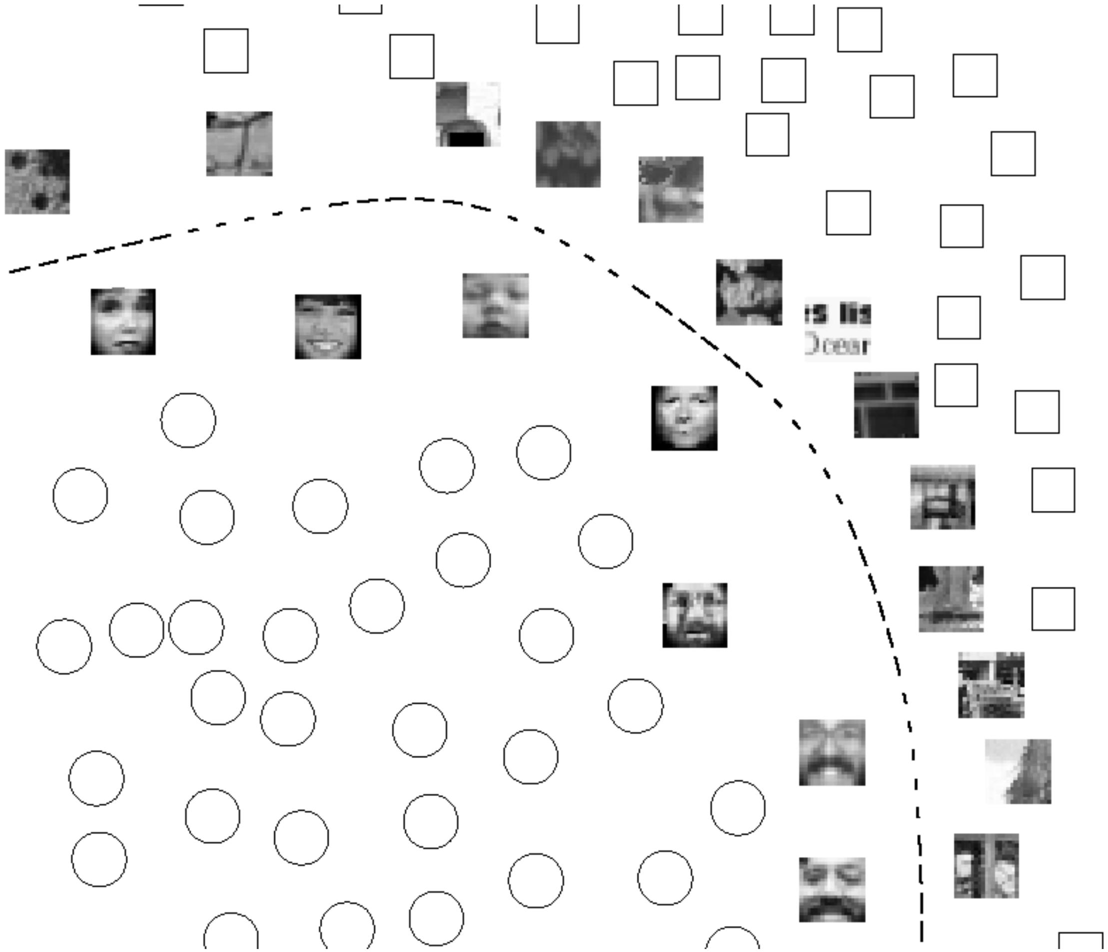
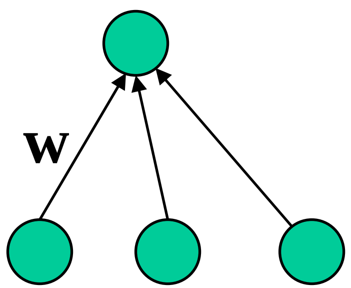
Lecture 10

Pattern Recognition & Learning II



Flashback: Sigmoidal Networks

Output $v = g(\mathbf{w}^T\mathbf{u}) = g(\sum_i w_i u_i)$



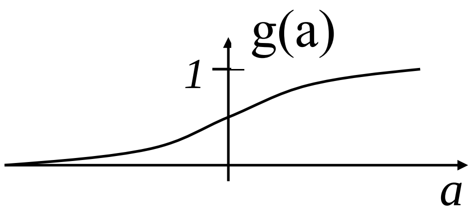
Input nodes

$\mathbf{u} = (u_1 \quad u_2 \quad u_3)^T$

The most commonly used differentiable function:

Sigmoid function:

$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



Non-linear "squashing" function: Squashes input to be between 0 and 1. The parameter β controls the slope.

How do we learn the weights?

Given training examples ($\mathbf{u}^m$,d^m) (m = 1, …, N), define a sum of squared output errors function (also called a cost function or "energy" function)

$$E(\mathbf{w}) = \frac{1}{2}\sum_m (d^m - v^m)^2$$

where $v^m = g(\mathbf{w}^T \mathbf{u}^m)$

How do we learn the weights?

We would like to choose **w** that minimize E – how?

$$E(\mathbf{w}) = \frac{1}{2}\sum_m (d^m - v^m)^2$$

where $v^m = g(\mathbf{w}^T \mathbf{u}^m)$

Gradient-Descent Learning ("Hill-Climbing")

Idea: Change **w** in proportion to –dE/d**w**

(why does this work?)

$$\mathbf{w} \leftarrow \mathbf{w} - \varepsilon \frac{dE}{d\mathbf{w}}$$

$$\frac{dE}{d\mathbf{w}} = -\sum_m (d^m - v^m) \frac{dv^m}{d\mathbf{w}} = -\sum_m (d^m - v^m) g'(\mathbf{w}^T \mathbf{u}^m) \mathbf{u}^m$$

Derivative of sigmoid

"Stochastic" Gradient Descent

What if the inputs only arrive one-by-one?

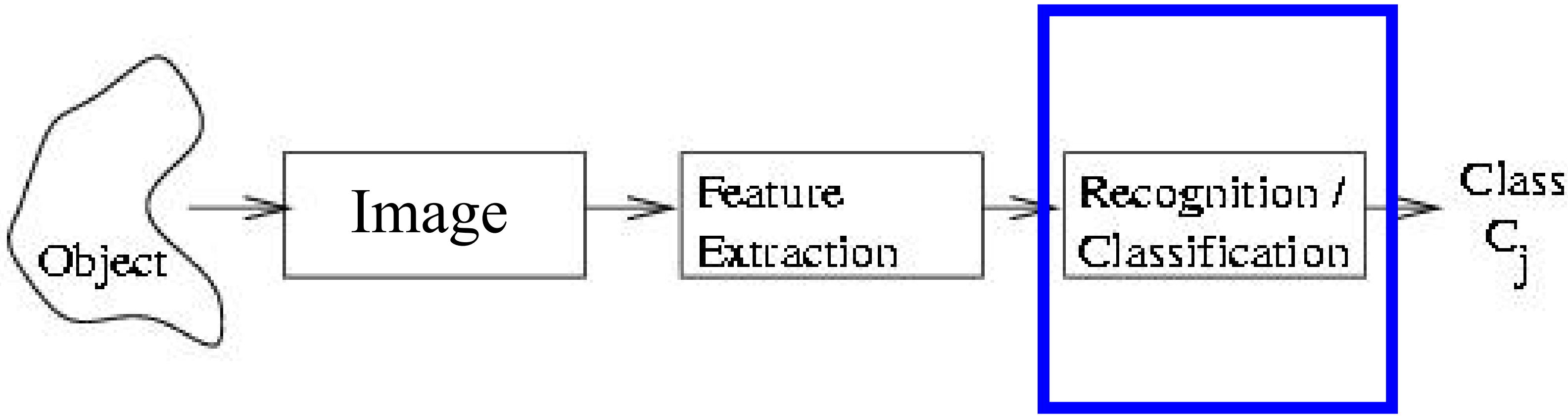
Stochastic gradient descent approximates sum over all inputs with an "on-line" running sum:

$$\mathbf{w} \rightarrow \mathbf{w} - \varepsilon \frac{dE_1}{d\mathbf{w}}$$

$$\frac{dE_1}{d\mathbf{w}} = -\underbrace{(d^m - v^m)}_{\text{delta = error}} g'(\mathbf{w}^T \mathbf{u}^m)\mathbf{u}^m$$

Also known as the "delta rule" or "LMS (least mean square) rule"

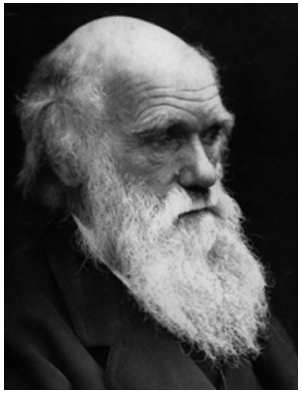
Recall from Last Time: Classification Problem

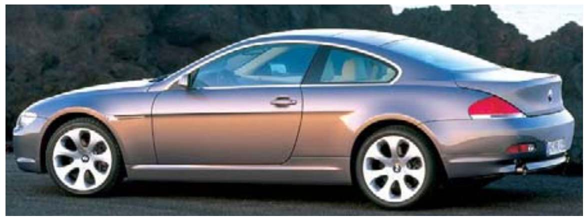


Classification problem: Given a training dataset of (input image, output class) pairs, build a classifier that outputs a class for any new input image

Example: Face Detection

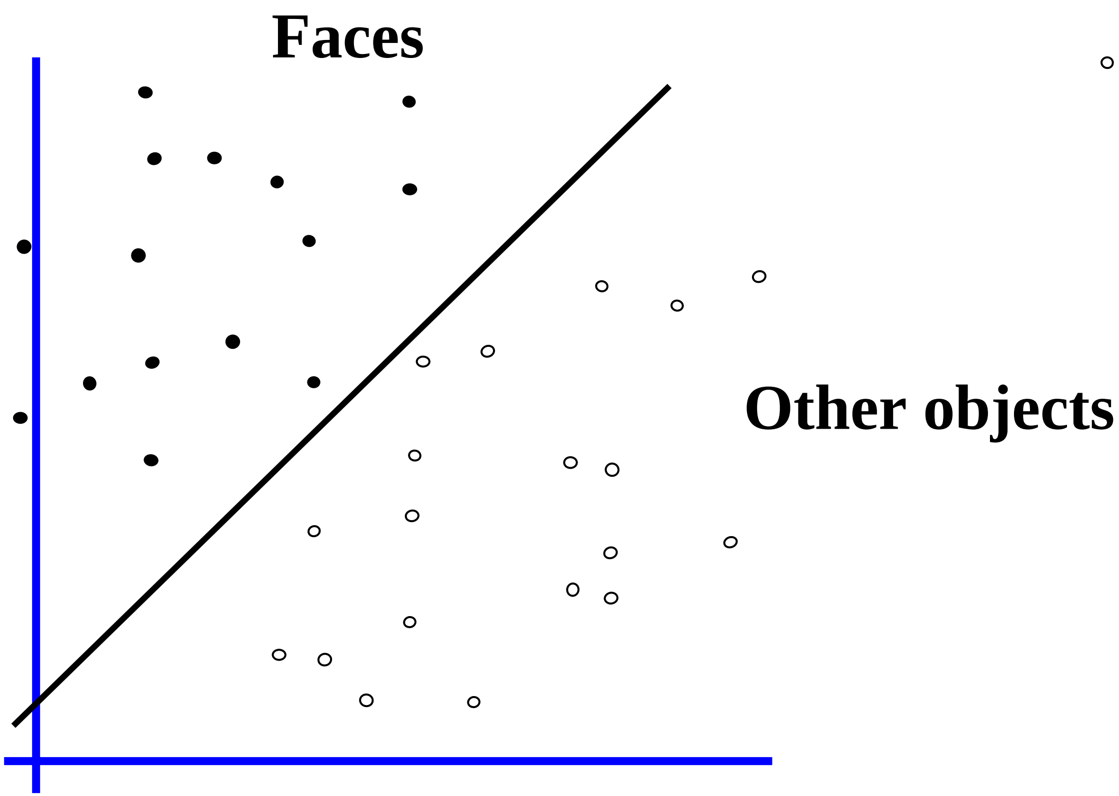
How do we build a classifier to distinguish between faces and other objects?





The Classification Problem

- • denotes +1 (faces)
- ◦ denotes -1 (other)



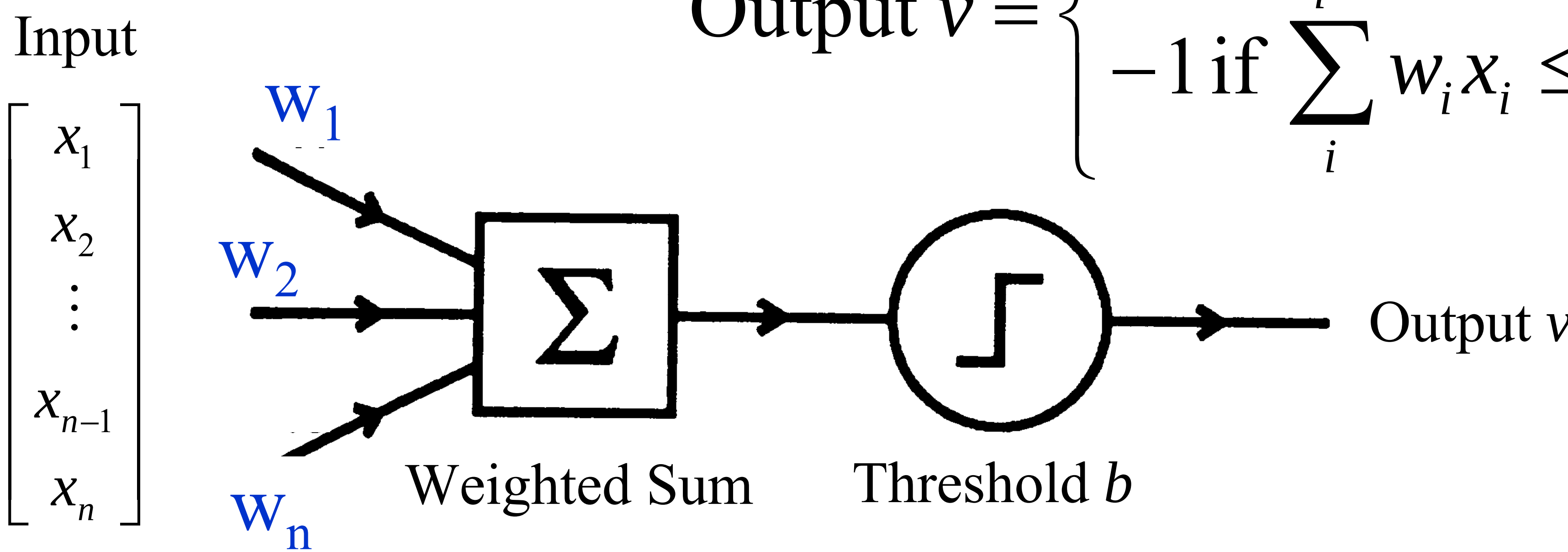
Idea: Find a separating hyperplane (line in this case)

Recall from Last Time: Perceptron

Artificial "neuron":

- Input vector **x** and output value *v*
- Weight vector **w**
- Threshold *b*

$$\text{Output } v = \begin{cases} +1 \text{ if } \sum_i w_i x_i > b \\ -1 \text{ if } \sum_i w_i x_i \le b \end{cases}$$

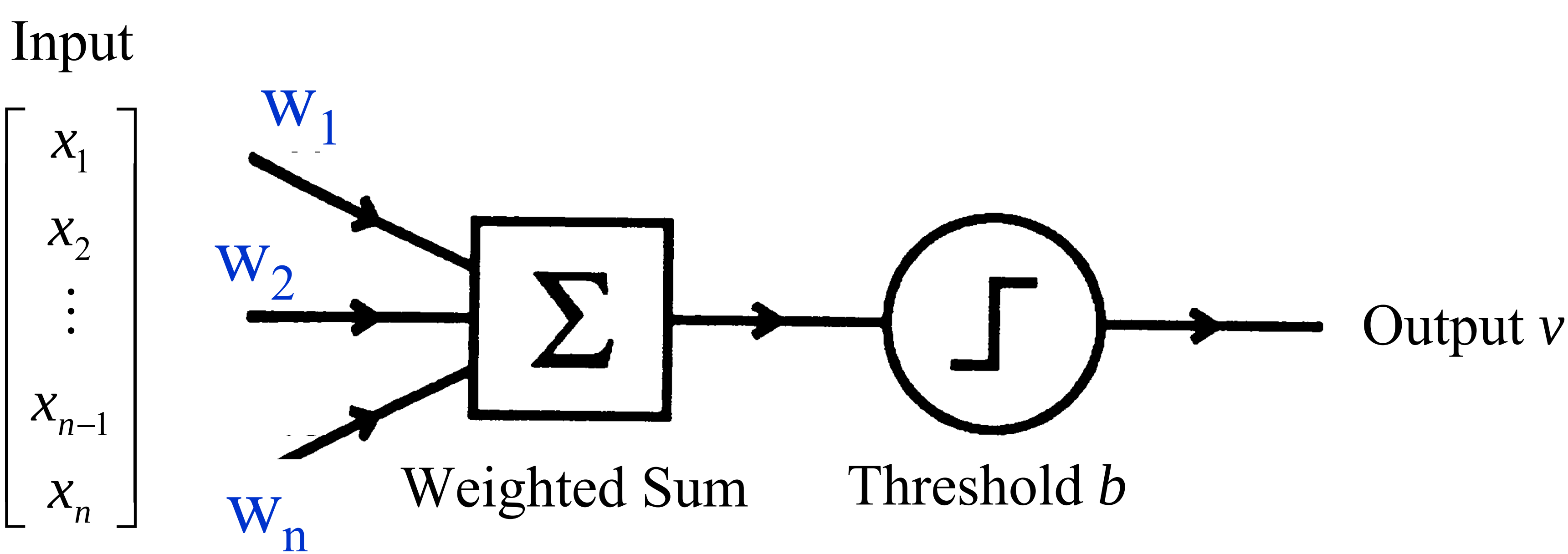


Perceptron

Equivalently:

$$v = sign(\sum_i w_i x_i - b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$$

where $sign(z)$ = +1 if z > 0 and -1 if z ≤ 0



Perceptrons and Classification

- Weighted sum forms a *linear hyperplane*

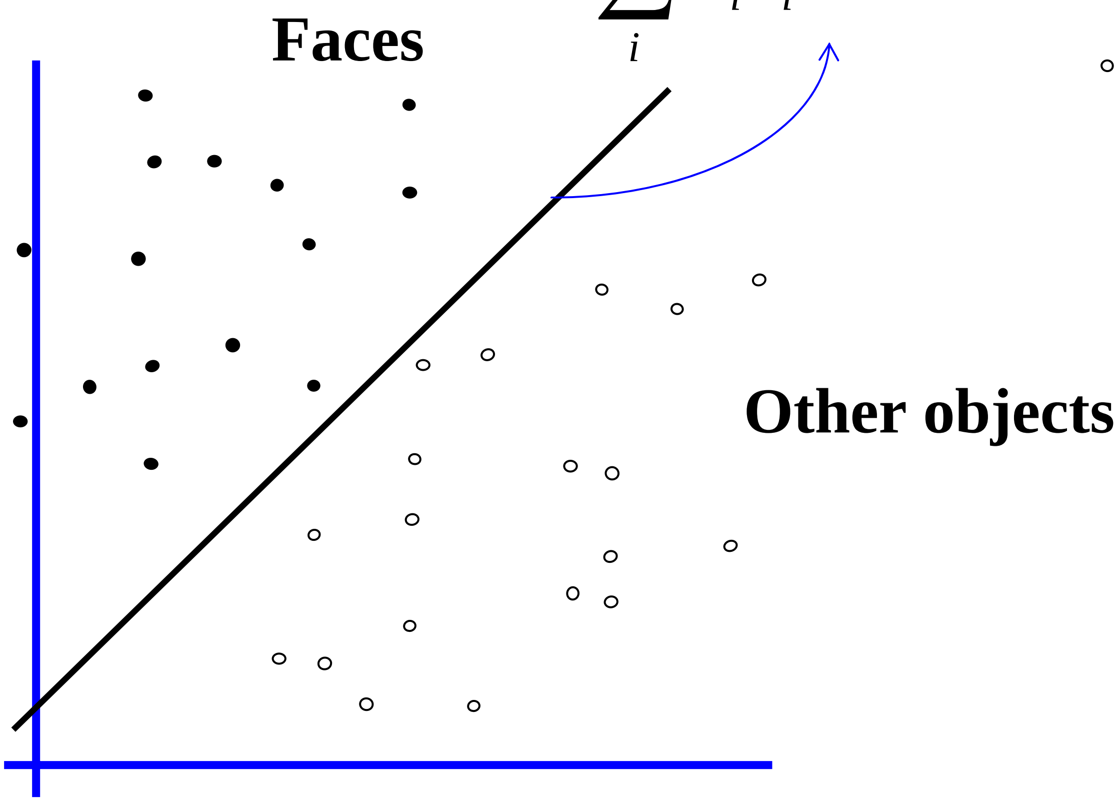
$$\sum_i w_i x_i - b = 0$$

- Due to threshold function, everything *on one side* of this hyperplane is labeled as class 1 (output = +1) and everything *on other side* is labeled as class 2 (output = -1)

Separating Hyperplane

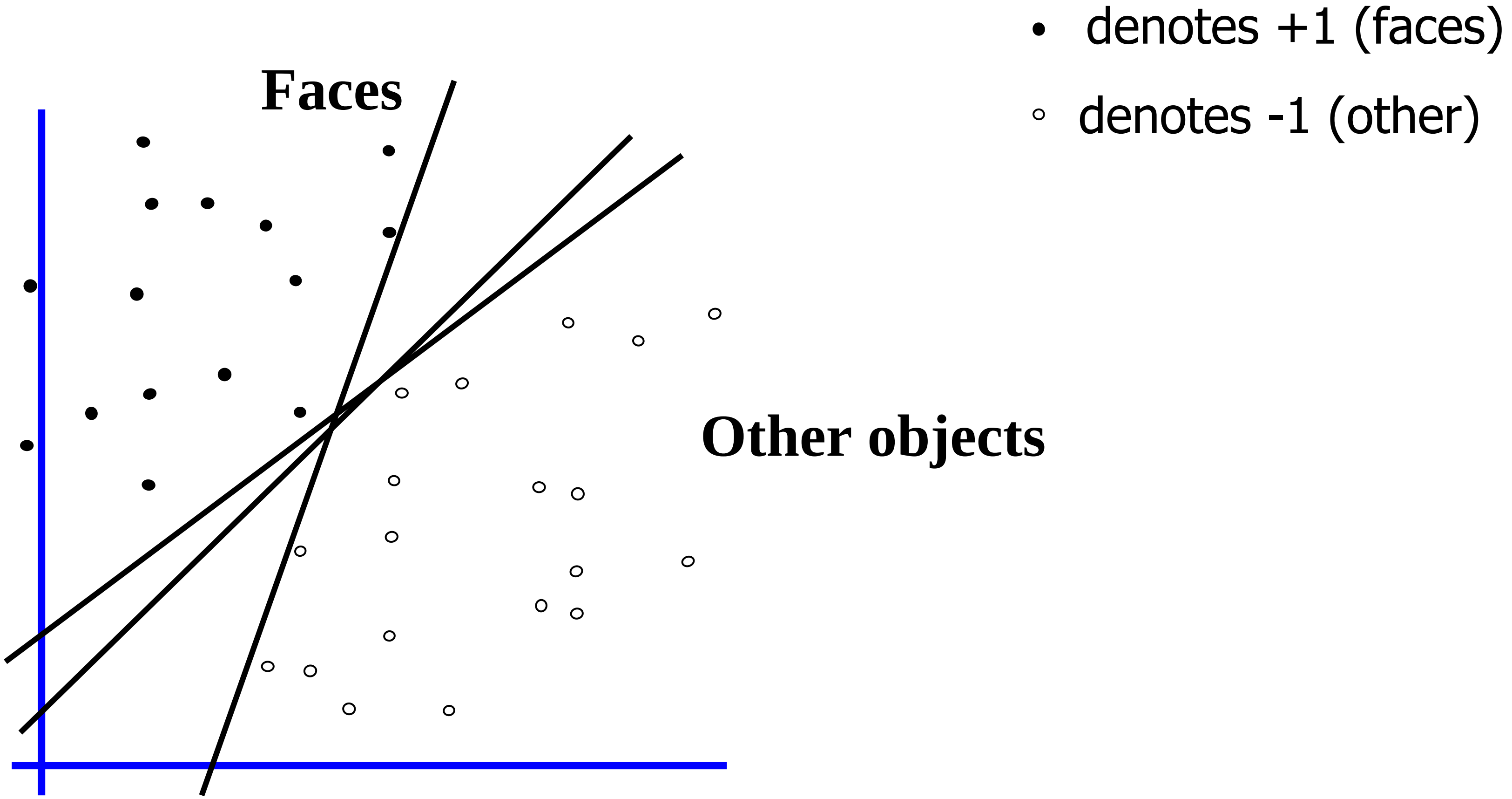
$$\sum_i w_i x_i - b = 0$$

- • denotes +1 (faces)
- ○ denotes -1 (other)



Need to choose **w** and *b* based on training data

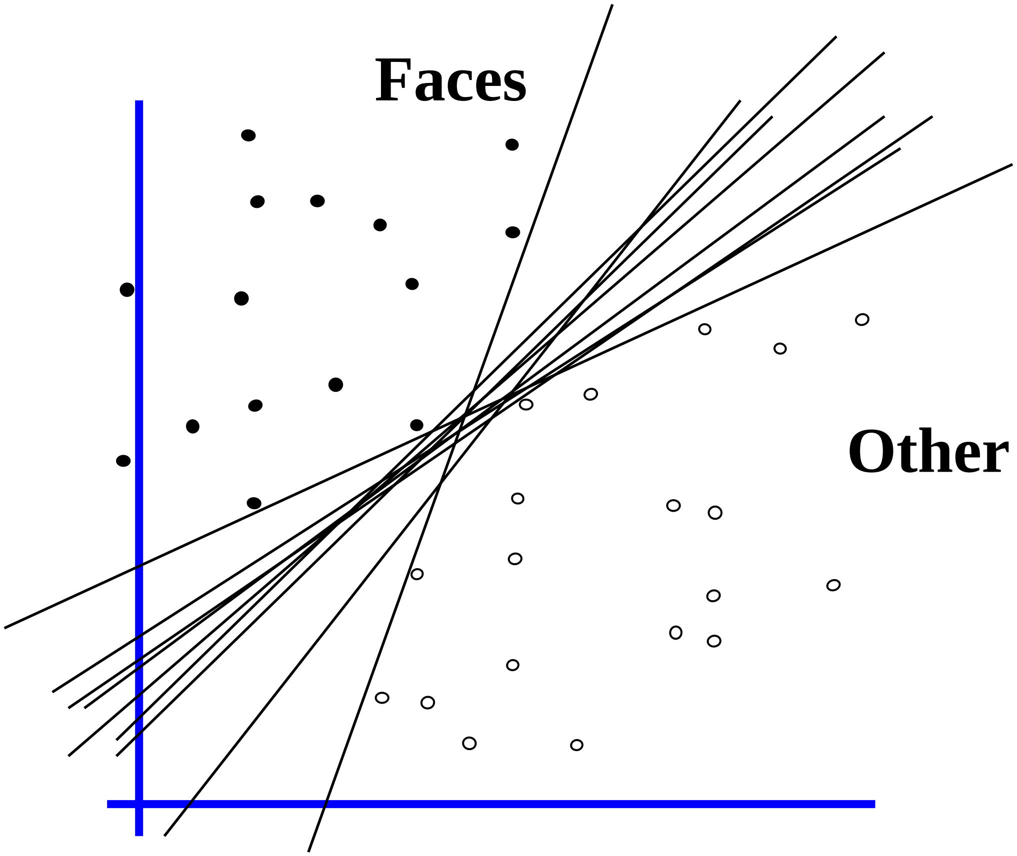
Separating Hyperplanes



Different choices of **w** and *b* give different hyperplanes

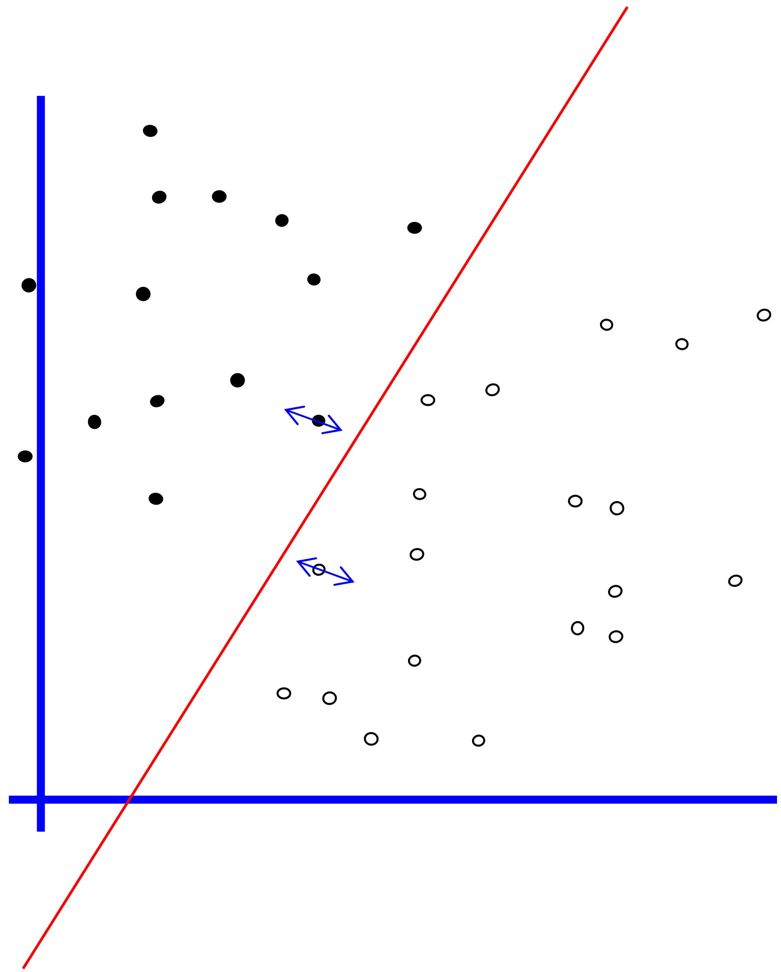
(This and next few slides adapted from Andrew Moore's)

Which hyperplane is best?



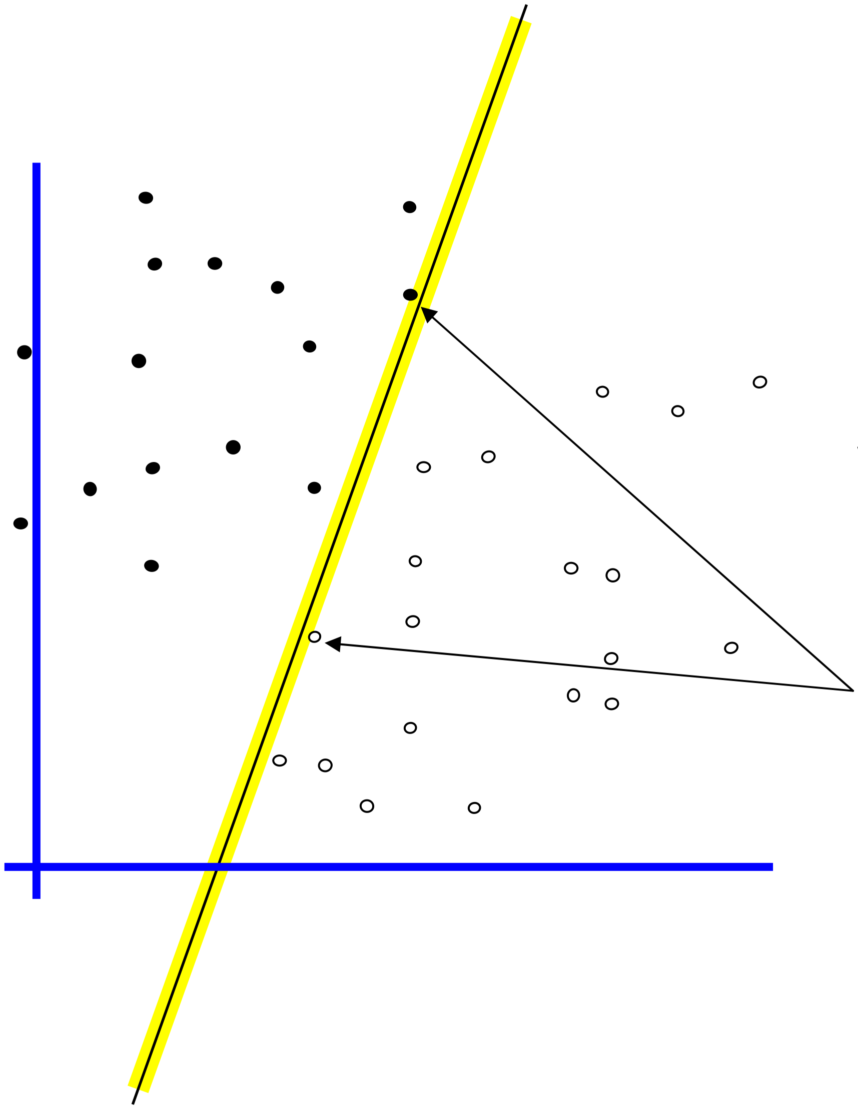
- • denotes +1 (faces)
- ∘ denotes -1 (other)

How about the one right in the middle?



Intuitively, this boundary seems good because it is robust to minor perturbations of data points near the boundary (output does not switch from +1 to -1)

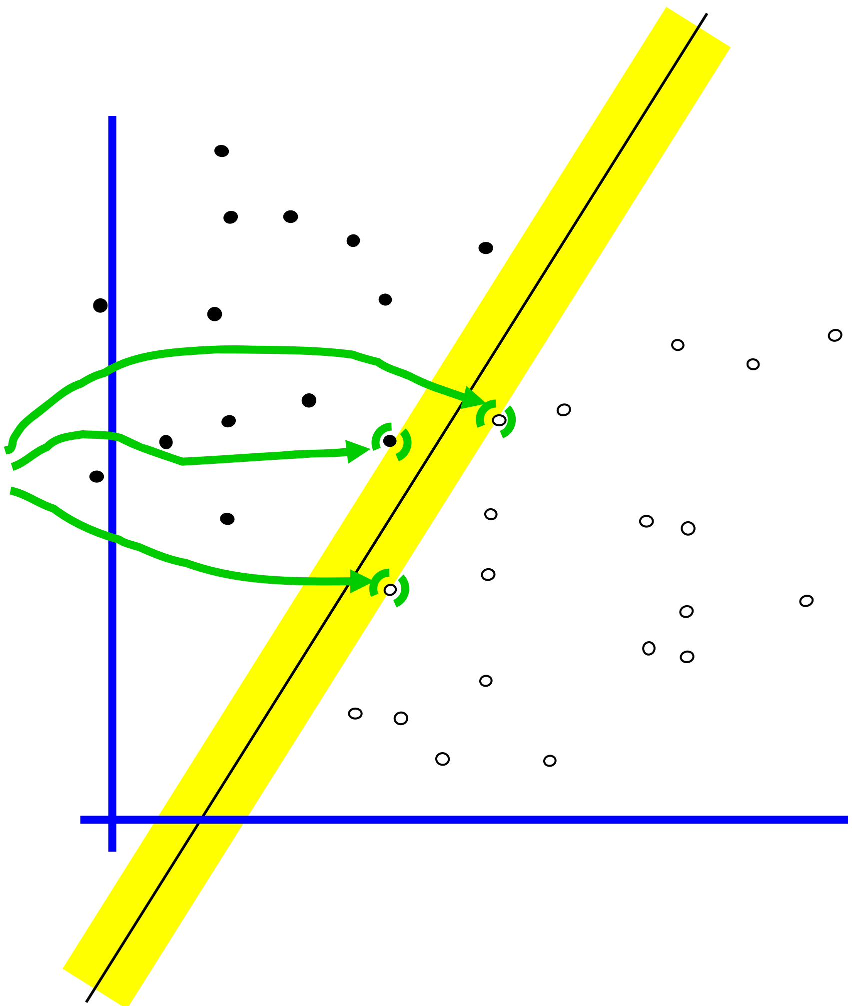
Margin



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

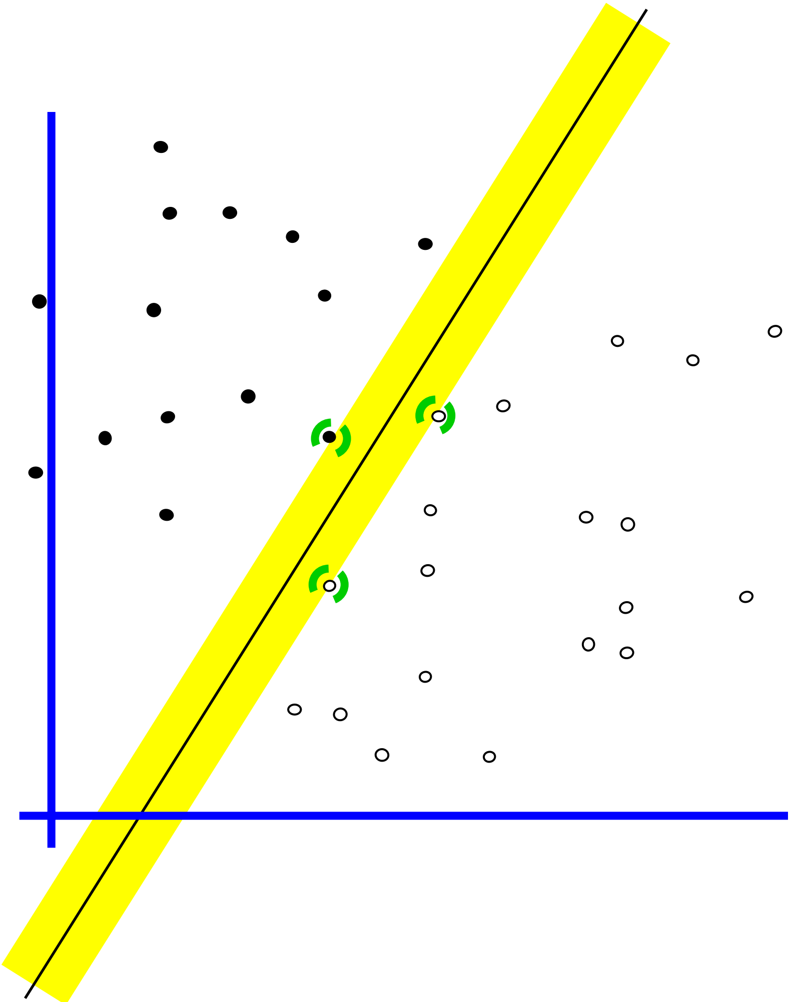
Maximum Margin and Support Vector Machine

Support Vectors are those datapoints that the margin pushes up against



The maximum margin classifier is called a Support Vector Machine (in this case, a Linear SVM or LSVM)

Why Maximum Margin?



- Robust to small perturbations of data points near boundary
- There exists theory showing this is best for generalization to new points (see online tutorial on class webpage)
- Empirically works great

Support Vector Machines

Suppose the training data points $(\mathbf{x}_i, y_i)$ satisfy :

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 \text{ for } y_i = +1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \text{ for } y_i = -1$$

This can be rewritten as

$$y_i\left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \geq +1$$

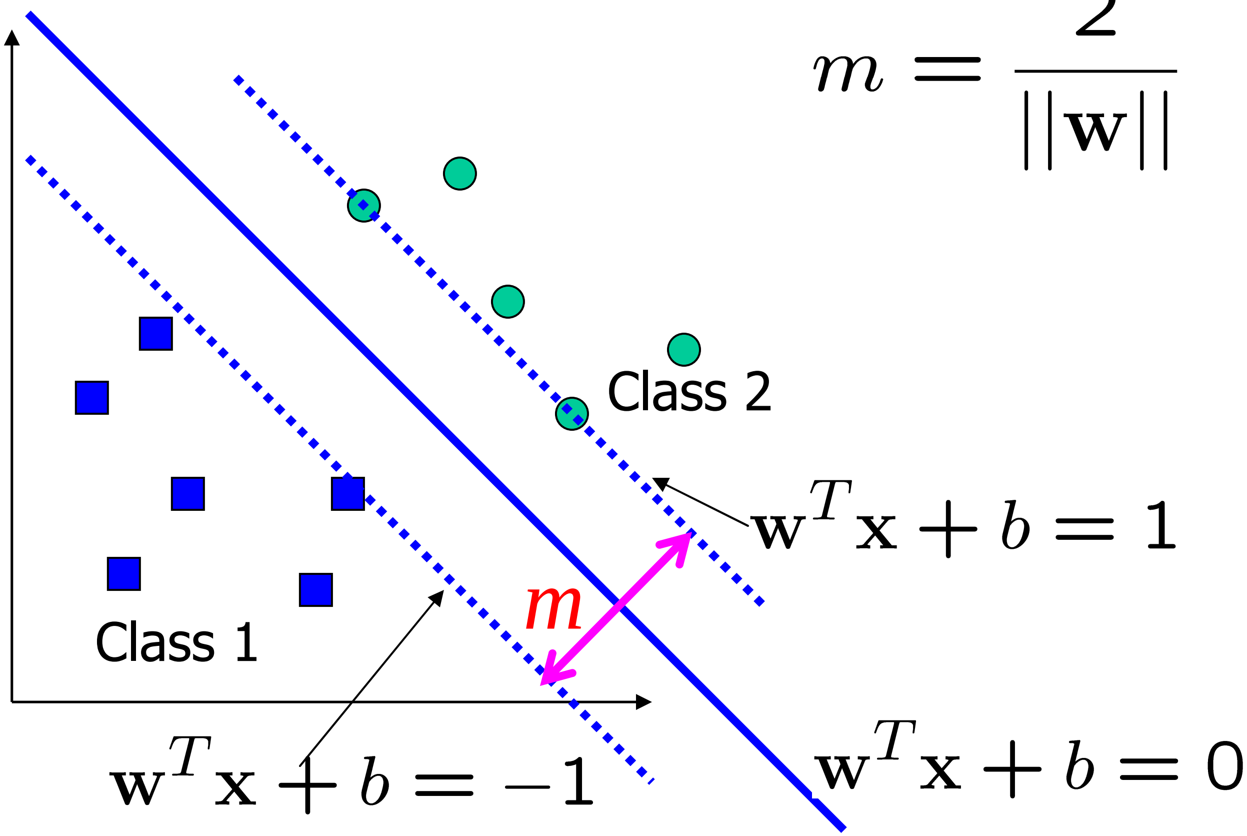
We can always do this by rescaling **w** and *b*, without affecting the separating hyperplane:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Estimating the Margin

The margin is given by (see Burges tutorial online):

$$m = \frac{2}{||\mathbf{w}||}$$



Margin can be calculated based on expression for distance from a point to a line, see, e.g., http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html

Learning the Maximum Margin Classifier

Want to maximize margin:

$$2/\|\mathbf{w}\| \text{ subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq +1, \forall i$$

Equivalent to finding **w** and *b* that minimize:

$$\frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq +1, \forall i$$

Constrained optimization problem that can be solved using Lagrange multiplier method

Learning the Maximum Margin Classifier

Using Lagrange formulation and Lagrangian multipliers α_i, we get (see Burges tutorial online):

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

where the α_i are obtained by maximizing:

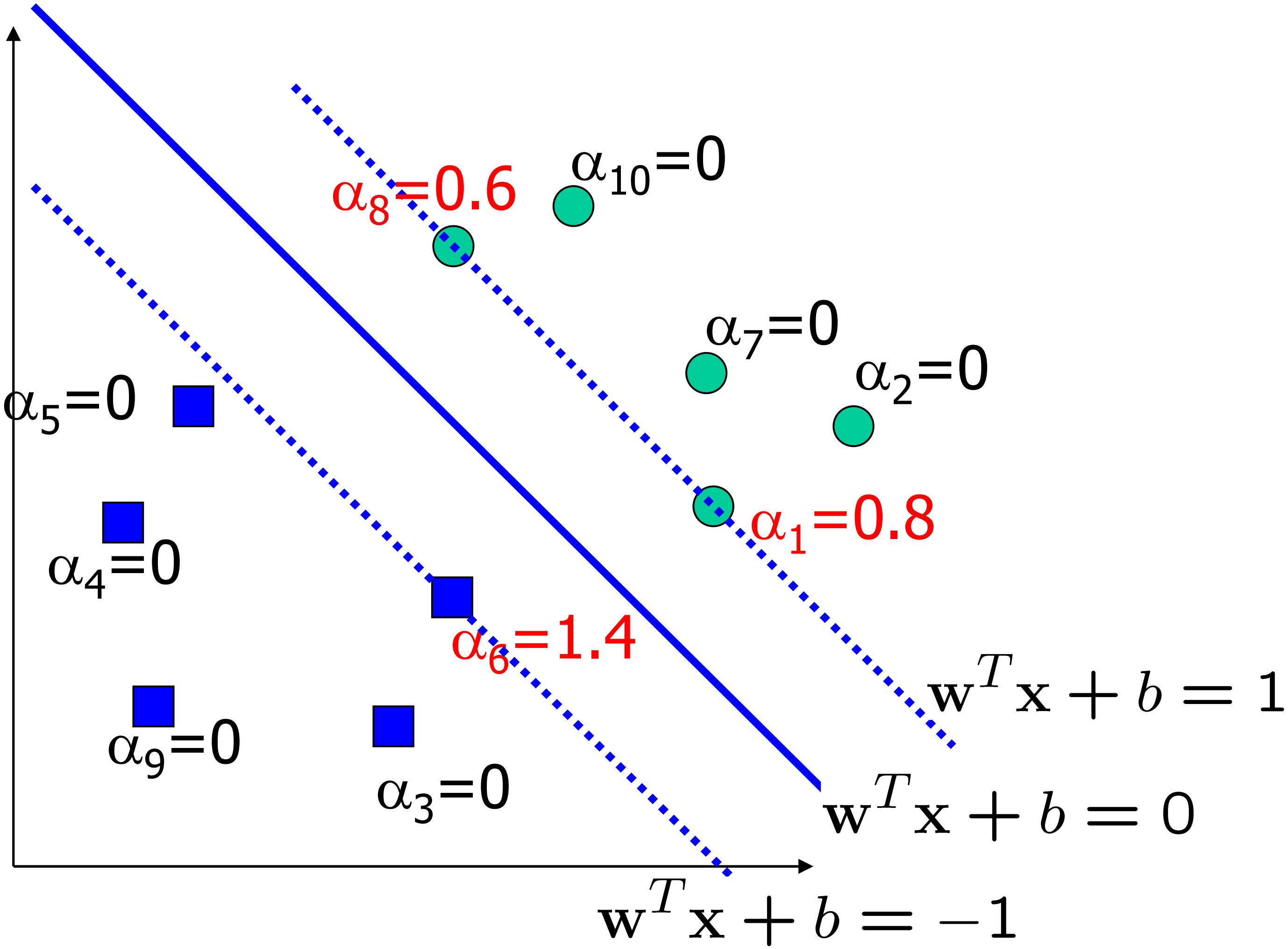
$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to } \alpha_i \geq 0 \text{ and } \sum_i \alpha_i y_i = 0$$

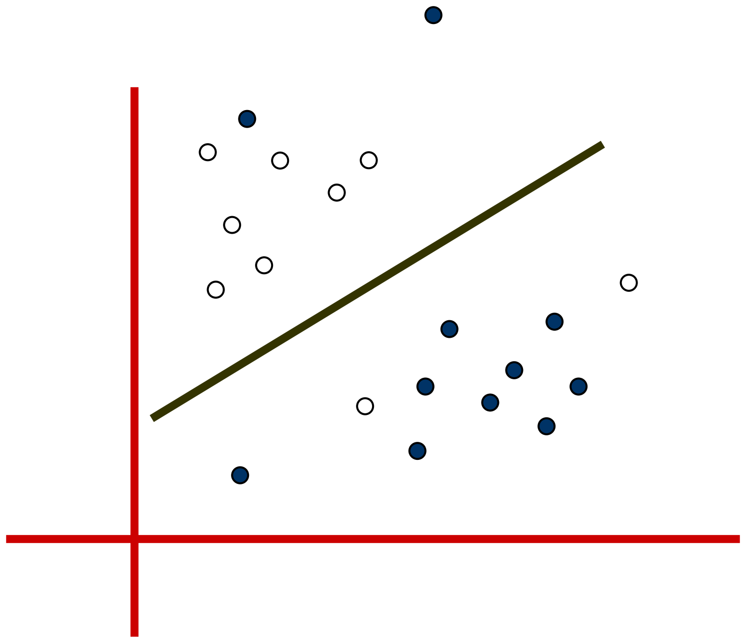
This is a quadratic programming (QP) problem
 - A global maximum can always be found

Geometrical Interpretation

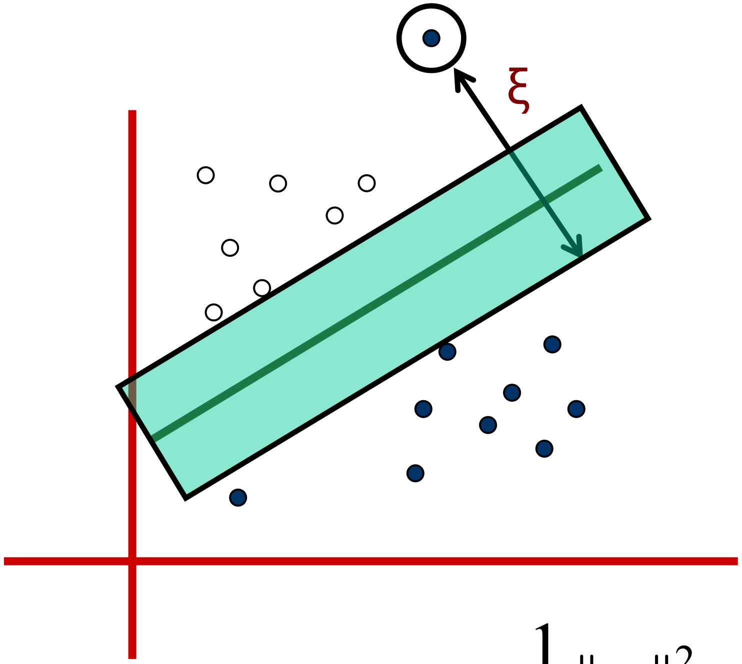
$\mathbf{x}_i$ with non-zero α_i are called support vectors



What if data is not linearly separable?



Approach 1: Soft Margin SVMs



Allow *errors* ξ_i (deviations from margin)

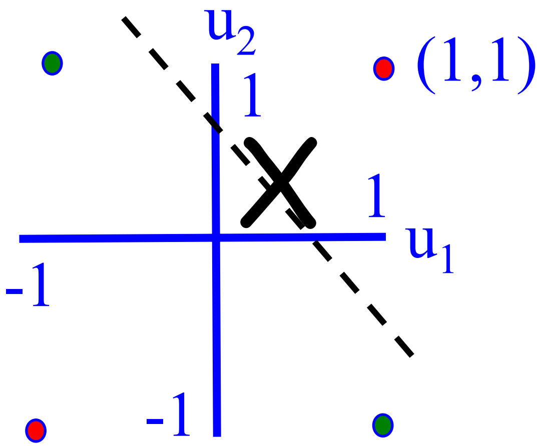
Trade off margin with errors.

Minimize: $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$ subject to :

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and } \xi_i \geq 0, \forall i$$

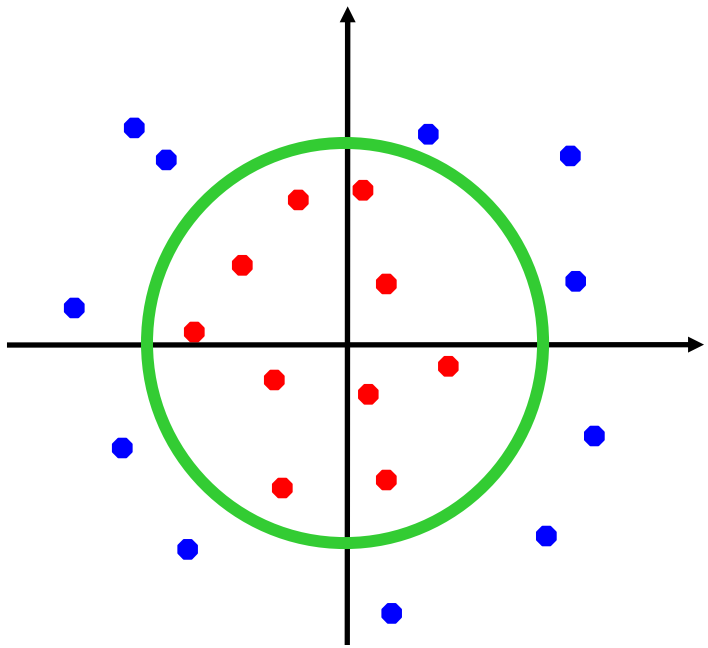
What if data is not linearly separable: Other Ideas?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1



Can we do something to the inputs?

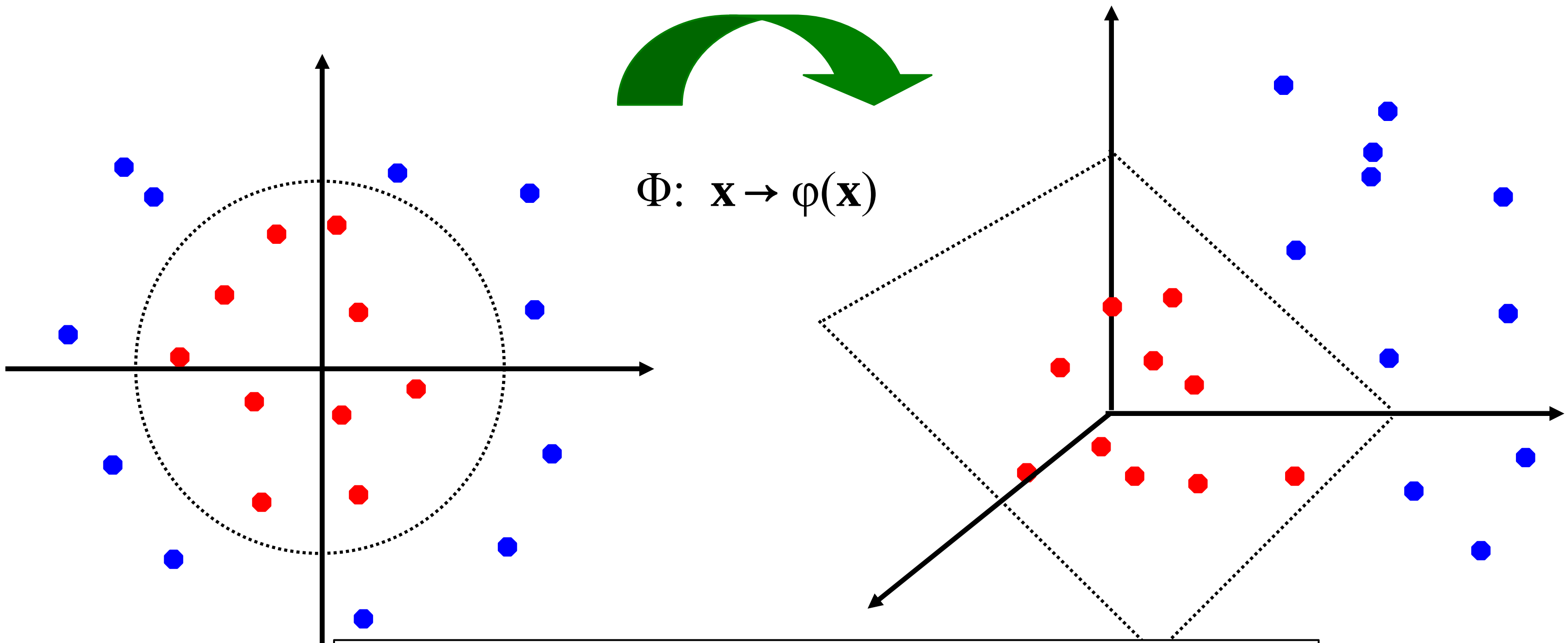
Another Example



Not linearly separable

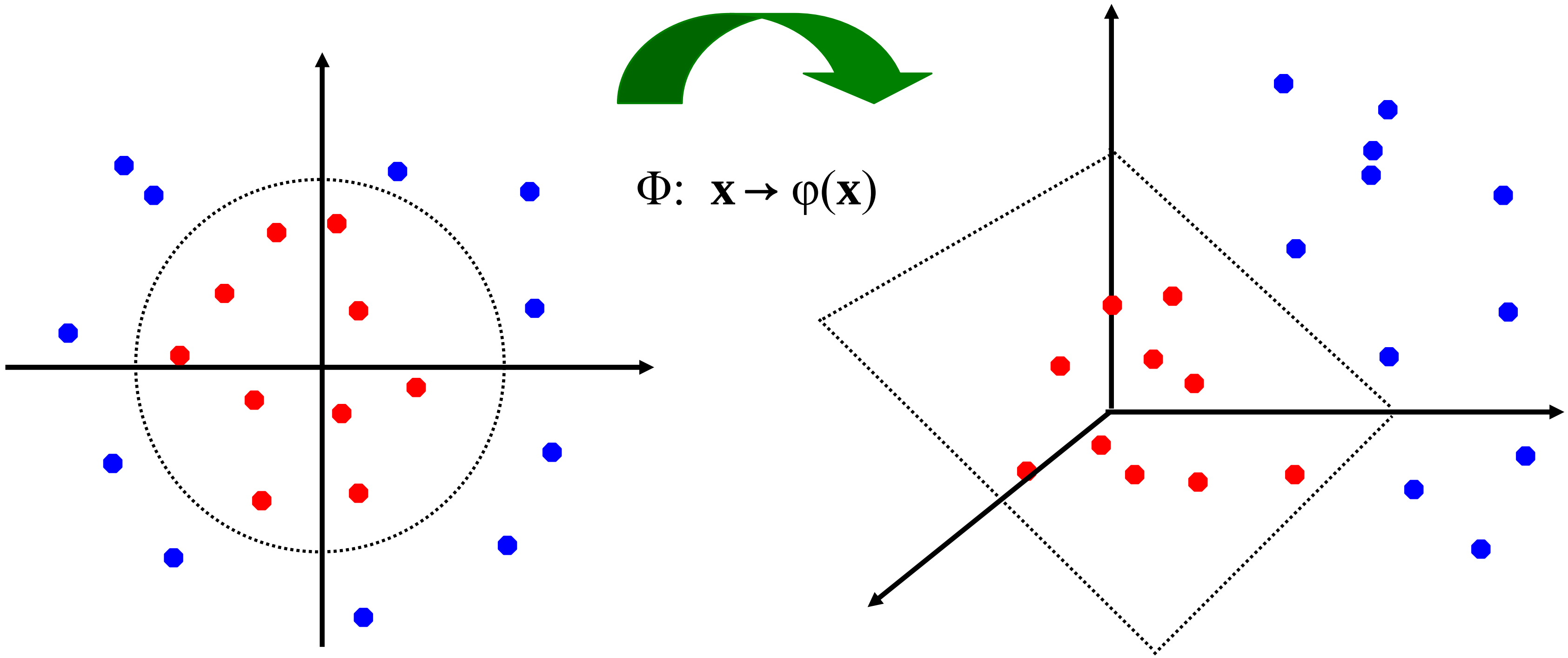
What if data is not linearly separable?

Approach 2: Map original input space to higher-dimensional feature space; use linear classifier in higher-dim. space



$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Problem with high dimensional spaces



Computation in high-dimensional feature space can be costly

The high dimensional projection function $\Phi(\mathbf{x})$ may be too complicated to compute

Kernel trick to the rescue!

The Kernel Trick

Recall the SVM optimization problem: Maximize

$$\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

subject to $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

Insight:

The data points only appear as inner product

- No need to compute $\phi(\mathbf{x})$ explictly!
- Just replace inner product $\mathbf{x}_i \cdot \mathbf{x}_j$ with a **kernel function** $\boldsymbol{K}(\mathbf{x_i},\mathbf{x_j}) = \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})$
- E.g., Gaussian kernel $\boldsymbol{K}(\mathbf{x_i},\mathbf{x_j}) = \exp(-\|\mathbf{x}_i-\mathbf{x}_j\|^2/2\sigma^2)$
- E.g., Polynomial kernel $\boldsymbol{K}(\mathbf{x_i},\mathbf{x_j}) = (\mathbf{x_i}\cdot\mathbf{x_j}+1)^d$

An Example for ϕ(.) and K(.,.)

Suppose ϕ(.) is given as follows

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

An inner product in the feature space is

$$\langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle = (1 + x_1 y_1 + x_2 y_2)^2$$

So, if we define the kernel function as follows, there is no need to compute ϕ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1 y_1 + x_2 y_2)^2$$

This use of kernel function to avoid computing ϕ(.) explicitly is known as the kernel trick

Summary: Steps for Classification using SVMs

Prepare the data matrix

Select the kernel function to use

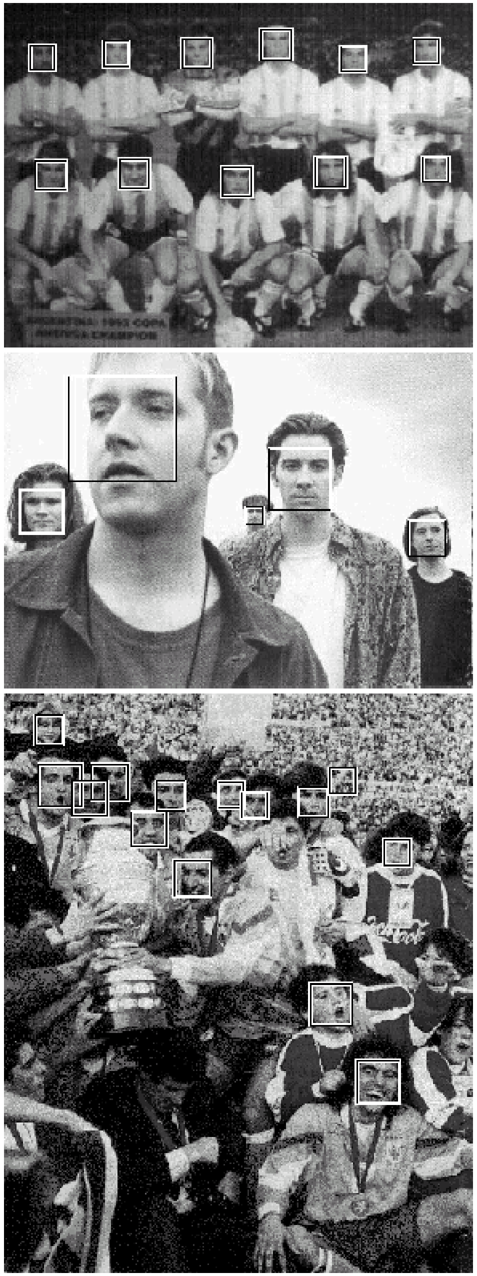
Select parameters of the kernel function

- You can use the values suggested by the SVM software, or use cross-validation

Execute the training algorithm and obtain the α_i

Classify new data using the learned α_i

Face Detection using SVMs

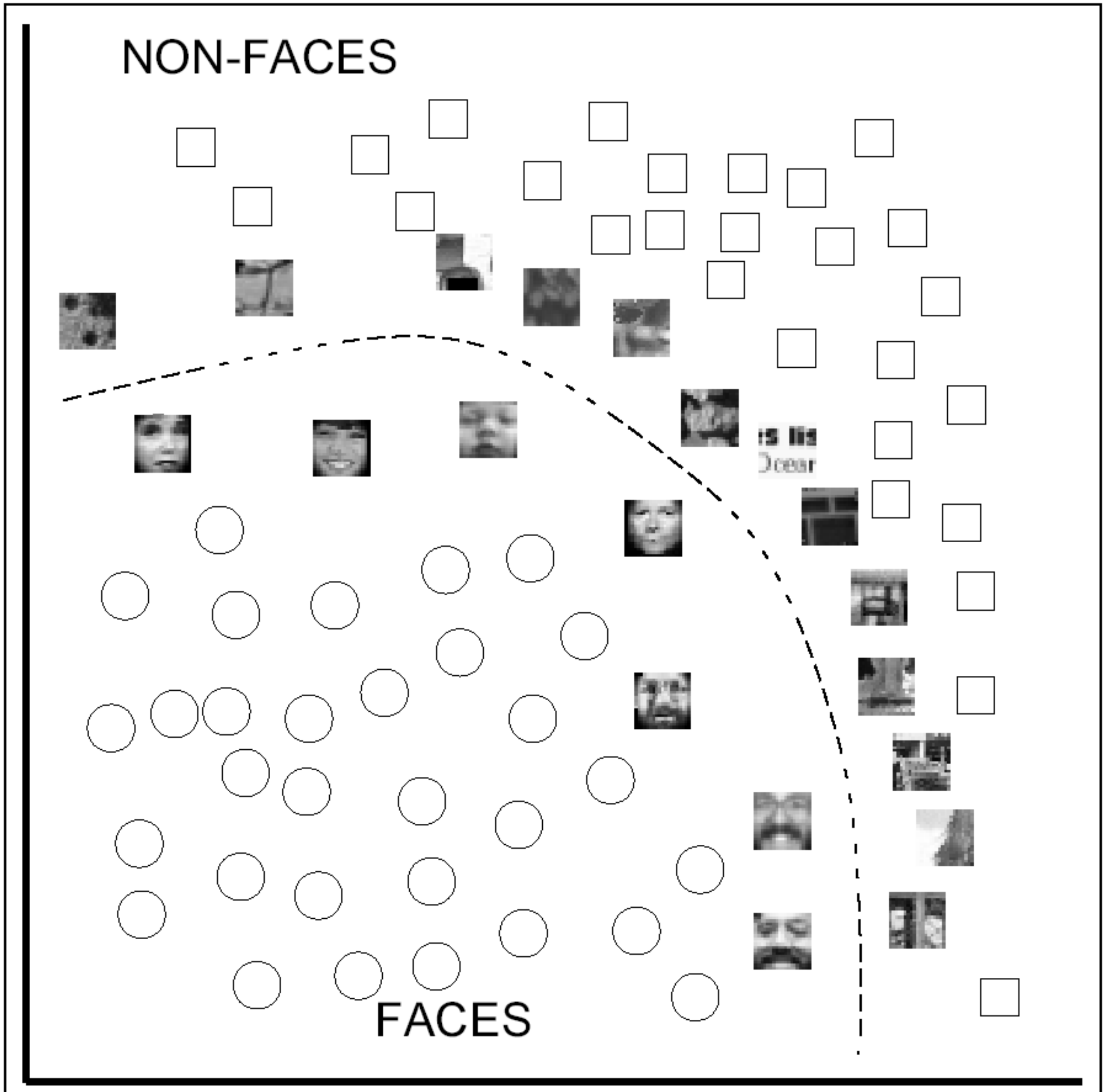


	Test Set A		Test Set B	
	Detect Rate	False Alarms	Detect Rate	False Alarms
SVM	97.1 %	4	74.2%	20
Sung *et al.*	94.6 %	2	74.2%	11

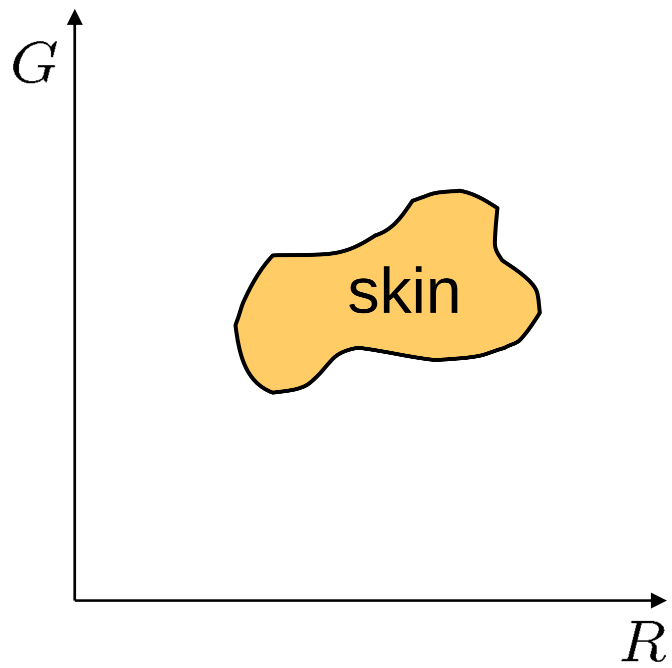
Kernel used: Polynomial of degree 2

(Osuna, Freund, Girosi, 1998)

Support Vectors



Another Problem: Skin Detection



Skin pixels have a distinctive range of colors

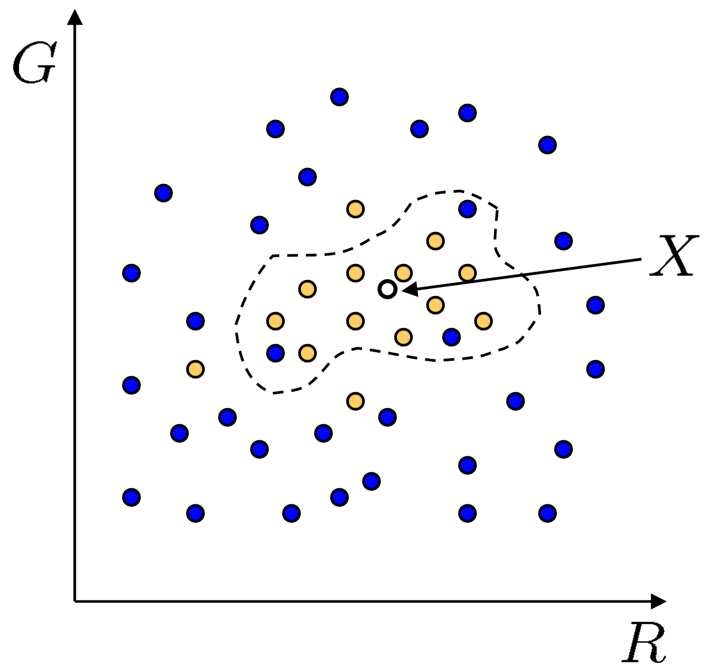
- Corresponds to region(s) in RGB color space
  - for visualization, only R and G components are shown above

Skin classifier

- A pixel X = (R,G,B) is skin if it is in the skin region
- But how to find this region?

(This and next few slides adapted from Steve Seitz's slides)

Skin detection as a classification problem



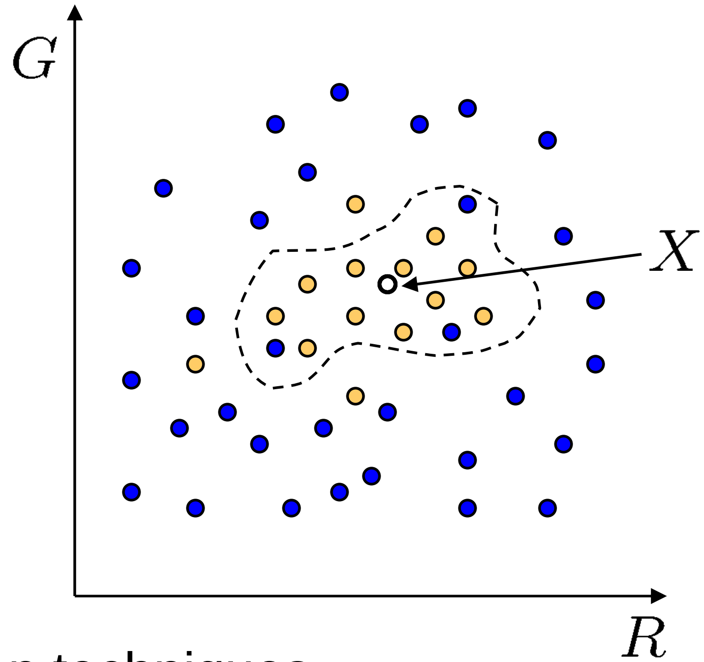
Learn the skin region from labeled examples

- Manually label pixels in one or more “training images” as skin or not skin
- Plot the training data in RGB space
  - skin pixels shown in orange, non-skin pixels shown in blue
  - some skin pixels may be outside the region, non-skin pixels inside. Why?

Skin classifier

- Given X = (R,G,B): determine if it is skin or not

Skin classification techniques



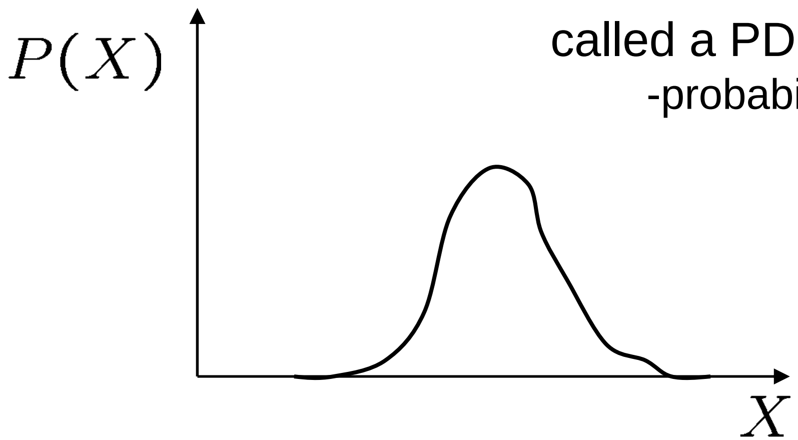
Possible classification techniques

- Nearest neighbor (or K-NN)
  - find labeled pixel closest to X
- Find plane/curve that separates the two classes
  - E.g., Support Vector Machines (SVM)
- Probabilistic approach
  - fit a probability density/distribution model to each class

Probability

Basic probability

- X is a random variable
- P(X) is the probability that X achieves a certain value



called a PDF
-probability distribution/density function

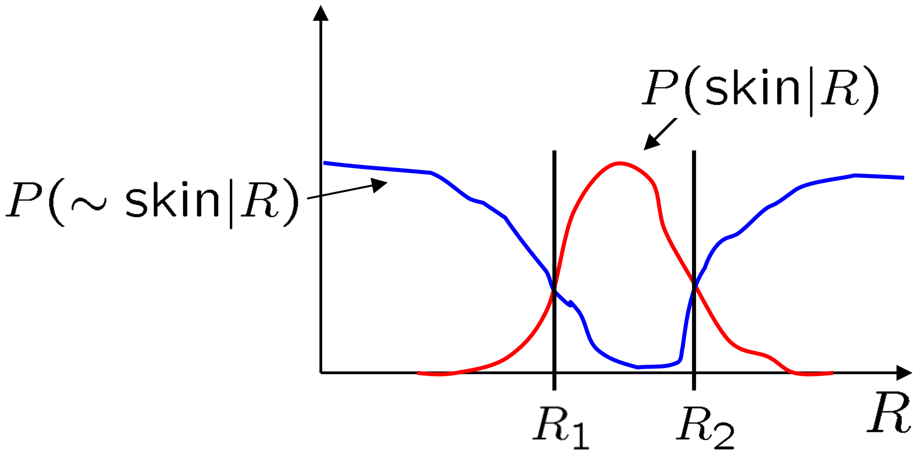
- $0 \leq P(X) \leq 1$

- $\int_{-\infty}^{\infty} P(X)dX = 1$ or $\sum P(X) = 1$

continuous X discrete X

- Conditional probability: P(X | Y)
  - probability of X given that we already know Y

Probabilistic skin classification



Now we can model uncertainty

- Each pixel has a probability of being skin or not skin

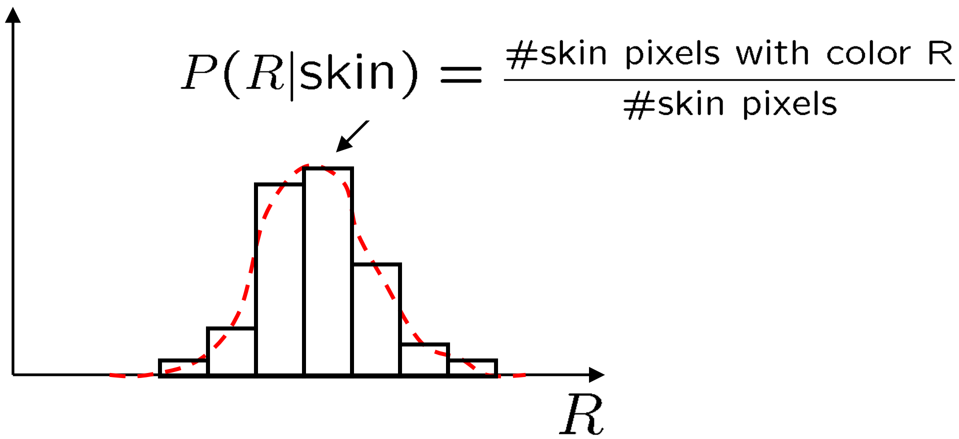
$$P(\sim \text{skin}|R) = 1 - P(\text{skin}|R)$$

Skin classifier

- Given X = (R,G,B): how to determine if it is skin or not?
- Choose interpretation of highest probability
  - set X to be a skin pixel if and only if $R_1 < X \leq R_2$

Where do we get $P(\text{skin}|R)$ and $P(\sim \text{skin}|R)$?

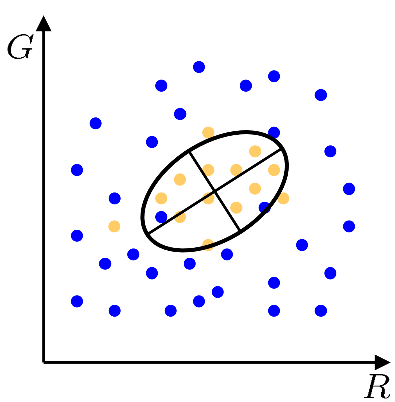
Learning conditional PDF's



We can calculate P(R | skin) from a set of training images

- It is simply a histogram over the pixels in the training images
  - each bin R_i contains the proportion of skin pixels with color R_i

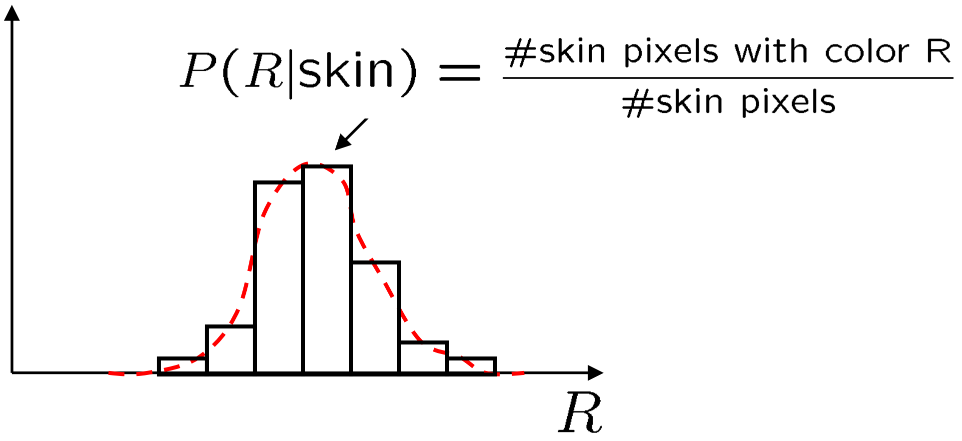
This doesn't work as well in higher-dimensional spaces. Why not?



Approach: fit parametric PDF functions

- common choice is rotated Gaussian
  - center $\mathbf{c} = \overline{X}$
  - covariance $\sum_X (X - \overline{X})(X - \overline{X})^T$

Learning conditional PDF's



We can calculate P(R | skin) from a set of training images

- It is simply a histogram over the pixels in the training images
  - each bin R_i contains the proportion of skin pixels with color R_i

But this isn't quite what we want

- Why not? How to determine if a pixel is skin?
- We want P(skin | R) not P(R | skin)
- How can we get it?

Bayes rule

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

In terms of our problem:

what we measure (**likelihood**)

domain knowledge (**prior**)

$$P(\text{skin}|R) = \frac{P(R|\text{skin})\ P(\text{skin})}{P(R)}$$

what we want (**posterior**)

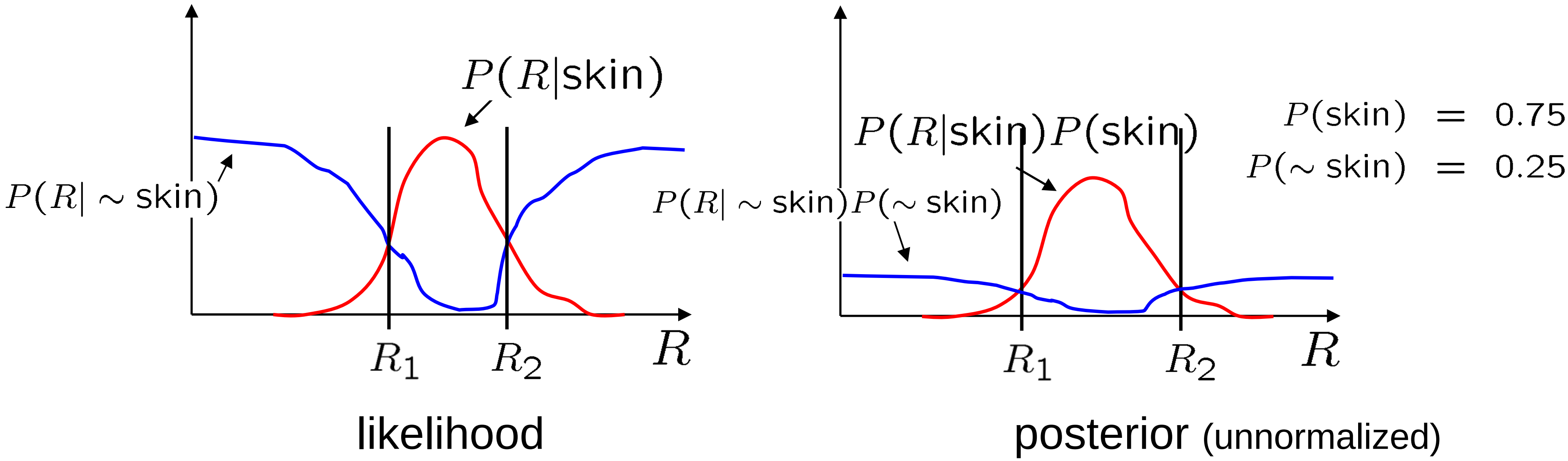
normalization term

$$P(R) = P(R|\text{skin})P(\text{skin}) + P(R|\sim\text{skin})P(\sim\text{skin})$$

What could we use for the prior P(skin)?

- Could use domain knowledge
  - P(skin) may be larger if we know the image contains a person
  - for a portrait, P(skin) may be higher for pixels in the center
- Could learn the prior from the training set. How?
  - P(skin) may be proportion of skin pixels in training set

Bayesian estimation

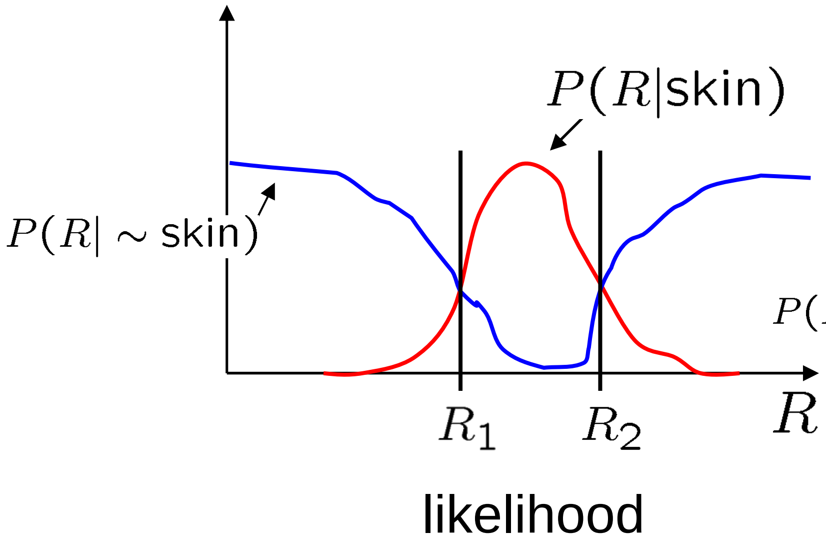


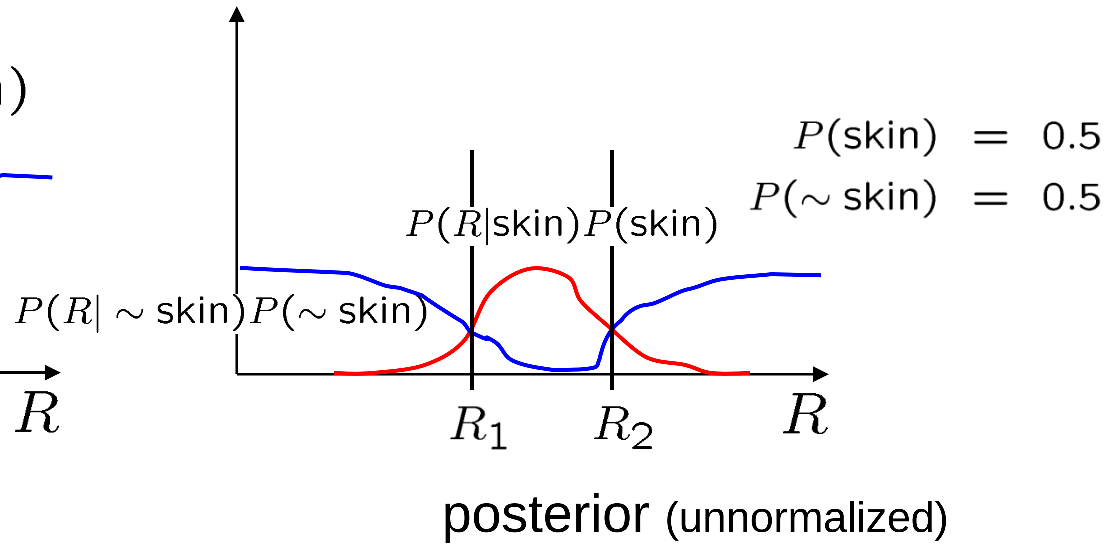
= minimize probability of misclassification

Bayesian estimation

- Goal is to choose the label (skin or ~skin) that maximizes the posterior
  - this is called **Maximum A Posteriori (MAP) estimation**

Bayesian estimation



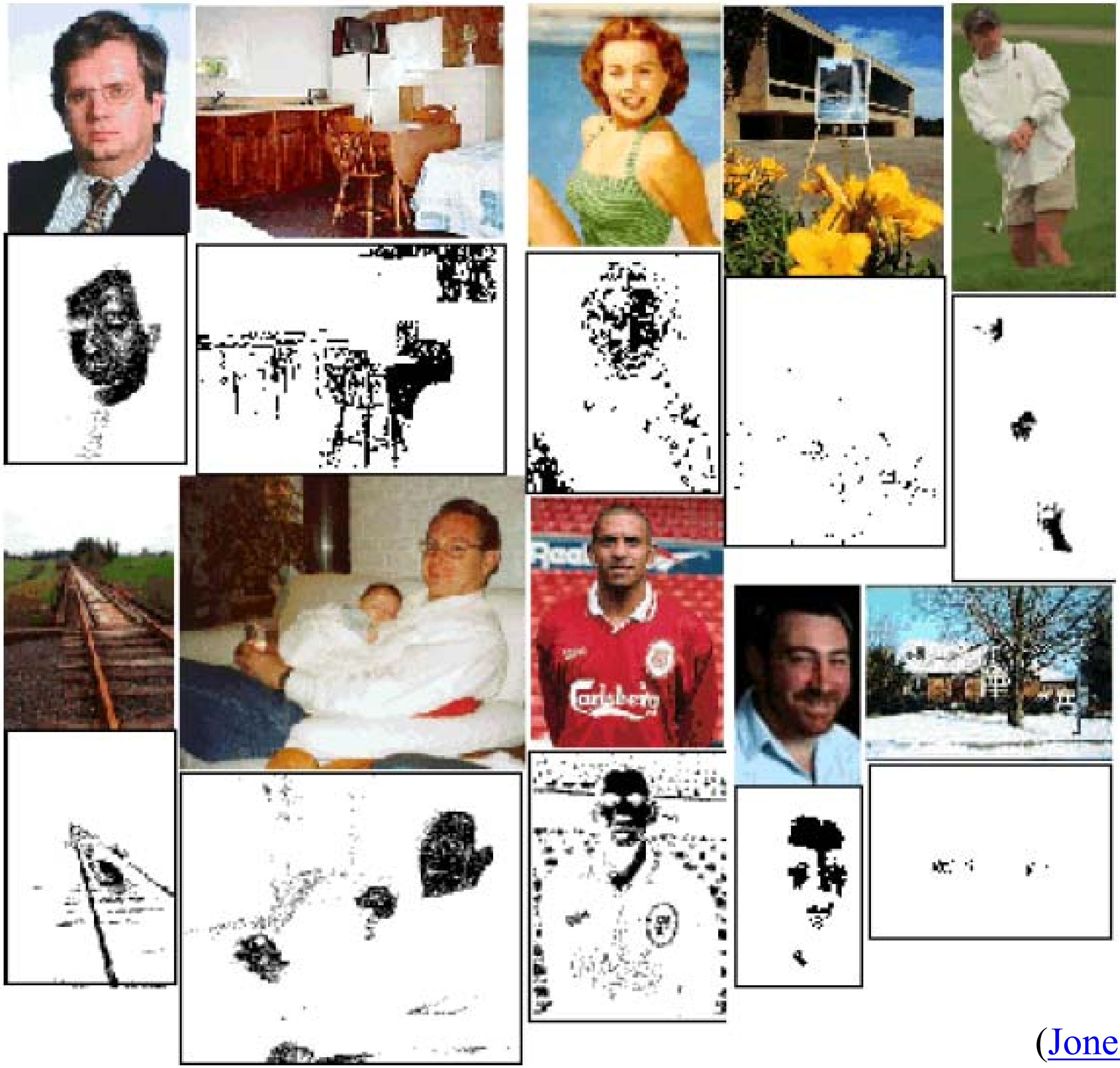


Bayesian estimation

= minimize probability of misclassification

- Goal is to choose the label (skin or ~skin) that maximizes the posterior
  - this is called **Maximum A Posteriori (MAP) estimation**
- Suppose the prior is uniform: P(skin) = P(~skin) = 0.5
  - in this case $P(\text{skin}|R) = cP(R|\text{skin}),\quad P(\sim\text{skin}|R) = cP(R|\sim\text{skin})$
  - maximizing the posterior is equivalent to maximizing the likelihood
    - » $P(\text{skin}|R) > P(\sim\text{skin}|R)$ if and only if $P(R|\text{skin}) > P(R|\sim\text{skin})$
  - this is called **Maximum Likelihood (ML) estimation**

Skin detection results



(Jones & Rehg, 1999)

Next Time: Color

Things to do:

- Work on Project 2
- Read Chap. 6

Reverand Thomas Bayes
Nonconformist minister
(1702-1761)

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Bayes rules!