CSCE 990 Lecture 0: Administrivia

Stephen D. Scott

January 10, 2006

Welcome to CSCE 990 (aka 978)!

You should have the following handouts:

1. Syllabus

2. Copies of slides (also on web page)

Please check off or write your name on the roster (if you write your name, indicate if you plan to register for the course)

Also, don't forget Homework 0 (due January 17) on the web page: have a JPEG image of yourself (and yourself only) ready to upload. The image must be smaller than 100k, and must be in JPEG format.

CSCE 990 Lecture 1: Introduction

Stephen D. Scott

January 10, 2006

Outline

- Overview of Machine Learning

- Overview of SVMs:

  1. Introduction to linear classifiers and the Perceptron algorithm

  2. Introducing nonlinear remappings

  3. Margins, duality, kernels, convexity

What is Machine Learning?

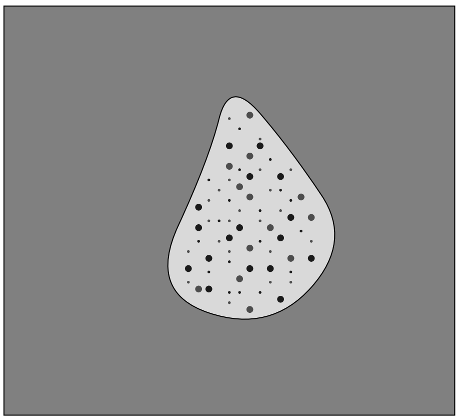
- Machine Learning: classify objects (instances, examples) into categories (classes, labels)

- Has roots in artificial intelligence, probability theory, statistics, computational complexity theory, information theory, linear algebra, and algorithms

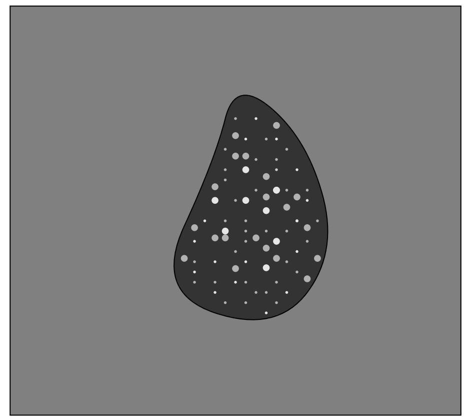
- Applications: Machine vision, OCR, handwriting recognition, computer-aided diagnosis, speech recognition, computational biology

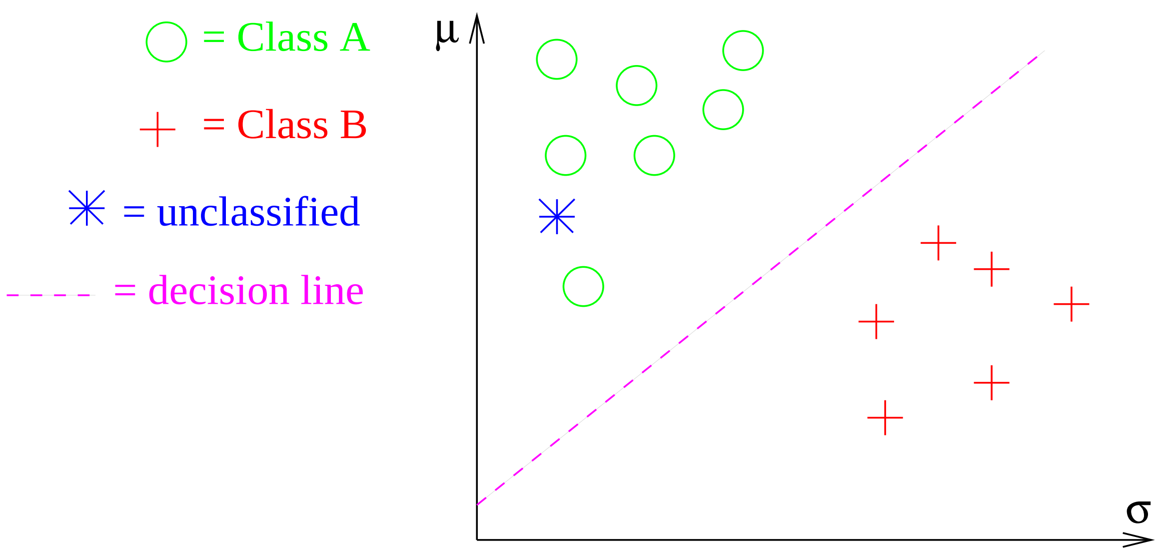
An Example

Class A



Class B





Features, Feat. Vectors, Classifiers

- $\mathbf{x} = (x_1, \ldots, x_\ell)$ is a feature vector of ℓ features
  - E.g. $\mathbf{x} = (\mu, \sigma)$ from previous slide
  - Feature vectors also known as instances or examples

- A classifier separates the feature space into regions corresponding to two or more classes (also known as labels)
  - Use to classify new, unlabeled instances
  - E.g. decision line from previous slide

- Classifier built by training (learning) using a training set of labeled instances

- Can also use labeled instances as a testing set to evaluate the classifier

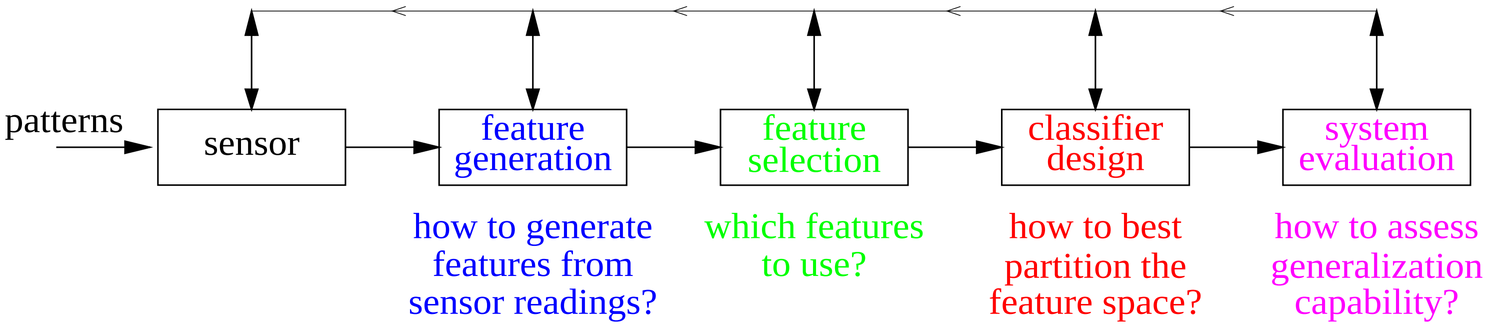
Applications of Machine Learning

- Data mining: Extracting new information from medical records, maintenance records, biological sequence databases, etc.

- Self-customizing programs: E.g. a learning newsreader/browser that learns what you like and seeks it out

- Applications we can't program by hand: E.g. speech recognition, image analysis, autonomous driving

Solving a ML Problem



1. Feat. Gen.: Want to reduce sensitivity to noise and reduce complexity but retain important info

   $\boxed{\text{“Pack” sensor info into small number of features}}$

2. Feat. Sel.: Want to reduce complexity and reduce redundancy but retain important info

   $\boxed{\text{Select small set of features that separates classes}}$

3. Classif. Des.: Want small generalization error and fast training and classification (i.e. low complexity)

4. Sys. Eval.: Want to accurately estimate classifier's generalization error

We'll focus on stage 3, and a little on 4

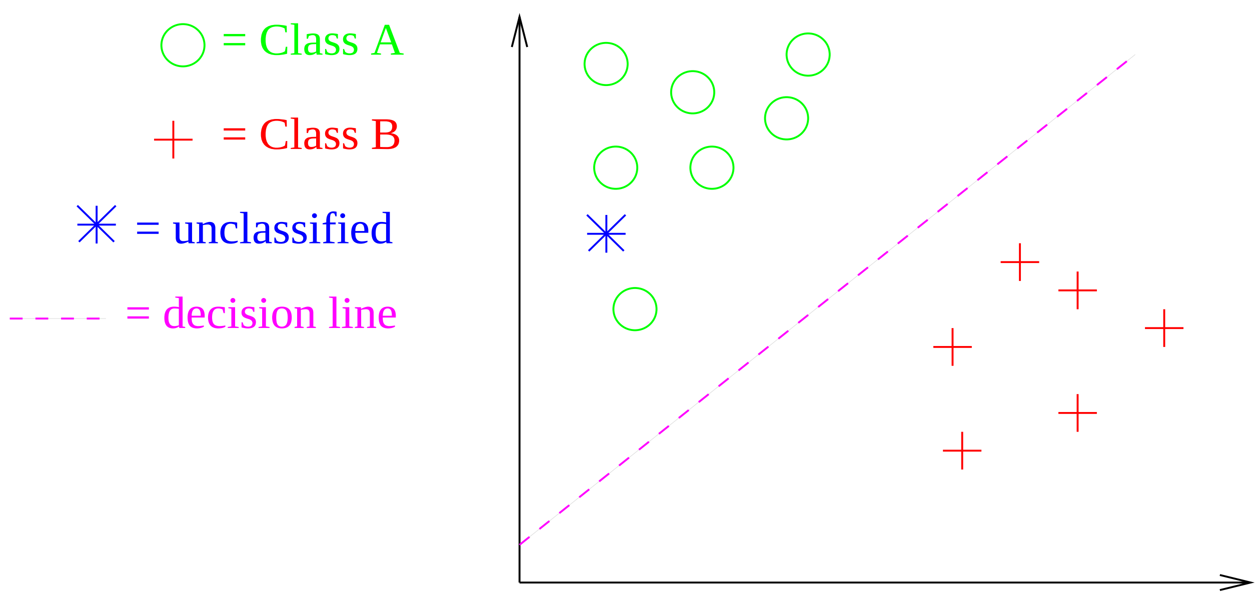
Introduction to SVMs

Linear Classifiers

- Linear classifiers use a decision hyperplane to perform classification

- Simple and efficient to train and use

- Optimality requires linear separability of classes



Linear Discriminant Functions

- Let $\mathbf{w} = (w_1, \ldots, w_\ell)$ be a weight vector and w_0 (a.k.a. θ) be a threshold

- Decision surface is a hyperplane:

$$\mathbf{w} \cdot \mathbf{x} + w_0 = 0$$

- E.g. predict label $y_{\mathbf{x}} = +1$ if $\sum_{i=1}^{\ell} w_i x_i > w_0$, otherwise predict that $y_{\mathbf{x}} = -1$

- Where the learning comes in: How to find w_i's

  - Perceptron algorithm

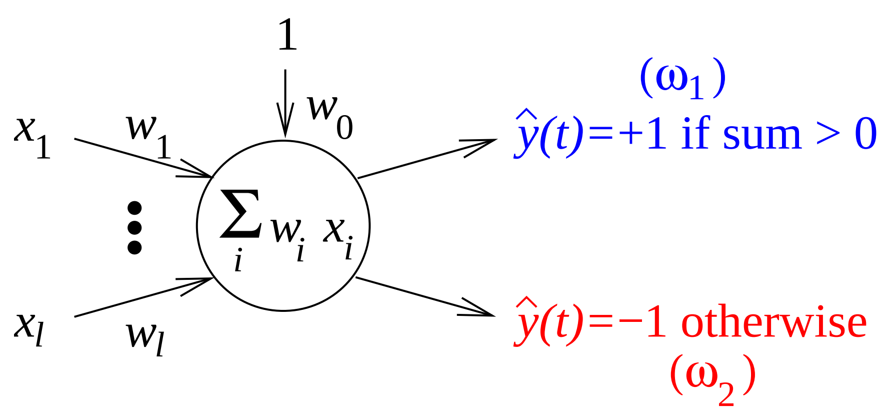
  - Winnow algorithm

The Perceptron Algorithm

- Assume linear separability, i.e. $\exists\, \mathbf{w}^*$ such that

$$\mathbf{w}^* \cdot \mathbf{x} > 0 \quad \forall\, \mathbf{x} \text{ s.t. } y_{\mathbf{x}} = +1$$
$$\mathbf{w}^* \cdot \mathbf{x} \leq 0 \quad \forall\, \mathbf{x} \text{ s.t. } y_{\mathbf{x}} = -1$$

(w_0^* is included in $\mathbf{w}^*$)



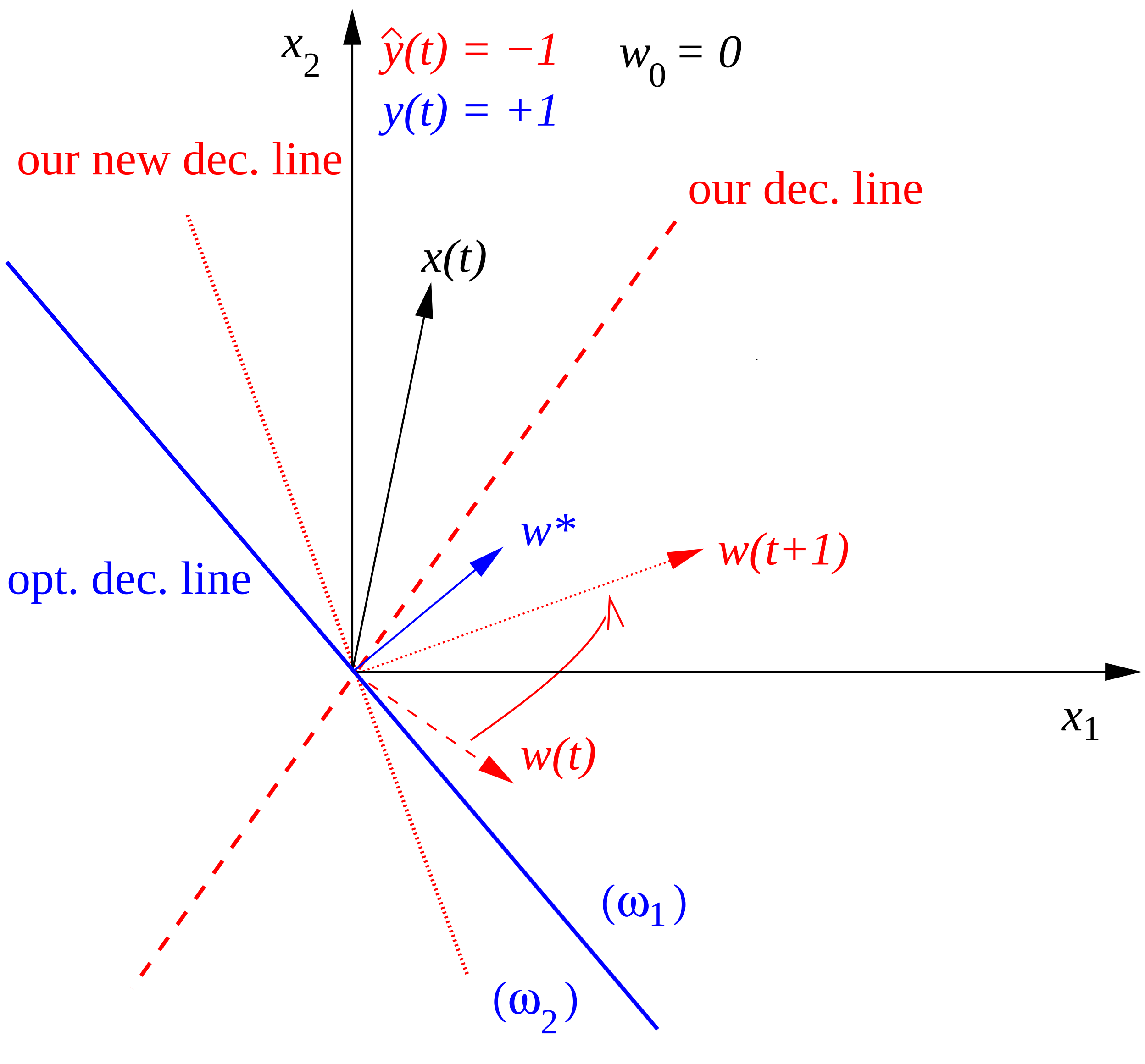
- Given actual label $y(t)$ for trial t, update weights:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \rho(y(t) - \hat{y}(t))\mathbf{x}(t)$$

  - $\rho > 0$ is learning rate
  - $(y(t) - \hat{y}(t))$ moves weights toward correct prediction for $\mathbf{x}$

The Perceptron Algorithm

Example



The Perceptron Algorithm

Intuition

- Compromise between correctiveness and conservativeness
  - Correctiveness: Tendency to improve on $\mathbf{x}(t)$ if prediction error made
  - Conservativeness: Tendency to keep $\mathbf{w}(t+1)$ close to $\mathbf{w}(t)$
- Use cost function that measures both:

$$
\begin{aligned}
U(\mathbf{w}) &= \overbrace{\|\mathbf{w}(t+1) - \mathbf{w}(t)\|_2^2}^{conserv.} + \eta \overbrace{(y(t) - \mathbf{w}(t+1)\cdot\mathbf{x}(t))^2}^{corrective} \\
&= \sum_{i=1}^{\ell} (w_i(t+1) - w_i(t))^2 + \\
&\quad \eta \left(y(t) - \sum_{i=1}^{\ell} w_i(t+1)\, x_i(t) \right)^2
\end{aligned}
$$

The Perceptron Algorithm
Intuition
(cont'd)

- Take gradient w.r.t. $\mathbf{w}(t+1)$ and set to $\mathbf{0}$:

$$\begin{aligned} 0 = & 2\left(w_i(t+1) - w_i(t)\right) - \\ & 2\eta\left(y(t) - \sum_{i=1}^{\ell} w_i(t+1)\, x_i(t)\right) x_i(t) \end{aligned}$$

- Approximate with

$$\begin{aligned} 0 = & 2\left(w_i(t+1) - w_i(t)\right) - \\ & 2\eta\left(y(t) - \sum_{i=1}^{\ell} w_i(t)\, x_i(t)\right) x_i(t), \end{aligned}$$

which yields

$$\begin{aligned} w_i(t+1) = & w_i(t) + \\ & \eta\left(y(t) - \sum_{i=1}^{\ell} w_i(t) x_i(t)\right) x_i(t) \end{aligned}$$

- Applying threshold to summation yields

$$w_i(t+1) = w_i(t) + \eta\left(y(t) - \hat{y}(t)\right) x_i(t)$$

The Perceptron Algorithm

Miscellany

- If classes linearly separable, then by cycling through vectors, guaranteed to converge in finite number of steps

- For real-valued output (aka regression), can replace threshold function on sum with
  - Identity function: $f(x) = x$
  - Sigmoid function: e.g. $f(x) = \frac{1}{1+\exp(-ax)}$
  - Hyperbolic tangent: e.g. $f(x) = c \tanh(ax)$

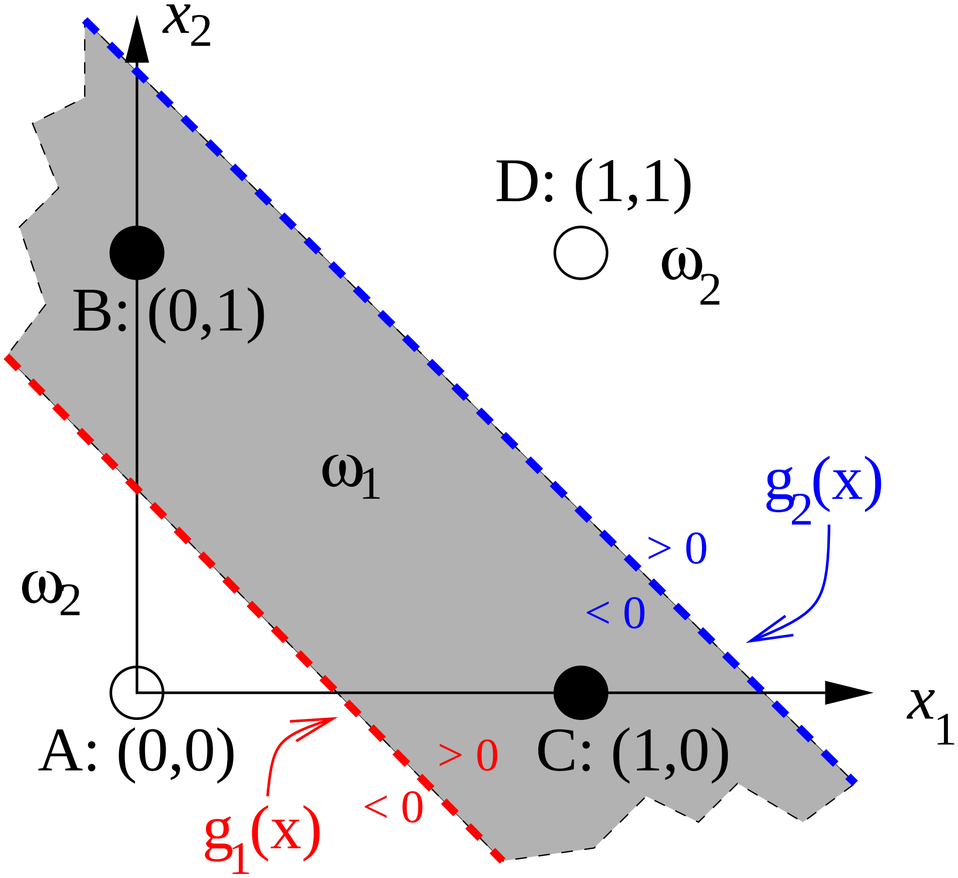
Adding Nonlinearity

- For non-linearly separable classes, performance of even the best linear classifier might not be good

- Thus we will remap feature vectors to new space where they are (almost) linearly separable

- Many ways to do this; we'll introduce a few and then focus on using kernels

Getting Started: The XOR Problem



- Can't represent with a single linear separator, but can with intersection of two:

$$g_1(\mathbf{x}) = 1 \cdot x_1 + 1 \cdot x_2 - 1/2$$
$$g_2(\mathbf{x}) = 1 \cdot x_1 + 1 \cdot x_2 - 3/2$$

- $\omega_1 = \left\{ \mathbf{x} \in \mathbb{R}^\ell : g_1(\mathbf{x}) > 0 \text{ AND } g_2(\mathbf{x}) < 0 \right\}$

- $\omega_2 = \left\{ \mathbf{x} \in \mathbb{R}^\ell : g_1(\mathbf{x}), g_2(\mathbf{x}) < 0 \text{ OR } g_1(\mathbf{x}), g_2(\mathbf{x}) > 0 \right\}$

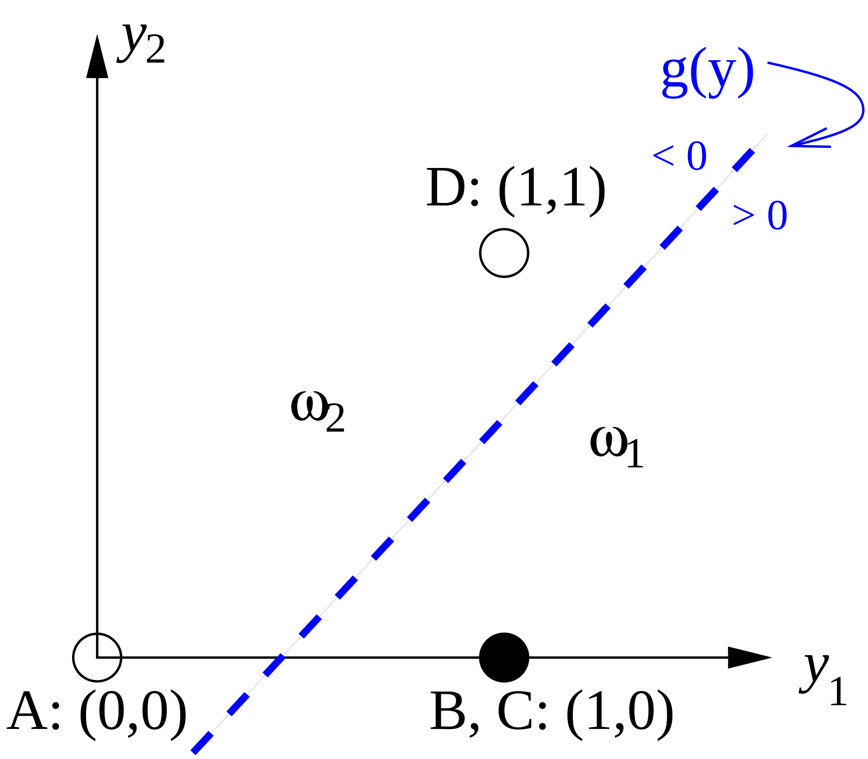
Getting Started: The XOR Problem (cont'd)

- Let $y_i = \begin{cases} 0 & \text{if } g_i(\mathbf{x}) < 0 \\ 1 & \text{otherwise} \end{cases}$

Class	(x_1, x_2)	$g_1(\mathbf{x})$	y_1	$g_2(\mathbf{x})$	y_2
ω_1	B: $(0,1)$	$1/2$	1	$-1/2$	0
ω_1	C: $(1,0)$	$1/2$	1	$-1/2$	0
ω_2	A: $(0,0)$	$-1/2$	0	$-3/2$	0
ω_2	D: $(1,1)$	$3/2$	1	$1/2$	1

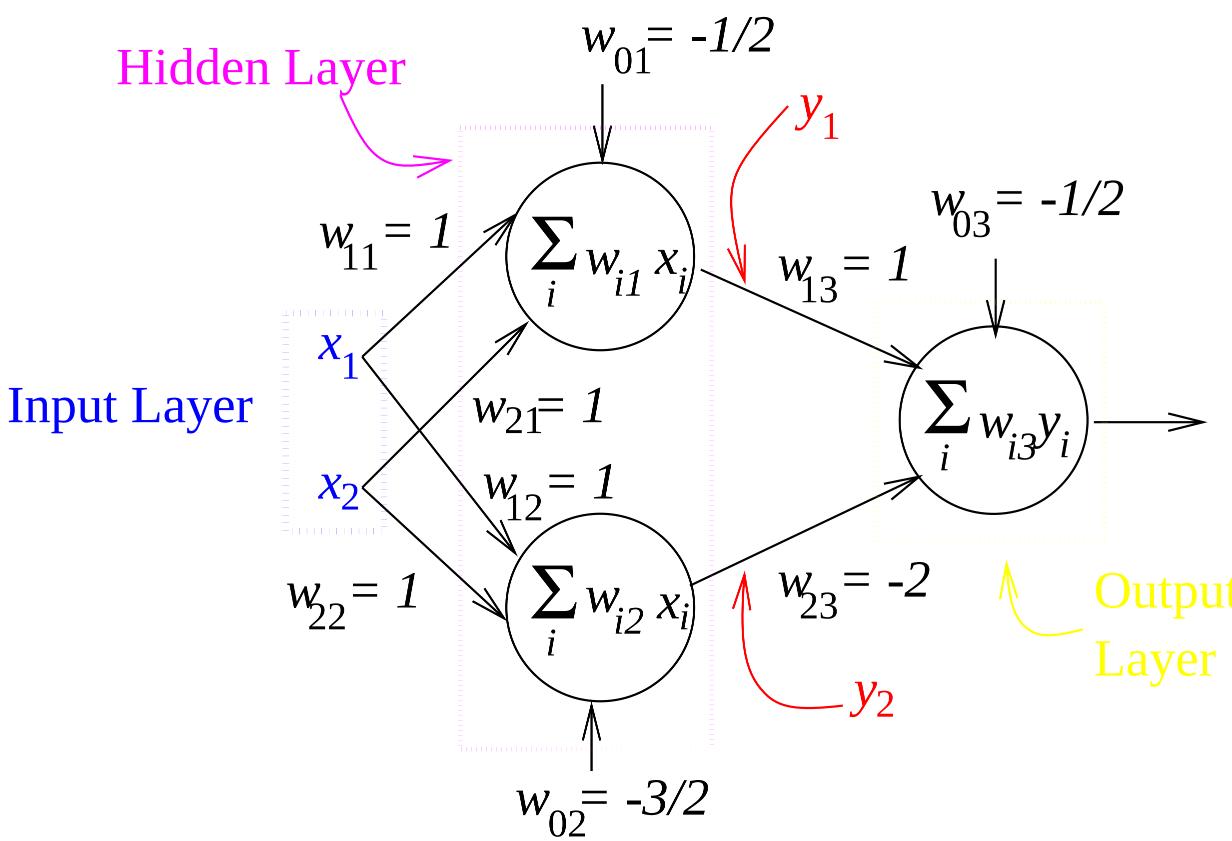
- Now feed y_1, y_2 into:

$$g(\mathbf{y}) = 1 \cdot y_1 - 2 \cdot y_2 - 1/2$$



Getting Started: The XOR Problem (cont'd)

- In other words, we remapped all vectors $\mathbf{x}$ to $\mathbf{y}$ such that the classes are linearly separable in the new vector space



- This is a two-layer perceptron or two-layer feedforward neural network

- Each neuron outputs 1 if its weighted sum exceeds its threshold, 0 otherwise

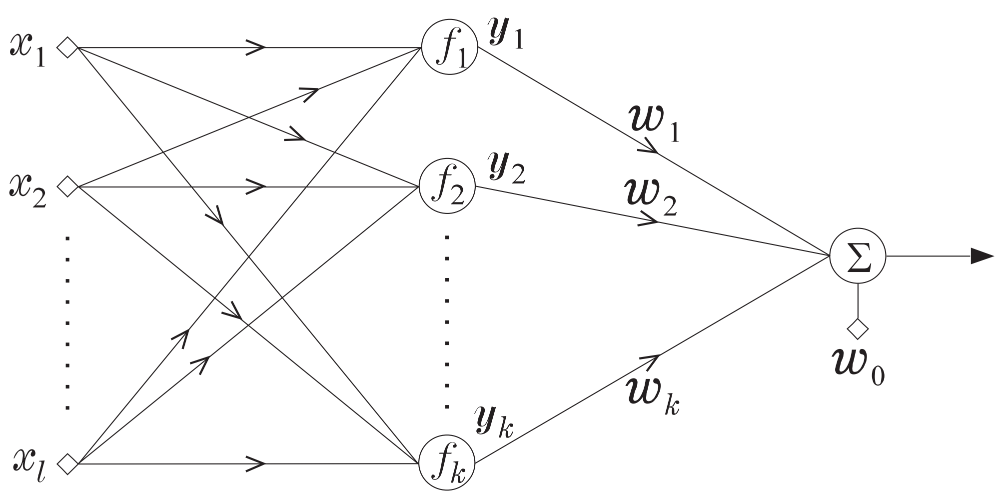
Generalized Linear Classifiers

- In XOR problem, used linear threshold funcs. in hidden layer to map non-linearly sep. classes to new space where they were lin. sep.

- Output layer gave sep. hyperplane in new space

- Replace hidden-layer lin. thresh. funcs. with family of nonlinear functions $f_i : \mathbb{R}^\ell \to \mathbb{R}$, $i = 1, \ldots, k$

- Hidden layer maps $\mathbf{x} \in \mathbb{R}^\ell$ to $\mathbf{y} = (f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))$ and output layer finds separating hyperplane:

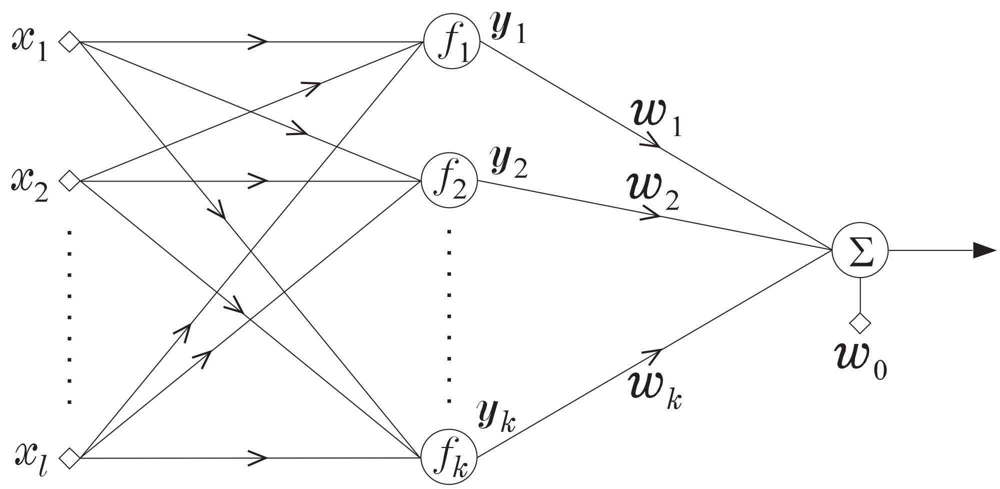


- I.e. approximating separating surface as linear combination of interpolation functions:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{k} w_i \, f_i(\mathbf{x})$$

Generalized Linear Classifiers

Polynomial Classifiers



- Approximate $g(\mathbf{x})$ by linear combination of up to order r polynomials over components of $\mathbf{x}$
- E.g. for $r = 2$

$$g(\mathbf{x}) = w_0 + \overbrace{\sum_{i=1}^{\ell} w_i x_i}^{w_1 f_1 + \cdots + w_\ell f_\ell} + \overbrace{\sum_{i=1}^{\ell-1} \sum_{m=i+1}^{\ell} w_{im} x_i x_m}^{w_{\ell+1} f_{\ell+1} + \cdots + w_{k-\ell} f_{k-\ell}}$$

$$+ \underbrace{\sum_{i=1}^{\ell} w_{ii} x_i^2}_{w_{k-\ell+1} f_{k-\ell+1} + \cdots w_k f_k}, \qquad k = \ell(\ell+3)/2$$

- For $\ell = 2$, $\mathbf{x} = (x_1, x_2)$ and

$$\mathbf{y} = \left(x_1, x_2, x_1 x_2, x_1^2, x_2^2\right)$$
$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{y} + w_0$$
$$\mathbf{w} = (w_1, w_2, w_{12}, w_{11}, w_{22})$$

Generalized Linear Classifiers

Polynomial Classifiers

(cont'd)

- In general, will use all terms of form $x_1^{p_1} x_2^{p_2} \cdots x_\ell^{p_\ell}$ for all $p_1 + \cdots + p_\ell \leq r$

- This gives size of $\mathbf{y}$ to be
$$k = \frac{(\ell + r)!}{r!\,\ell!},$$
so time to classify and update exponential in $(\ell + r)$

Generalized Linear Classifiers

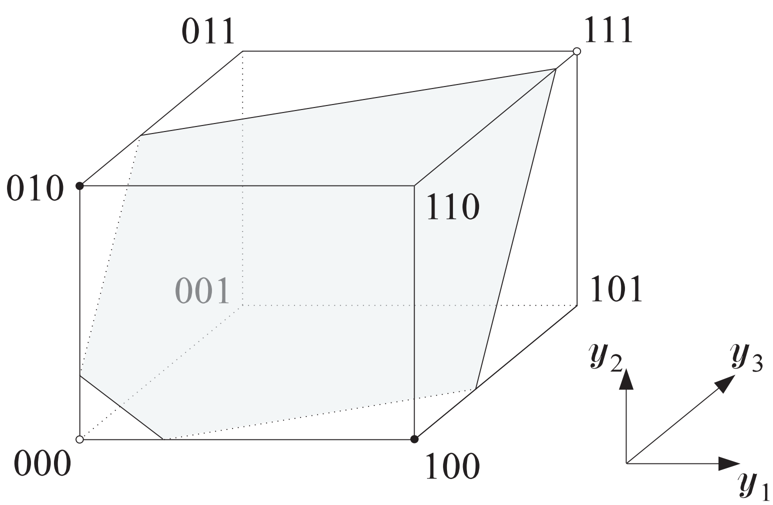
Polynomial Classifiers

Example: XOR

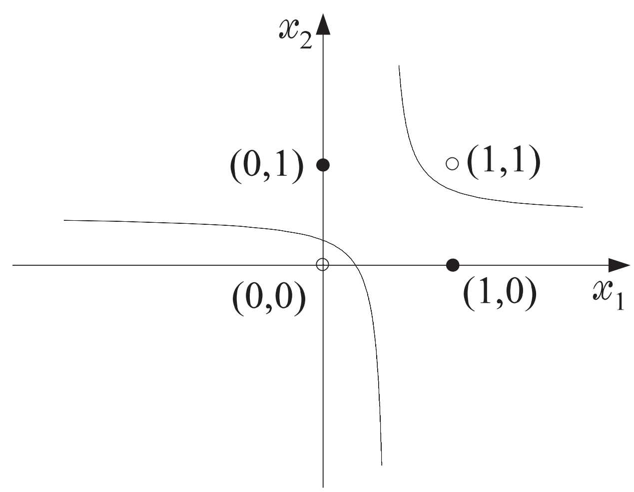
- Use $\mathbf{y} = [x_1, x_2, x_1 x_2]$

Class	$[x_1, x_2]$	$[y_1, y_2, y_3]$
ω_1	$[0, 1]$	$[0, 1, 0]$
ω_1	$[1, 0]$	$[1, 0, 0]$
ω_2	$[0, 0]$	$[0, 0, 0]$
ω_2	$[1, 1]$	$[1, 1, 1]$

$$g(\mathbf{y}) = y_1 + y_2 - 2y_3 - \frac{1}{4} \qquad g(\mathbf{x}) = -\frac{1}{4} + x_1 + x_2 - 2x_1 x_2$$



(a)



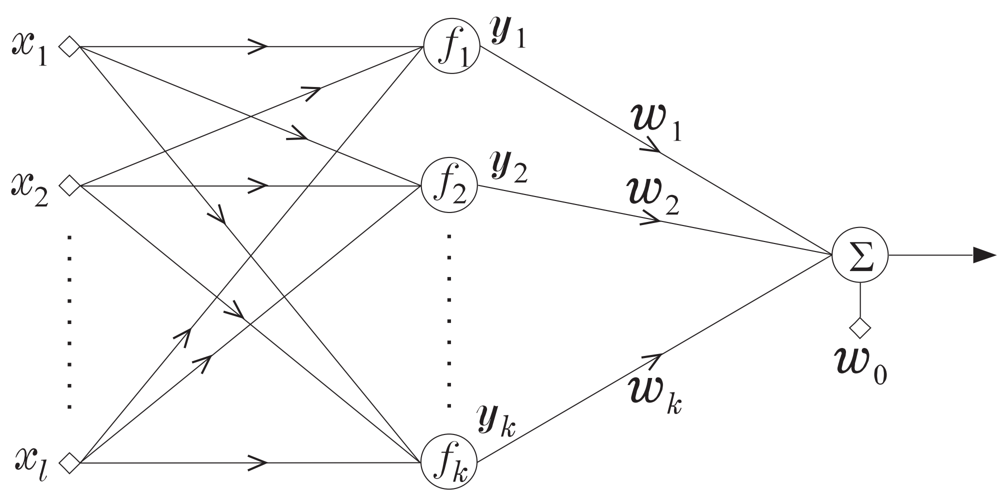
(b)

$$> 0 \Rightarrow \mathbf{x} \in \omega_1$$

$$< 0 \Rightarrow \mathbf{x} \in \omega_2$$

Generalized Linear Classifiers

Radial Basis Function Networks



- Argument of func. f_i is $\mathbf{x}$'s Euclidian distance from designated center $\mathbf{c}_i$, e.g.

$$f_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|_2^2}{2\sigma_i^2}\right)$$

- So

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{k} w_i \exp\left(-\frac{(\mathbf{x}-\mathbf{c}_i)\cdot(\mathbf{x}-\mathbf{c}_i)}{2\sigma_i^2}\right)$$

- Exponential decrease in increased distance gives a very <u>localized</u> activation response

Generalized Linear Classifiers

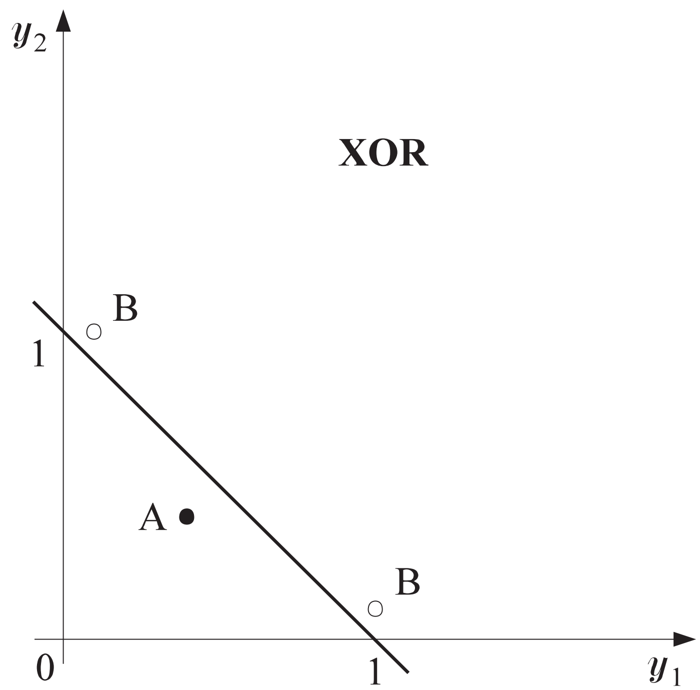
Radial Basis Function Networks

Example: XOR

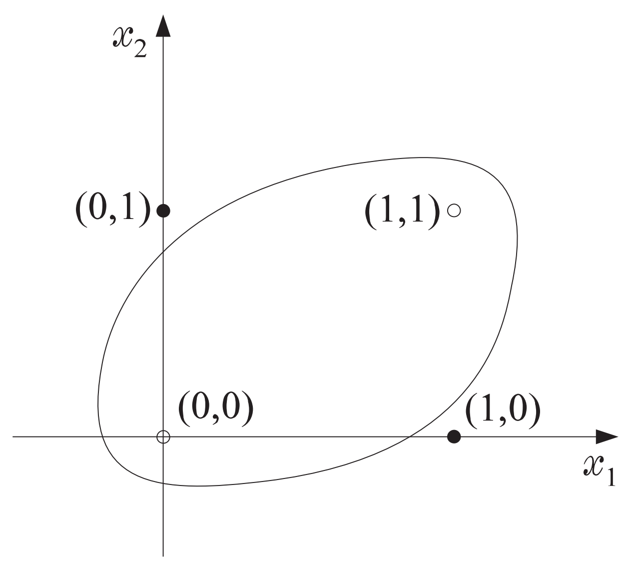
- $\mathbf{c}_1 = [1,1]$, $\mathbf{c}_2 = [0,0]$, $f_i(\mathbf{x}) = \exp\left(-\|\mathbf{x}-\mathbf{c}_i\|_2^2\right)$

Class	$[x_1, x_2]$	$[y_1, y_2]$
ω_1 (A)	$[0, 1]$	$[0.368, 0.368]$
ω_1 (A)	$[1, 0]$	$[0.368, 0.368]$
ω_2 (B)	$[0, 0]$	$[0.135, 1]$
ω_2 (B)	$[1, 1]$	$[1, 0.135]$

$$g(\mathbf{y}) = y_1 + y_2 - 1 \qquad g(\mathbf{x}) = -1 + e^{-\|\mathbf{x}-\mathbf{c}_1\|_2^2} + e^{-\|\mathbf{x}-\mathbf{c}_2\|_2^2}$$



(a)



(b)

$$< 0 \Rightarrow \mathbf{x} \in \omega_1$$

$$> 0 \Rightarrow \mathbf{x} \in \omega_2$$

Support Vector Machines

- Introduced in 1992

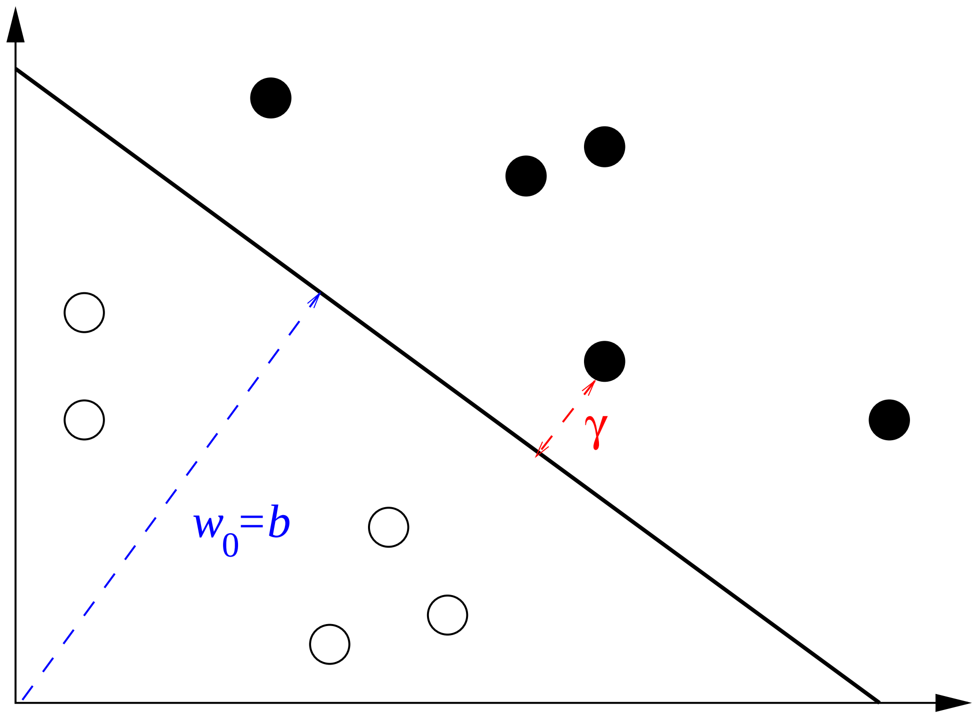
- State-of-the-art technique for classification and regression

- Techniques can also be applied to e.g. clustering and principal components analysis

- Similar to polynomial classifiers and RBF networks in that it remaps inputs and then finds a hyperplane
  - Main difference is how it works

- Features of SVMs:
  - Maximization of margin
  - Duality
  - Use of kernels
  - Use of problem convexity to find classifier (often without local minima)

Support Vector Machines
Margins



- A hyperplane's margin γ is the shortest distance from it to any training vector

- Intuition: larger margin $\Rightarrow$ higher confidence in classifier's ability to generalize

  - Guaranteed generalization error bound in terms of $1/\gamma^2$

- Definition assumes linear separability (more general definitions exist that do not)

Support Vector Machines

Reformulating the Perceptron Algorithm

- $\mathbf{w}(0) \leftarrow \mathbf{0}$, $b(0) \leftarrow 0$, $k \leftarrow 0$, $y_i \in \{-1, +1\} \, \forall i$

- While mistakes are made on training set
  - For $i = 1$ to N ($=$ # training vectors)
    - $*$ If $y_i \, (\mathbf{w}_k \cdot \mathbf{x}_i + b_k) \leq 0$
      - $\cdot$ $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta \, y_i \, \mathbf{x}_i$
      - $\cdot$ $b_{k+1} \leftarrow b_k + \eta \, y_i$
      - $\cdot$ $k \leftarrow k + 1$

- Final predictor: $h(\mathbf{x}) = \text{sgn} \, (\mathbf{w}_k \cdot \mathbf{x} + b_k)$

Support Vector Machines

Duality

- Another way of representing predictor:

$$h(\mathbf{x}) = \text{sgn}\,(\mathbf{w}\cdot\mathbf{x} + b) = \text{sgn}\left(\eta \sum_{i=1}^{N} (\alpha_i\, y_i\, \mathbf{x}_i)\cdot\mathbf{x} + b\right)$$
$$= \text{sgn}\left(\eta \sum_{i=1}^{N} \alpha_i\, y_i\, (\mathbf{x}_i\cdot\mathbf{x}) + b\right)$$

  ($\alpha_i = \#$ mistakes on $\mathbf{x}_i$)

- So perceptron alg has equivalent dual form:
- $\alpha \leftarrow \mathbf{0}$, $b \leftarrow 0$
- While mistakes are made in For loop
  - For $i = 1$ to N ($= \#$ training vectors)
    * If $y_i\left(\eta\sum_{j=1}^{N}\alpha_j\, y_j\, \left(\mathbf{x}_j\cdot\mathbf{x}_i\right) + b\right) \leq 0$
      · $\alpha_i \leftarrow \alpha_i + 1$
      · $b \leftarrow b + \eta y_i$
- Now data only in dot products

Kernels

- Duality lets us remap to <u>many</u> more features!
- Let $\phi : \mathbb{R}^\ell \rightarrow F$ be nonlinear map of f.v.s, so
$$h(\mathbf{x}) = \text{sgn}\left(\eta \sum_{i=1}^{N} \alpha_i\, y_i\, \left(\phi\left(\mathbf{x}_i\right) \cdot \phi\left(\mathbf{x}\right)\right) + b\right)$$
- (Update "If" statement in dual algorithm)
- Can we compute $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$ <u>without</u> evaluating $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x})$? <u>YES!</u>
- $\mathbf{x} = [x_1, x_2]$, $\mathbf{z} = [z_1, z_2]$:
$$\begin{aligned}(\mathbf{x} \cdot \mathbf{z})^2 &= (x_1\, z_1 + x_2\, z_2)^2 \\ &= x_1^2\, z_1^2 + x_2^2\, z_2^2 + 2\, x_1\, x_2\, z_1\, z_2 \\ &= \underbrace{\left[x_1^2, x_2^2, \sqrt{2}\, x_1\, x_2\right]}_{\phi(\mathbf{x})} \cdot \left[z_1^2, z_2^2, \sqrt{2}\, z_1\, z_2\right]\end{aligned}$$
- LHS requires 2 mults + 1 squaring to compute, RHS takes 3 mults
- In general, $(\mathbf{x} \cdot \mathbf{z})^d$ takes ℓ mults + 1 expon., vs. $\binom{\ell+d-1}{d} \geq \left(\frac{\ell+d-1}{d}\right)^d$ mults if compute ϕ first

Kernels
(cont'd)

- In general, a kernel is a function K such that $\forall\, \mathbf{x}, \mathbf{z},\ K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ for some mapping $\phi(\cdot)$
- Typically start with kernel and take the feature mapping that it yields
- E.g. Let $\ell = 1, \mathbf{x} = x, \mathbf{z} = z$, $K(x, z) = \sin(x - z)$
- By Fourier expansion,

$$\sin(x - z) = a_0 + \sum_{n=1}^{\infty} a_n \sin(n\, x) \sin(n\, z) + \sum_{n=1}^{\infty} a_n \cos(n\, x) \cos(n\, z)$$

  for Fourier coeficients $a_0, a_1, \ldots$

- This is the dot product of two infinite sequences of nonlinear functions:

$$\{\phi_i(x)\}_{i=0}^{\infty} = [1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots]$$

- I.e. there are an infinite number of features in this remapped space!

Support Vector Machines

Finding a Hyperplane

- Can show [Cristianini & Shawe-Taylor] that if data linearly separable in remapped space, then get maximum margin classifier by minimizing $\mathbf{w} \cdot \mathbf{w}$ subject to $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \geq 1$
- Can reformulate this into a convex quadratic program, which can be solved optimally, i.e. won't encounter local optima
- Can always find a kernel that will make training set linearly separable, but beware of choosing a kernel that is too powerful (overfitting)
- If kernel doesn't separate, can optimize subject to $y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \geq 1 - \xi_i$, where ξ_i are slack variables that soften the margin (can still solve optimally)
- If number of training vectors is very large, may opt to approximately solve these problems to save time and space
- Use e.g. gradient ascent and sequential minimal optimization (SMO) [Cristianini & Shawe-Taylor]
- When done, can throw out non-SVs

What's Next?

- Core material:
  - More on kernels (2.1–2.3)
  - Loss (error) functions (3.1–3.2)
  - Statistical learning theory (5.1–5.2)
  - Convex optimization (6.1–6.3)
  - Pattern recognition with SVMs (7)
- Advanced material:
  - Implementation issues (10)
  - Kernel design (13)
  - Regularization (4)
  - Bayesian kernels (16)
  - More depth on 3 and 5
  - Others?