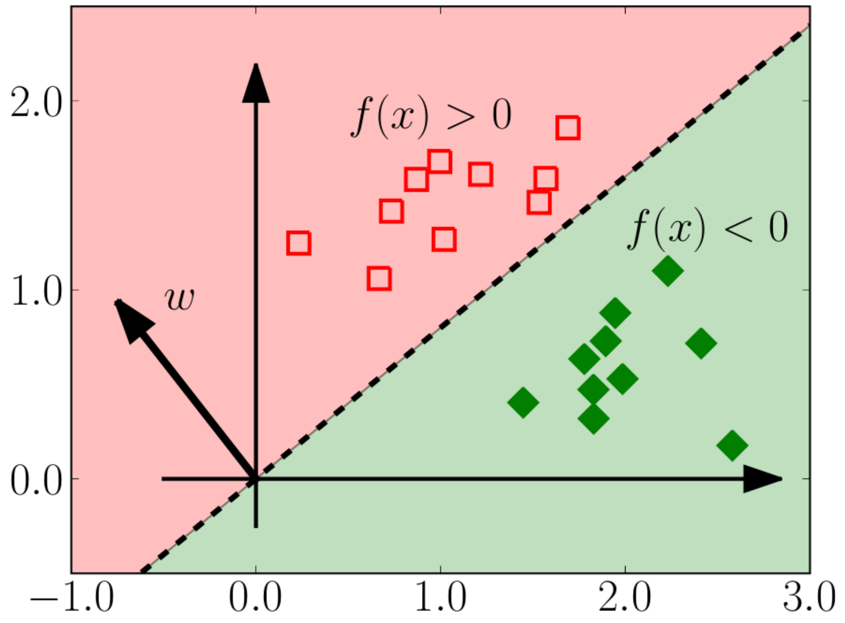
Viola-Jones

Pedestrian Detection

$\boxed{\text{Support Vector Machines}}$

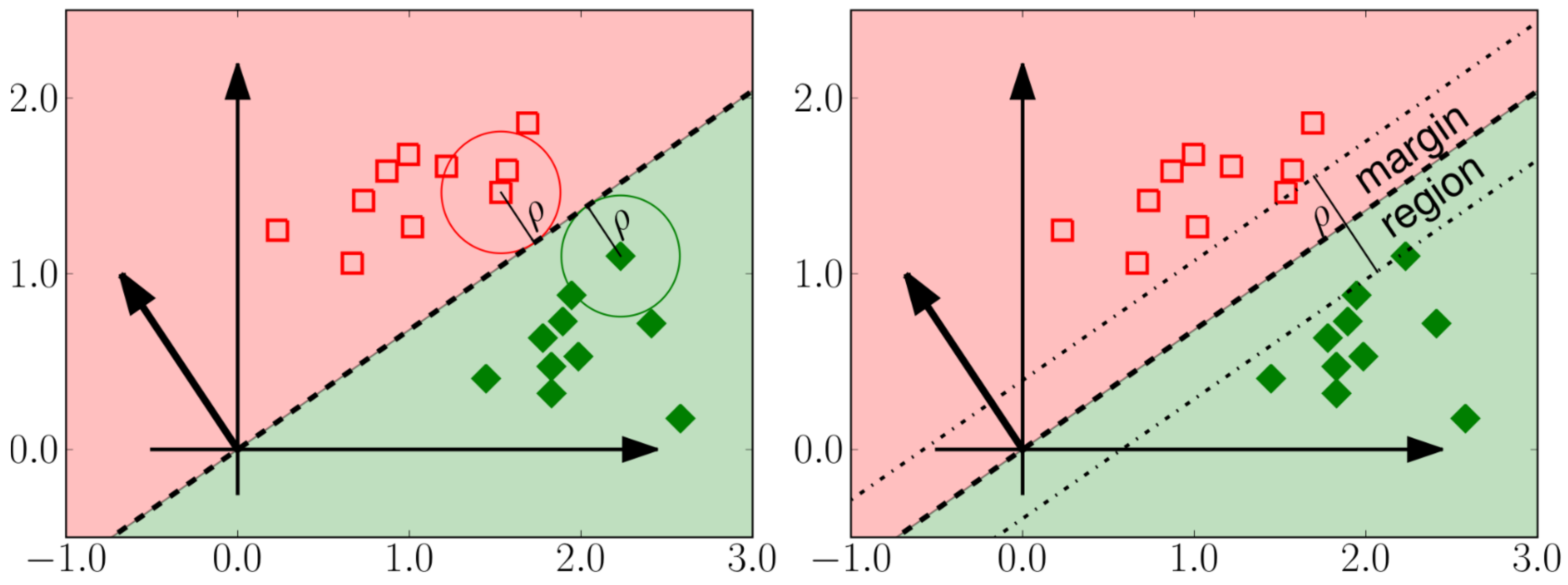
Classifier





- Two class example
- Linear hyperplane classifier, f(x) = <w, x>

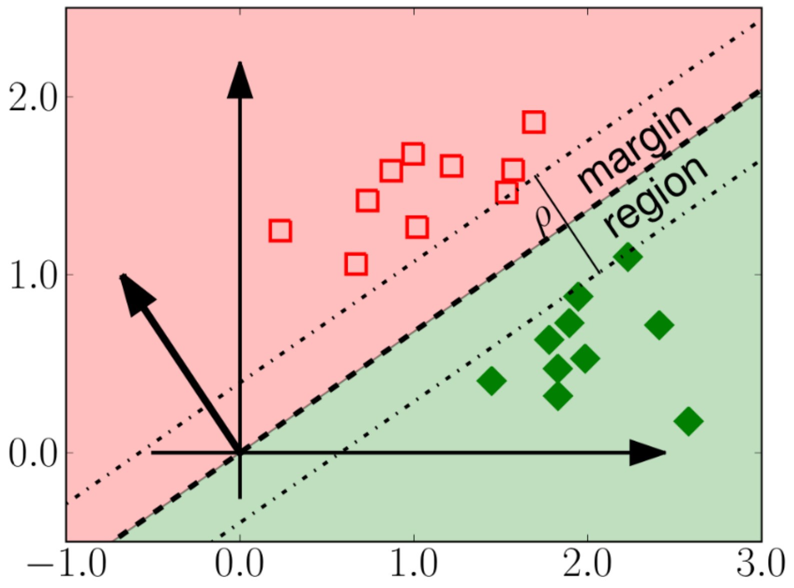
Margin



- Margin of an example
- Margin of a classifier

$$\rho = \min_{i=1,\ldots,n} \left| \langle \frac{w}{\|w\|}, x_i \rangle \right|$$

Learning SVMs



Intuitive maximization problem:

$$\boxed{\max_{w \in \mathbb{R}^d} \quad \rho}$$

subject to

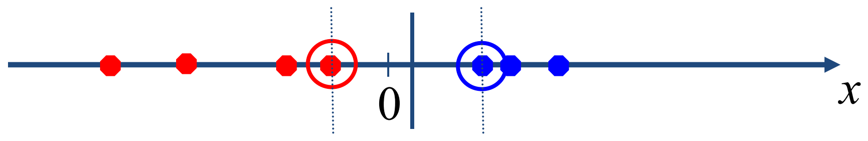
$$\rho = \min_{i=1,\ldots,n} \left| \langle \frac{w}{\|w\|}, x_i \rangle \right|$$

and

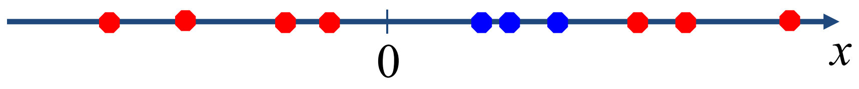
$$\operatorname{sign}\langle w, x_i \rangle = y_i, \qquad \text{for } i = 1, \ldots, n.$$

Nonlinear SVMs

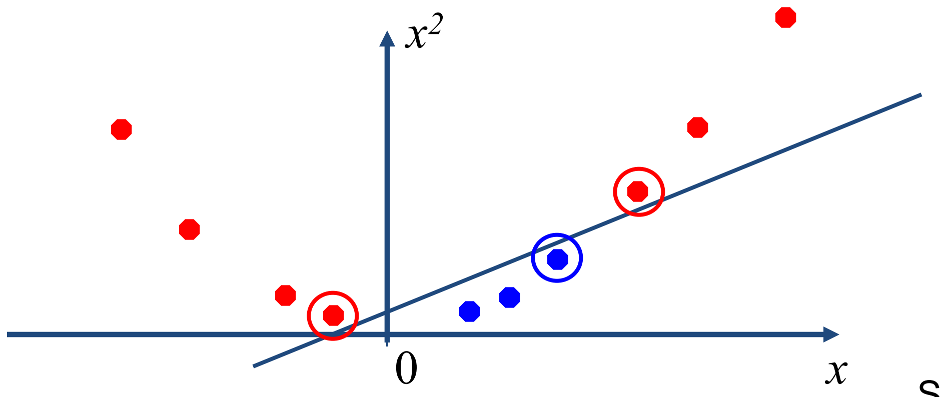
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?



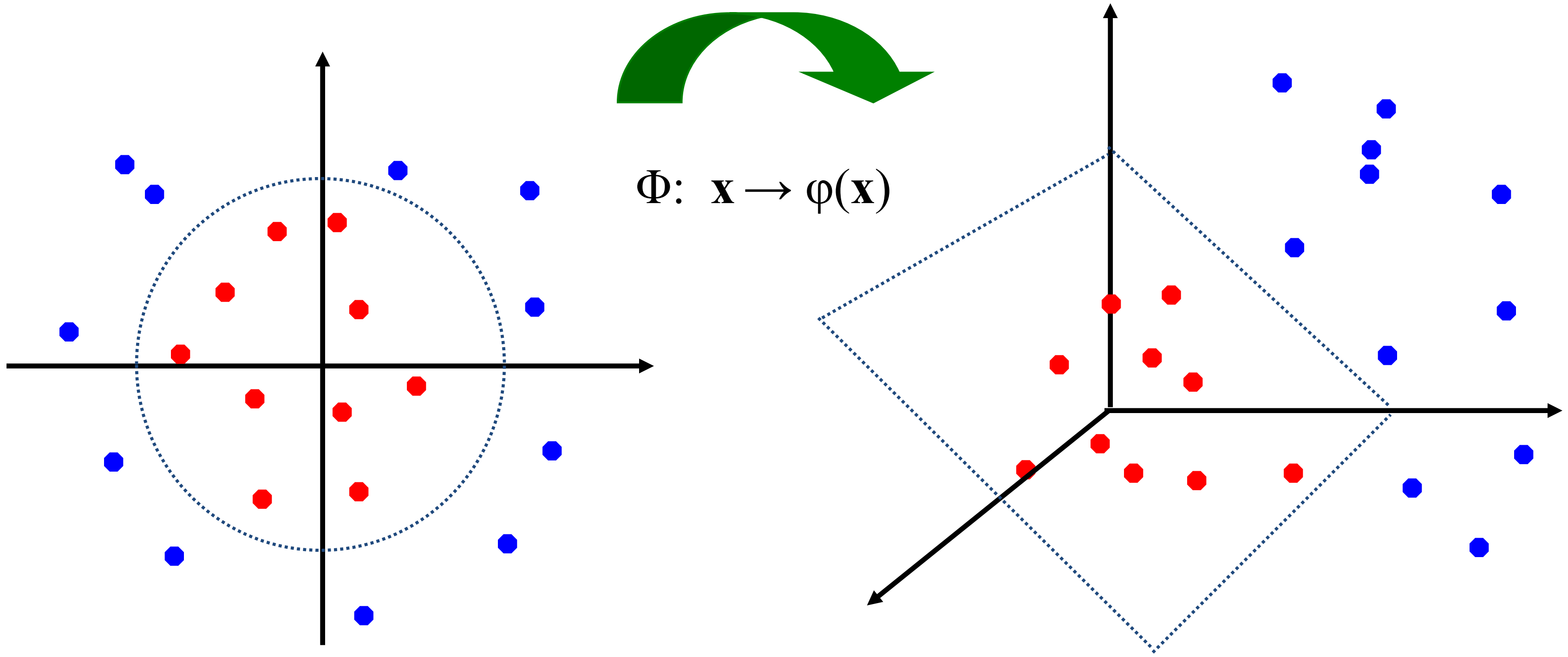
- We can map it to a higher-dimensional space:



Slide credit: Andrew Moore

Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Slide credit: Andrew Moore

Nonlinear SVMs

- *The kernel trick*: instead of explicitly computing the lifting transformation $\boldsymbol{\varphi}(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j)$$

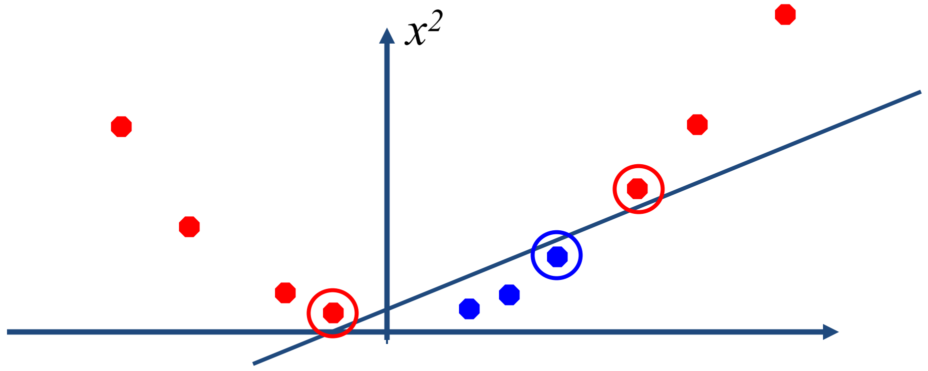
- This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i \varphi(\boldsymbol{x}_i) \cdot \varphi(\boldsymbol{x}) + b = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Nonlinear kernel: Example

- Consider the mapping $\varphi(x) = (x, x^2)$



$$\varphi(x) \cdot \varphi(y) = (x, x^2) \cdot (y, y^2) = xy + x^2 y^2$$

$$K(x, y) = xy + x^2 y^2$$

Kernels for bags of features

- Histogram intersection kernel:

$$I(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i))$$

- Generalized Gaussian kernel:

$$K(h_1, h_2) = \exp\left(-\frac{1}{A} D(h_1, h_2)^2\right)$$

- *D* can be (inverse) L1 distance, Euclidean distance, χ^2 distance, etc.

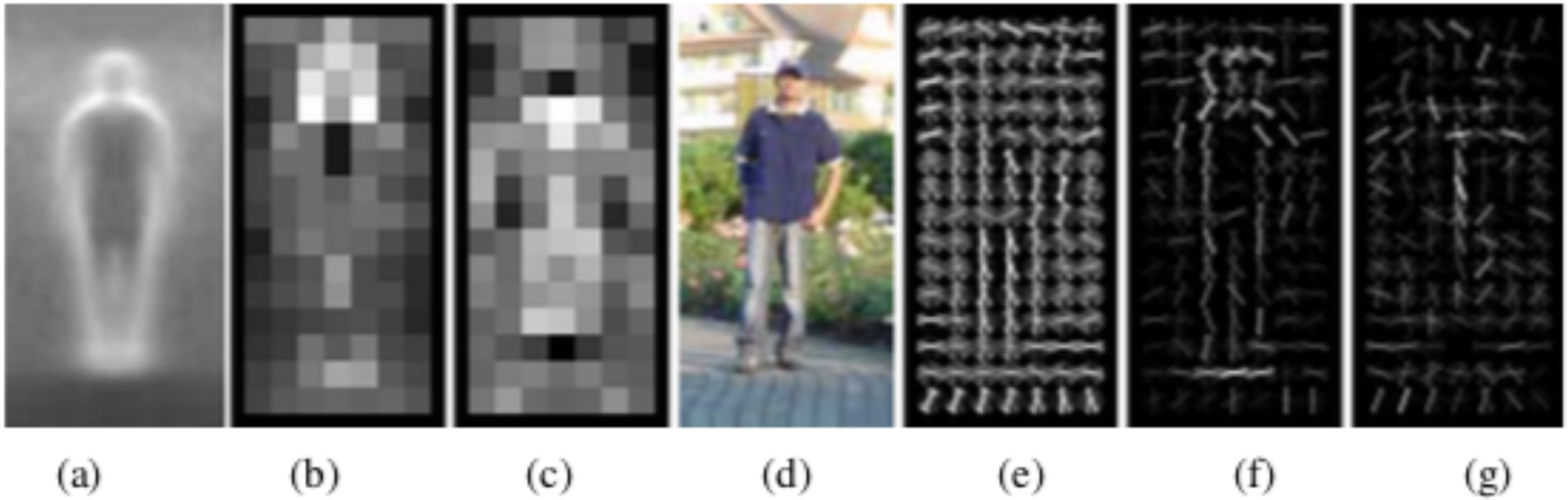
J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, Local Features and Kernels for Classifcation of Texture and Object Categories: A Comprehensive Study, IJCV 2007

Summary: SVMs for image classification

1. Pick an image representation (in our case, HOG)
2. Pick a kernel function for that representation
3. Compute the matrix of kernel values between every pair of training examples
4. Feed the kernel matrix into your favorite SVM solver to obtain support vectors and weights
5. At test time: compute kernel values for your test example and each support vector, and combine them with the learned weights to get the value of the decision function

Slide credit: L. Lazebnik

Pedestrian Detection Example



- (f) weighted by the positive SVM weights
- (g) weighted by the negative SVM weights

Figure from Dalal and Triggs 2005

SVMs: Pros and cons

- Pros
  - Many publicly available SVM packages: http://www.kernel-machines.org/software
  - Kernel-based framework is very powerful, flexible
  - SVMs work very well in practice, even with very small training sample sizes

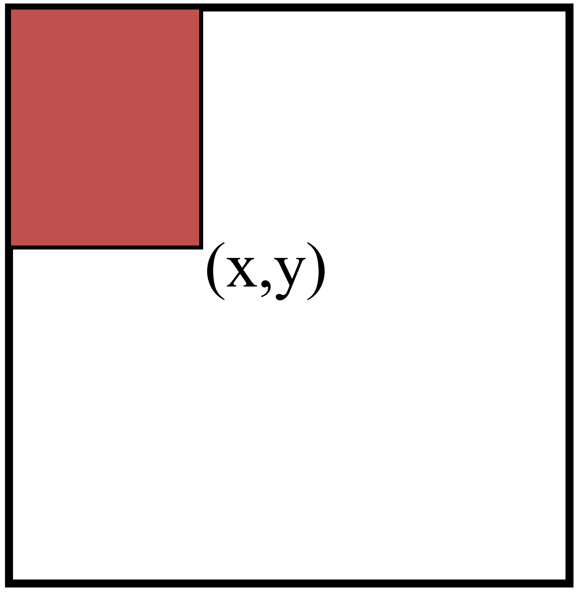
- Cons
  - No "direct" multi-class SVM, must combine two-class SVMs
  - Computation, memory
    - During training time, must compute matrix of kernel values for every pair of examples
    - Learning can take a very long time for large-scale problems

Bonus Slides

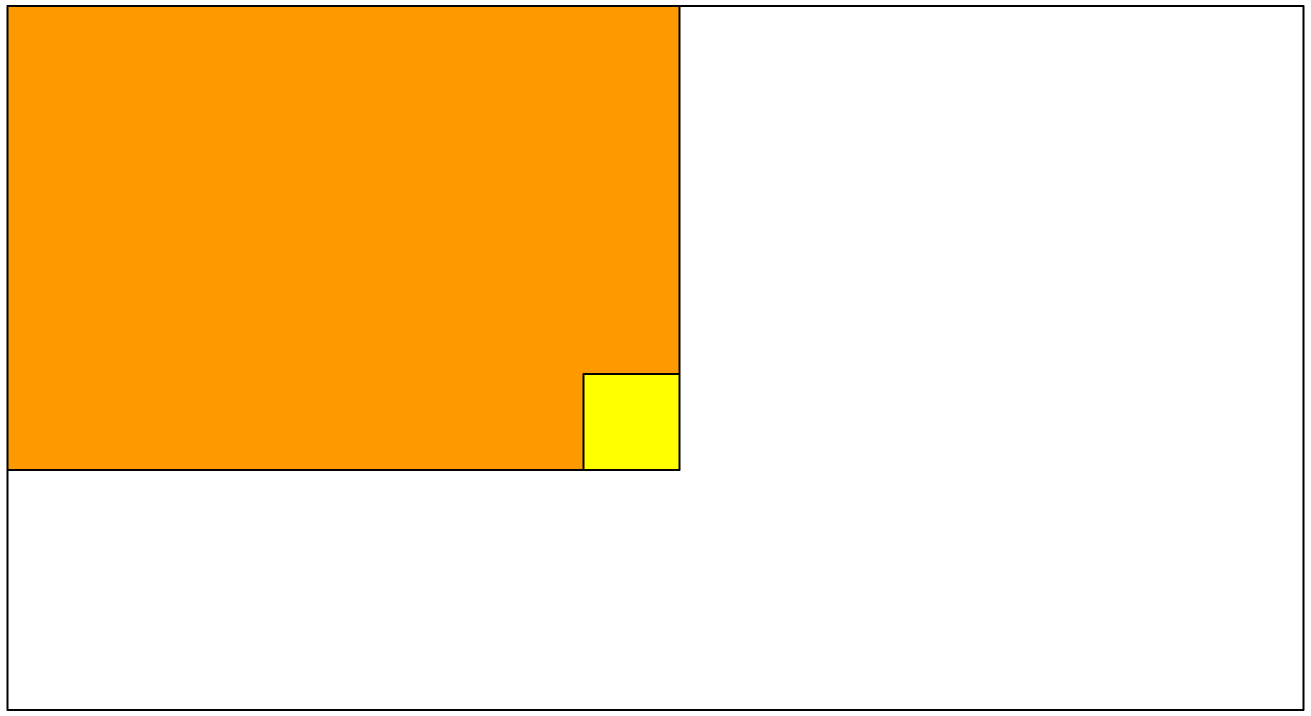
Integral Images

Fast computation with integral images

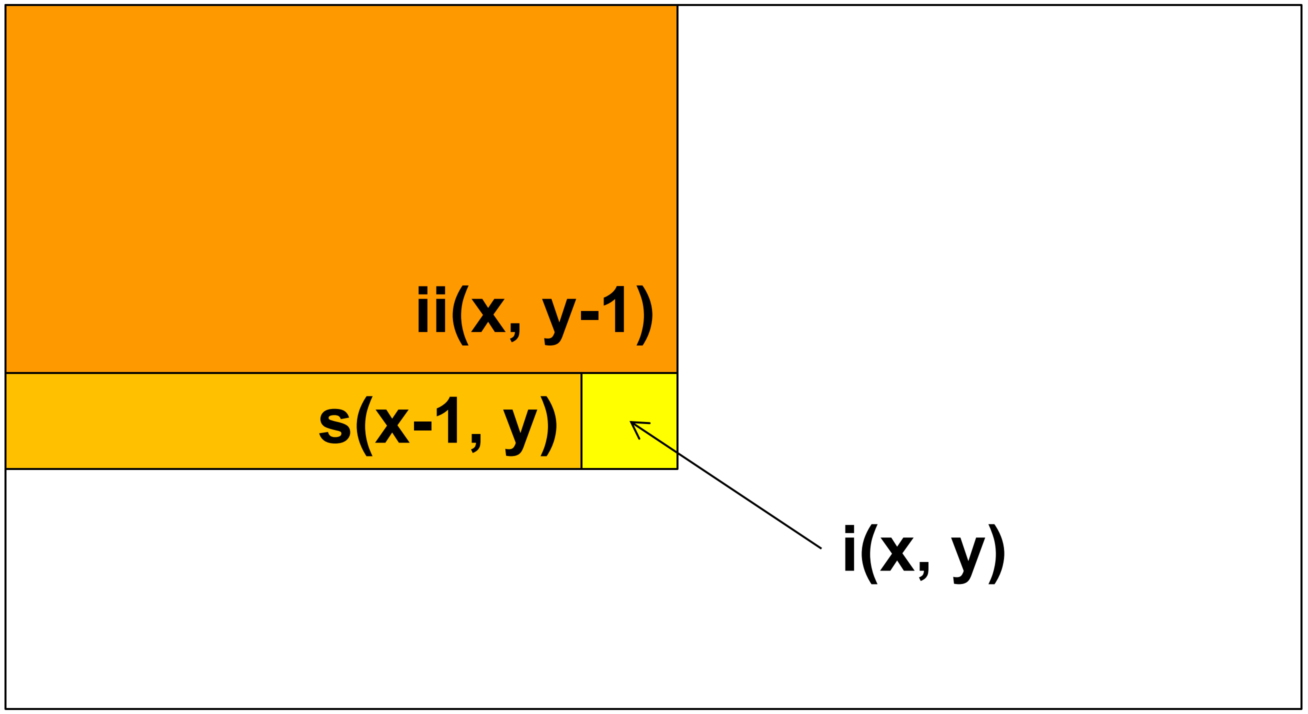
- The *integral image* computes a value at each pixel (*x*,*y*) that is the sum of the pixel values above and to the left of (*x*,*y*), inclusive
- This can quickly be computed in one pass through the image



Computing the integral image



Computing the integral image



- Cumulative row sum: $s(x, y) = s(x-1, y) + i(x, y)$
- Integral image: $ii(x, y) = ii(x, y-1) + s(x, y)$

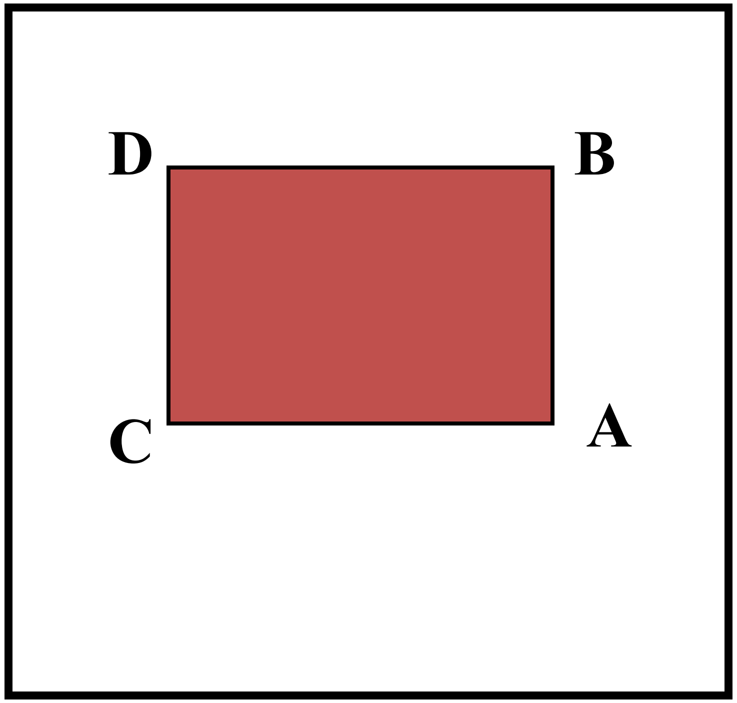
MATLAB: `ii = cumsum(cumsum(double(i)), 2);`

Computing sum within a rectangle

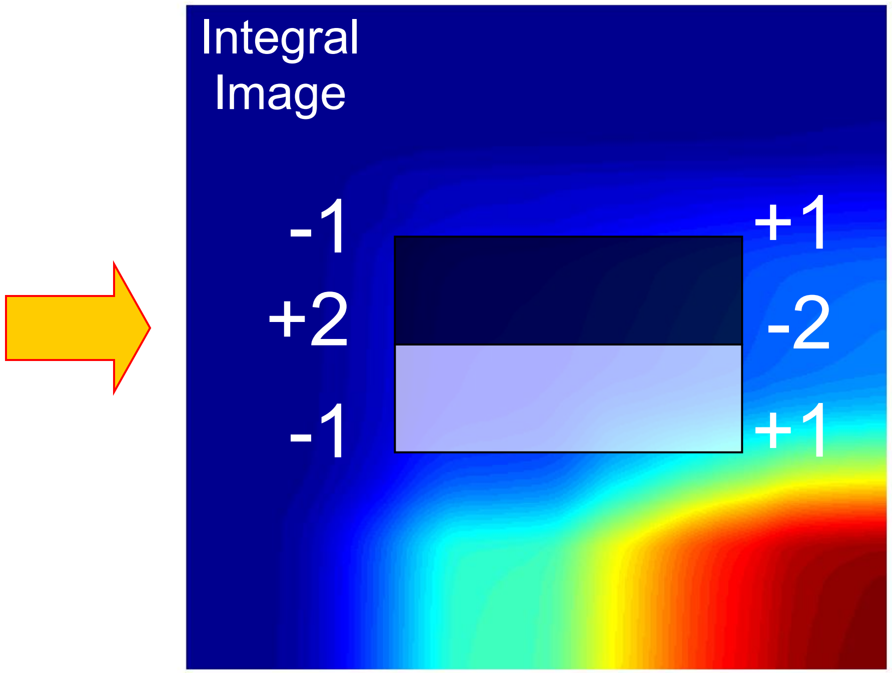
- Let A,B,C,D be the values of the integral image at the corners of a rectangle
- Then the sum of original image values within the rectangle can be computed as:

  sum = A – B – C + D

- Only 3 additions are required for any size of rectangle!



Computing a rectangle feature


Integral Image
-1
+1
+2
-2
-1
+1