HTF: Ch 4.5, 12

B: Chapter (6), 7, E

Support Vector Machines

R Greiner

Much of this is taken from
Andrew W. Moore; CMU

+ Nello Cristianini, Ron Meir, Ron Parr

A Little History

- Support Vector Machines (SVM)
  - introduced in COLT-92
  - greatly developed since then
- Now, a large and diverse community:
  - machine learning
  - optimization
  - statistics
  - neural networks
  - functional analysis, etc etc etc.
- Successful applications in many fields (bioinformatics, text, handwriting recognition, etc)
- Kernel Machines: large class of learning algs
  - SVM = a particular instance
- http://www.kernel-machines.org

Books

Books on SVMs and Other Kernel Machines

last modified 2007-01-31 12:45

- Vladimir Vapnik. **Estimation of Dependences Based on Empirical Data**. Springer Verlag, 2006, 2nd edition.

  The second edition of Vapnik's classic on learning theory, including several new chapters on the history of events and on non-inductive inference.

- Grace Wahba. **Spline Models for Observational Data**. SIAM CBMS-NSF Regional Conference Series in Applied Mathematics vol. 59, Philadelphia, 1990.

  Discusses (reproducing) kernel methods in nonparametric regression. Not easy reading for machine learning researchers, but containing fundamental material about precedents of today's kernel machines (169 pages, $33.5).

- Vladimir Vapnik. **The Nature of Statistical Learning Theory**. Springer, NY, 1995.

  An overview of statistical learning theory, containing no proofs, but most of the crucial theorems and milestones of learning theory. With a detailed chapter on SVMs for pattern recognition and regression (1st edition: 188 pages, $65; 2nd edition: 304 pages, $70).

- Vladimir Vapnik. **Statistical Learning Theory**. Wiley, NY, 1998.

  The comprehensive treatment of statistical learning theory, including a large amount of material on SVMs (768 pages, $120).

- Bernhard Schölkopf, Chris Burges, and Alex Smola (eds). **Advances in Kernel Methods - Support Vector Learning** MIT Press, Cambridge, MA, 1999.

  A collection of articles written by experts in the field. Includes an introductory tutorial, overviews of the theory of SVMs, contributions on novel algorithms, and three chapters on SVM implementations (392 pages, $53).

- Nello Cristianini and John Shawe-Taylor. **An Introduction to Support Vector Machines**. Cambridge University Pres: Cambridge, UK, 2000.

  An introduction to SVMs which is concise yet comprehensive in its description of the theoretical foundations of large margin algorithms (189 pages, $45).

- Alex Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans (eds). **Advances in Large Margin Classifier:** MIT Press, Cambridge, MA, 2000.

  A collection of articles dealing with one of the main ideas of SVMs, large margin regularization. Contains an introduction, articles on new kernels, SVMs, and boosting algorithms (422 pages, $45).

Software

Kernel-Machines.Org software links

last modified
2007-01-31 12:47

Gaussian Processes

- GP Demo. Demonstration Software for Gaussian Processes by David MacKay (in OCTAVE).
- gpml. Matlab implementations of algorithms from Rasmussen & Williams "Gaussian Processes for Machine Learning the MIT Press 2006.
- LS-SVMlab. Matlab/C toolbox for least squares support vector machines.
- MAP-1. Package for MAP estimation by Carl Rasmussen.
- MC-1. Package for MAP estimation by Carl Rasmussen.
- Flexible Bayesian Modelling. Package by Radford Neal. It includes programs for Neural Networks, Gaussian Processes, and Mixture Models.
- Netlab. Matlab toolbox including Gaussian Process Regression, Mixture models and Neural Networks.
- Sparse Gaussian Processes. Matlab Toolbox for Sparse Inference using Gaussian Processes.
- Tpros and Cpros. Package by Mark Gibbs.

Mathematical Programming

- CPLEX. Barrier/QP Solver.
- LOQO. Linear and Quadratic Optimization Package by Robert Vanderbei.
- MINOS. Linear and Quadratic Solver.

Support Vectors

- Nearest Point Algorithm. by Sathiya Keerthi (in FORTRAN).
- SVM Java Applet. by Chris Burges et al.
- BSVM. A decomposition method for bound-constrained SVM formulations.
- QP SVM Classification and Regression. Fortran Implementation.
- CLISP/LibSVM. A module for using LibSVM from GNU CLISP (an ANSI Common Lisp implementation).
- Chunking Code. by C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola at Royal Holloway, AT&T, and GMD FIRST (Documentation).
- cSVM. SVM for classification tasks with model selection.
- 2D SVM Interactive Demo. runs under Matlab 6 and produces nice pictures - useful for courses.
- DTREG. by Phillip H. Sherrod.
- Interior Point Optimizer for SVM Pattern Recognition. by Alex Smola.
- Equbits Foresight. Commerical SVM based Classification and Regression Application Designed for Drug Discovery.
- Gini-SVM. A multi-class Probabilistic regression software for large data sets.
- GiniSVM. Multi-class SVM Probability regression package.
- Gist. Gist contains software tools for support vector machine classification and for kernel principal components analysis. The SVM portion of Gist is available via an interactive web server.
- Parallel GPDT. Parallel and serial training of SVM.

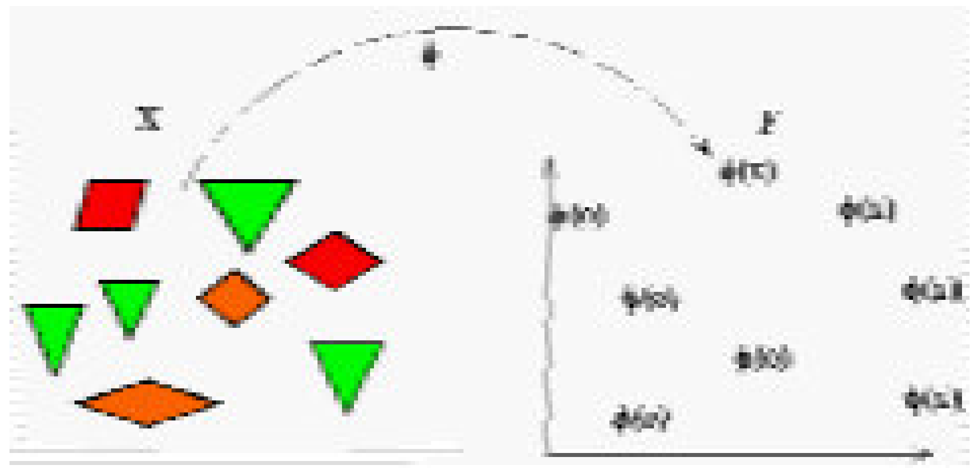
Preliminaries

- Goal:
  - detect and exploit complex patterns in data
  - eg: by clustering, classifying, ranking, cleaning, etc.
- Challenges:
  1. Representing complex patterns
  2. Excluding spurious (unstable) patterns (= overfitting)
- #1 is computational problem
  #2 is statistical problem

Basic Idea

- Kernel Methods work by embedding the data into a vector space,

  and by detecting linear relations in that space
- Main tools:
  Convex Optimization, Statistical Learning Theory, Functional Analysis



Outline

- Foundations
  - Primal/Dual; Lagrange
  - Perceptron Factoids
  - Dual Representation
- "Best" Linear Separator: Max Margin!
- Coping with Non-Linearly Separated Data
- Kernel Trick
- Regression

Background: LP

- **Linear Programming**
  - Given $\mathbf{c}$, $\mathbf{A}$, $\mathbf{b}$
    find $\mathbf{w}^* = \text{argmax}_{\mathbf{w}}\ \mathbf{c} \times \mathbf{w}$
  - subject to
    - $\mathbf{a_i}^{\mathbf{T}}\ \mathbf{w} \le b_i$ for i = 1 .. m $\qquad \mathbf{A\ w} \le \mathbf{b}$
    - $w_j \ge 0$ for j = 1 .. n

- ∃ fast algorithms for solving linear programs …including
  - simplex algorithm
  - Karmarkar's algorithm

Duality

- Given **c**, **A**, **b**, **...**

- **Primal**
  - find $\mathbf{w}^* = \text{argmax}_{\mathbf{w}}\ \mathbf{c} \times \mathbf{w}$
  - subject to
    - $\mathbf{A}\,\mathbf{w} \leq \mathbf{b}$
    - $w_j \geq 0$ for j = 1..n

- **Equivalent Dual**
  - find $\mathbf{y}^* = \text{argmin}_{\mathbf{y}}\ \mathbf{b} \times \mathbf{y}$
  - subject to
    - $\mathbf{A}^T\,\mathbf{y} \geq \mathbf{c}$
    - $y_i \geq 0$ for i = 1..m

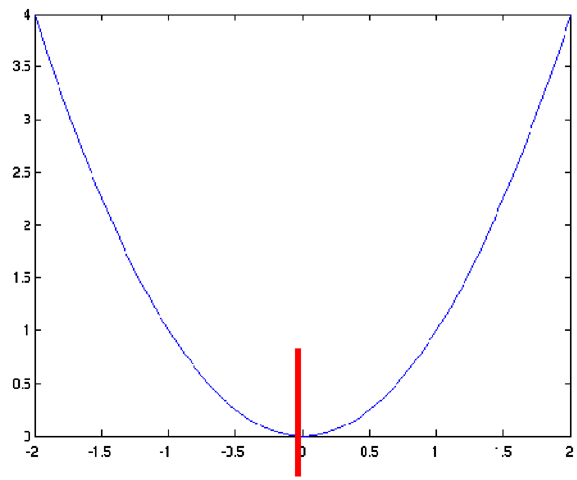
Strong duality result:
If $\mathbf{w}^*$ is an optimal solution for the primal, then the dual has optimal solution $\mathbf{y}^*$ s.t:
$\mathbf{c}^T\mathbf{w}^* = \mathbf{b}^T\mathbf{y}^*$

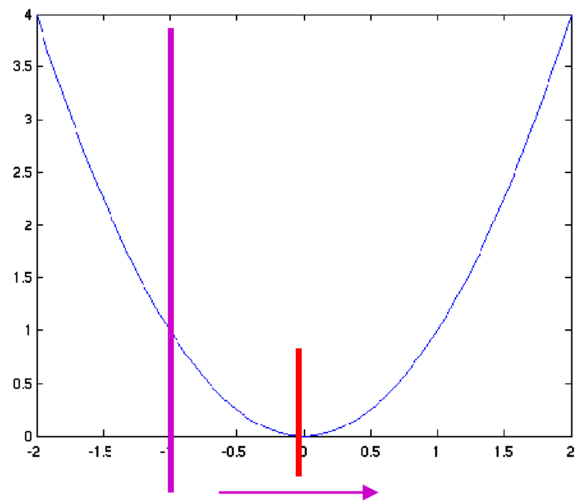
Constrained optimization

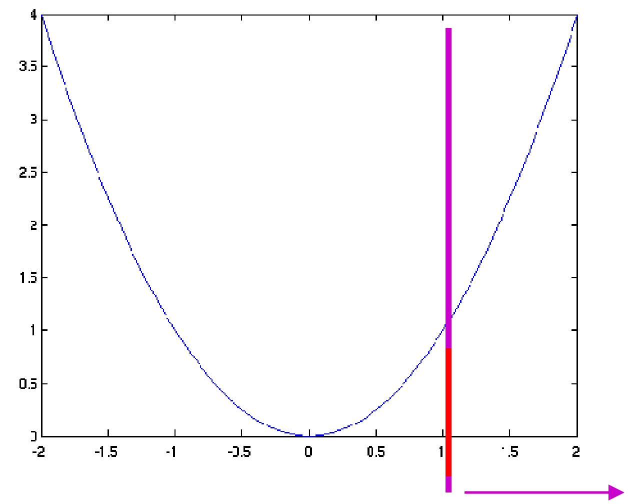
$\min_x x^2$

$\min_x x^2$
$s.t.\ x \geq -1$

$\min_x x^2$
$s.t.\ x \geq 1$







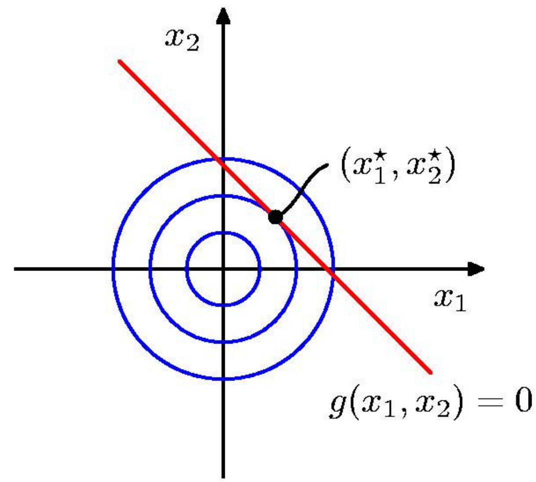
min @ x=0

min @ x=0

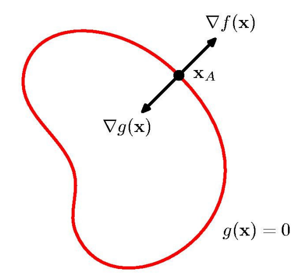
Constraint irrelevant

min @ x=+1

Lagrange Multiplier 101

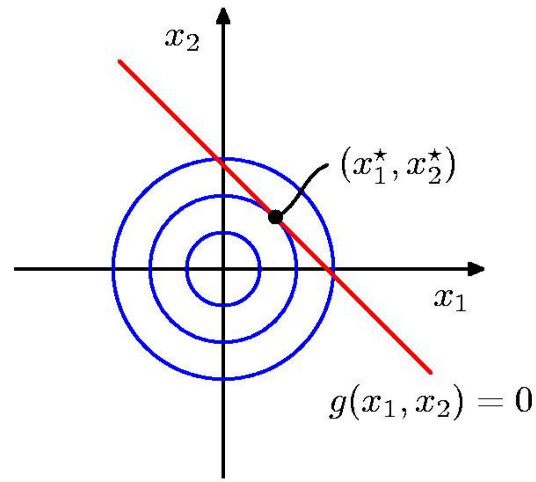


- Challenge: $\text{argmax}_{\mathbf{x}}\ f(\mathbf{x}) = -x_1^2 - x_2^2$
  s.t. $g(\mathbf{x}) = x_1 + x_2 - 1 = 0$
- Consider optimum $\mathbf{x}^* = (x^*_1, x^*_2)$
  - On $g(\mathbf{x}) = 0$ line, by def'n!
  - Note $\nabla f(\mathbf{x}^*) \perp g(\mathbf{x}^*)$
    - Otherwise, could walk along $g(\mathbf{x})=0$ to get larger $f(\mathbf{x})$ value



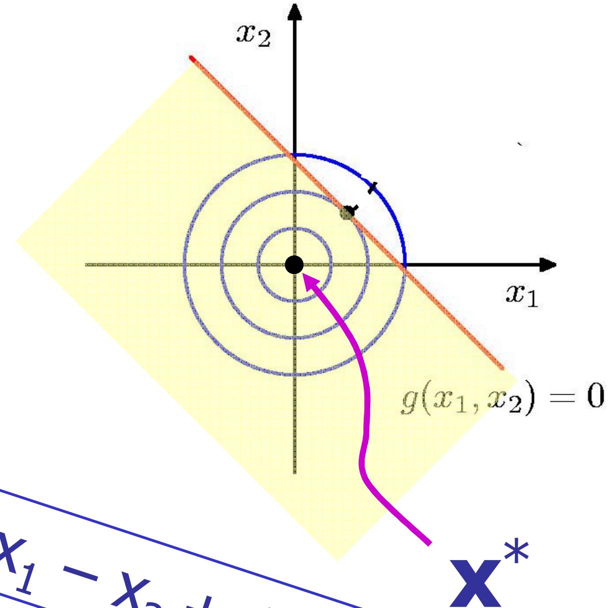
- Hence, $\nabla f(\mathbf{x}^*)$ is || to $\nabla g(\mathbf{x}^*)$
  $\Rightarrow\ \exists\lambda$ s.t. $\nabla f(\mathbf{x}^*) + \lambda \nabla g(\mathbf{x}^*) = 0$
- Write $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
- Note $\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x})$ … $= 0$ @ $\mathbf{x}^*$
  $\nabla_{\lambda} L(\mathbf{x}, \lambda) = g(\mathbf{x})$ … $= 0 \Rightarrow g(\mathbf{x}) = 0$ satisfied

Lagrange Multiplier 101



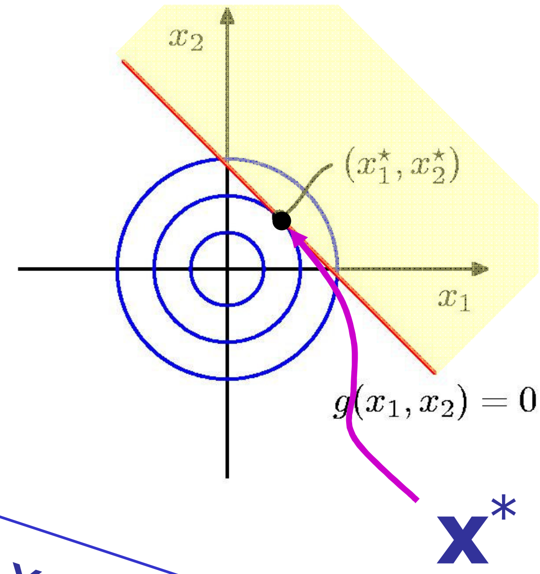
- Challenge: $\text{argmax}_{\mathbf{x}}\ f(\mathbf{x}) = -x_1^2 - x_2^2$
  s.t. $g(\mathbf{x}) = x_1 + x_2 - 1 = 0$
- $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
- $\nabla_{x1} L(\mathbf{x}, \lambda) = -2\, x_1 + \lambda \quad = 0$
  $\nabla_{x2} L(\mathbf{x}, \lambda) = -2\, x_2 + \lambda \quad = 0$
  $\nabla_{\lambda} L(\mathbf{x}, \lambda) = x_1 + x_2 - 1 \quad = 0$
  - Soln: $x_1 = x_2 = ½$
  - Also $\lambda = 1$ (who cares…)

Lagrange Multiplier 102 InEqualities…



- $\text{argmax}_{\mathbf{x}}\ f(\mathbf{x})$ s.t. $g(\mathbf{x}) \geq 0$
- Two cases:
  - At optimal $\mathbf{x}^*$, $g(\mathbf{x}^*) > 0$
    - "inactive constraint"
    - Just need $\nabla f(\mathbf{x}) = 0$
    - … corresponds to $\nabla f(\mathbf{x}) + \lambda\ \nabla g(\mathbf{x}) = 0$ when $\lambda=0$

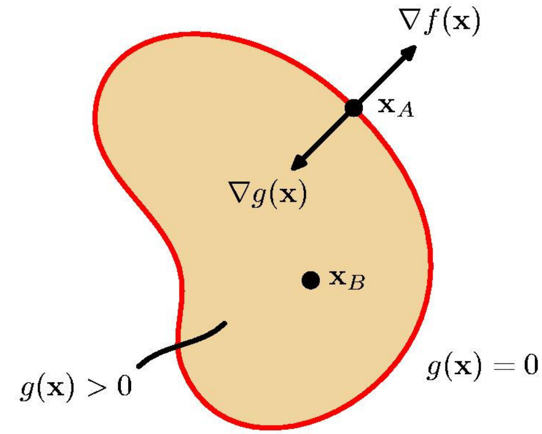
Lagrange Multiplier 102 InEqualities…



- $\arg\max_{\mathbf{x}} f(\mathbf{x})$ s.t. $g(\mathbf{x}) \geq 0$
- Two cases:
  - At optimal $\mathbf{x}^*$, $g(\mathbf{x}^*) > 0$
    - "inactive constraint"
    - Just need $\nabla f(\mathbf{x}) = 0$
    - … corresponds to $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ when $\lambda=0$
  - At optimal $\mathbf{x}^*$, $g(\mathbf{x}^*) = 0$
    - "active constraint"
    - = earlier case… need $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0$ with $\lambda \neq 0$
    - Actually… need $\lambda > 0$ as need $\nabla f(\mathbf{x})$ oriented AWAY from $g(x)>0$ region…
      $\nabla f(\mathbf{x}^*) = - \lambda \nabla g(\mathbf{x}^*)$ for some $\lambda > 0$

$g(x) = x_1 + x_2 - 1 \geq 0$

Lagrange Multiplier 102



- $\text{argmax}_{\mathbf{x}}\ f(\mathbf{x})$ s.t. $g(\mathbf{x}) \geq 0$
- At optimal $\mathbf{x}^*$, $g(\mathbf{x}^*) > 0$
  - $\nabla f(\mathbf{x}^*) + \lambda\ \nabla g(\mathbf{x}^*) = 0$ with $\lambda=0$
- At optimal $\mathbf{x}^*$, $g(\mathbf{x}^*) = 0$
  - $\nabla f(\mathbf{x}^*) + \lambda\ \nabla g(\mathbf{x}^*) = 0$ with $\lambda > 0$
- Either way… $\lambda\ g(\mathbf{x}^*) = 0$
- Summary:
  - $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
  - Solve $\nabla_{xi} L(\mathbf{x}, \lambda) = 0$ $\nabla_{\lambda} L(\mathbf{x}, \lambda) = 0$
  - s.t. $g(\mathbf{x}) \geq 0$ $\lambda \geq 0$ $\lambda\ g(\mathbf{x}) = 0$

Karush-Kuhn-Tucker (KKT):
If $g(x) > 0$, then $\lambda=0$

KKT Conditions: Inequality Case

- Karush-Kuhn-Tucker Theorem:
  If
  - function $f(x)$ has a minimum at x^* in the feasible set. and
  - $\nabla f(x^*)$ and $\nabla g_i(x^*)$, i=1,2,…,m exist,

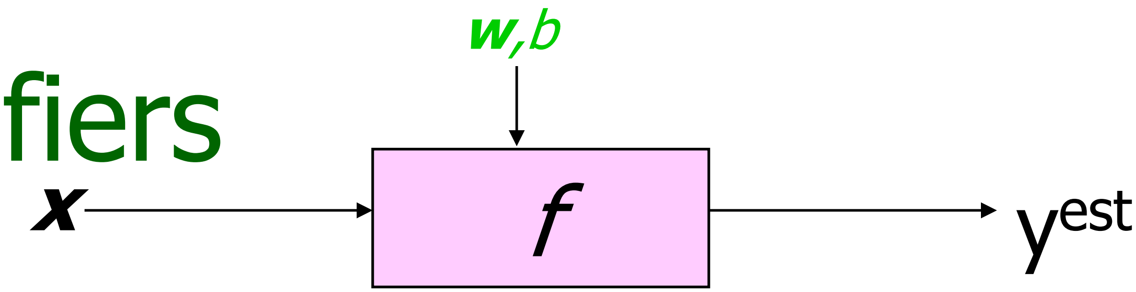
  then ∃ m-dimensional vector λ such that

$$\lambda \geq 0$$
$$\nabla f(x^*) - \Sigma_{i=1}^{m} \lambda_i \nabla g_i(x^*) = 0$$
$$\lambda_i \, [g_i(x^*) - b_i] = 0, \quad \text{for } i=1,2,\ldots,m.$$

- Each such (x^*, λ) is a KKT point;
  λ is the Dual Vector aka the Lagrange Multipliers.

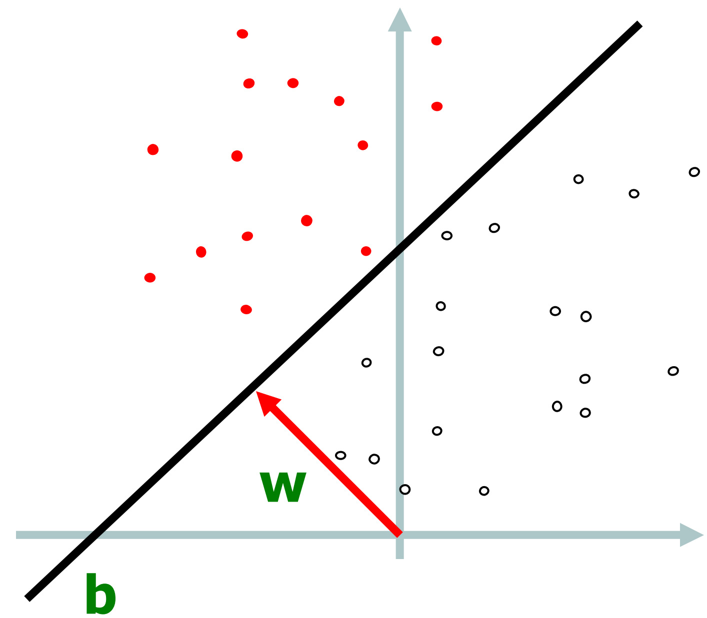
- These conditions are sufficient if dealing with a convex programming problem

Linear Classifiers



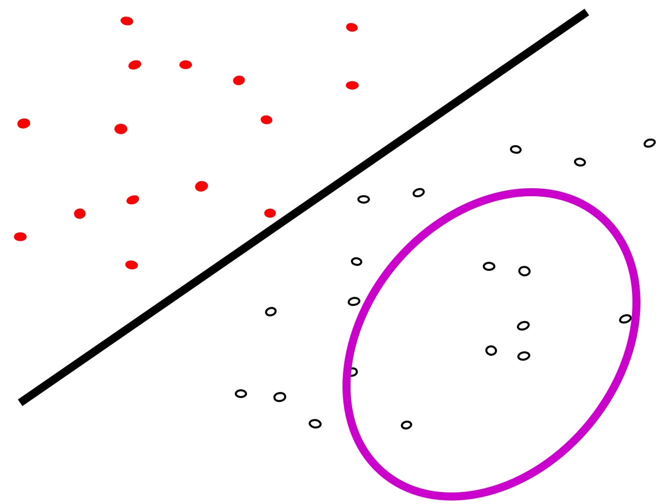
Input space	$\boldsymbol{x} \in \boldsymbol{X}$
Output space	$y \in Y = \{+1, -1\}$
Real-valued fn:	f: $\mathbf{X} \rightarrow \Re$
Training Set	S = { [$\mathbf{x}_1$,y_1], … [$\mathbf{x}_m$,y_m] }
Dot product	$\langle \mathbf{x}, \mathbf{z} \rangle$

$$f(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$



Perceptron Training Rule

Initialize $\mathbf{w} = 0$
Do until bored
  Predict "+" iff $\mathbf{w} \cdot \mathbf{x} > 0$
    else "–"
  Mistake on $y = +1$: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}$
  Mistake on $y = -1$: $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}$



$\mathbf{w} \leftarrow \mathbf{w} + y\ \mathbf{x}$

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

- … only uses Informative Points (mistake driven)
- Coefficient of point reflects its 'difficulty'

Mistake Bound Theorem

From earlier…

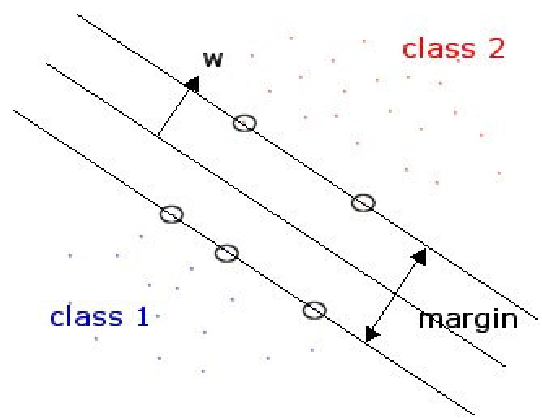
Theorem: [Rosenblatt 1960]

If data is consistent w/some linear threshold **w**,

then number of mistakes is $\leq (1/\Delta)^2$,

where $$\Delta = \min_x \frac{|\mathbf{w} \cdot x|}{|\mathbf{w}| \times |x|}$$ ≈ margin

- Δ measures "wiggle room" available:
  If |x| = 1, then Δ is max, over all consistent planes, of minimum distance of example to that plane
- w is $\perp$ to separator, as w · x = 0 at boundary
- So |**w** · **x**| is projection of **x** onto plane, PERPENDICULAR to boundary line
  … ie, is distance from **x** to that line (once normalized)



Dual Representation

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

$\Rightarrow$ can re-write decision function

$$f(\mathbf{x}) = \langle w, \mathbf{x} \rangle + b = \sum_i \alpha_i \, y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b$$

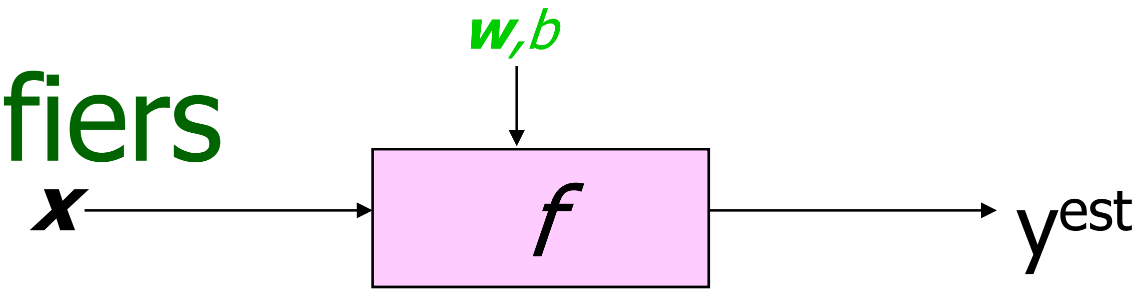
$\Rightarrow$ can re-write update rule:

If $y_j \sum_i \alpha_i \, y_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \le 0$

Then $\alpha_i \leftarrow \alpha_i + \eta$

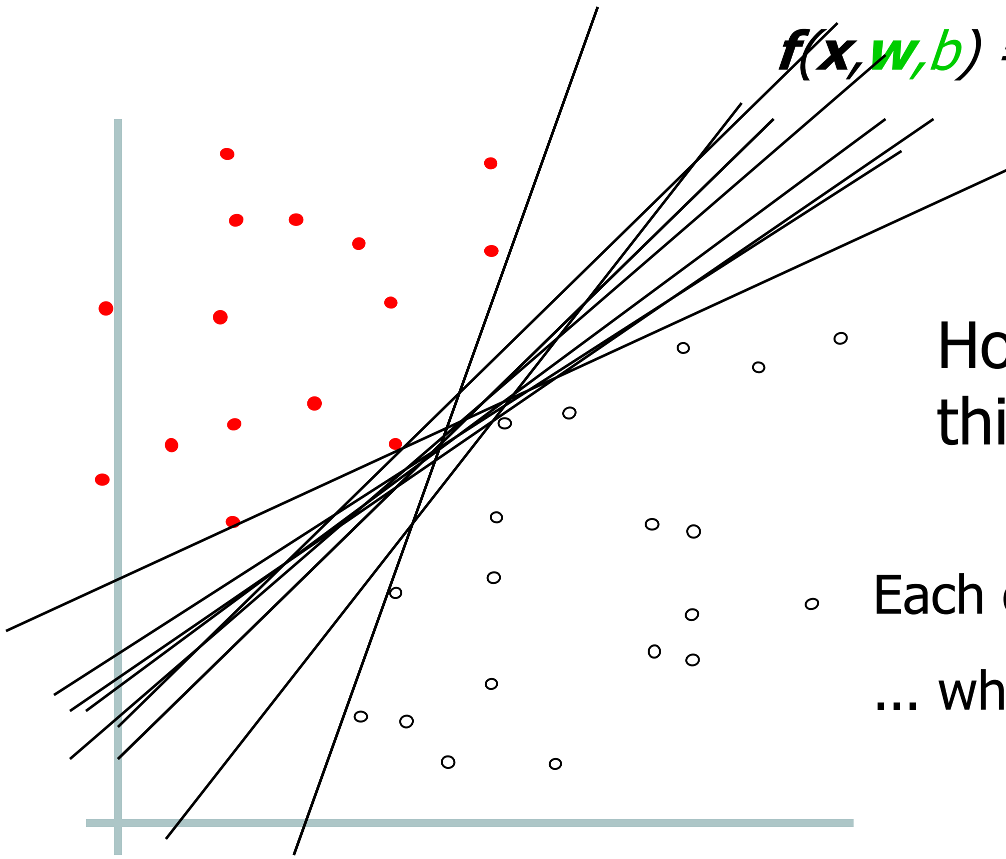
- In dual representation, data appears only inside dot products

Linear Classifiers



$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} + b)$

- • denotes +1
- ○ denotes −1

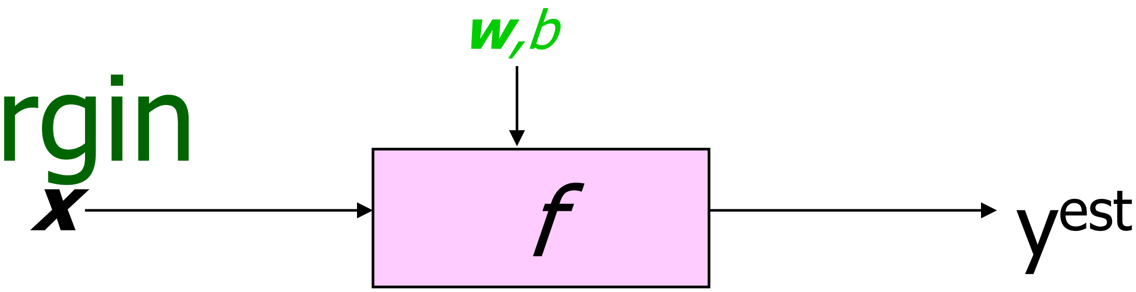


How to classify this data?

Each of these seems fine..

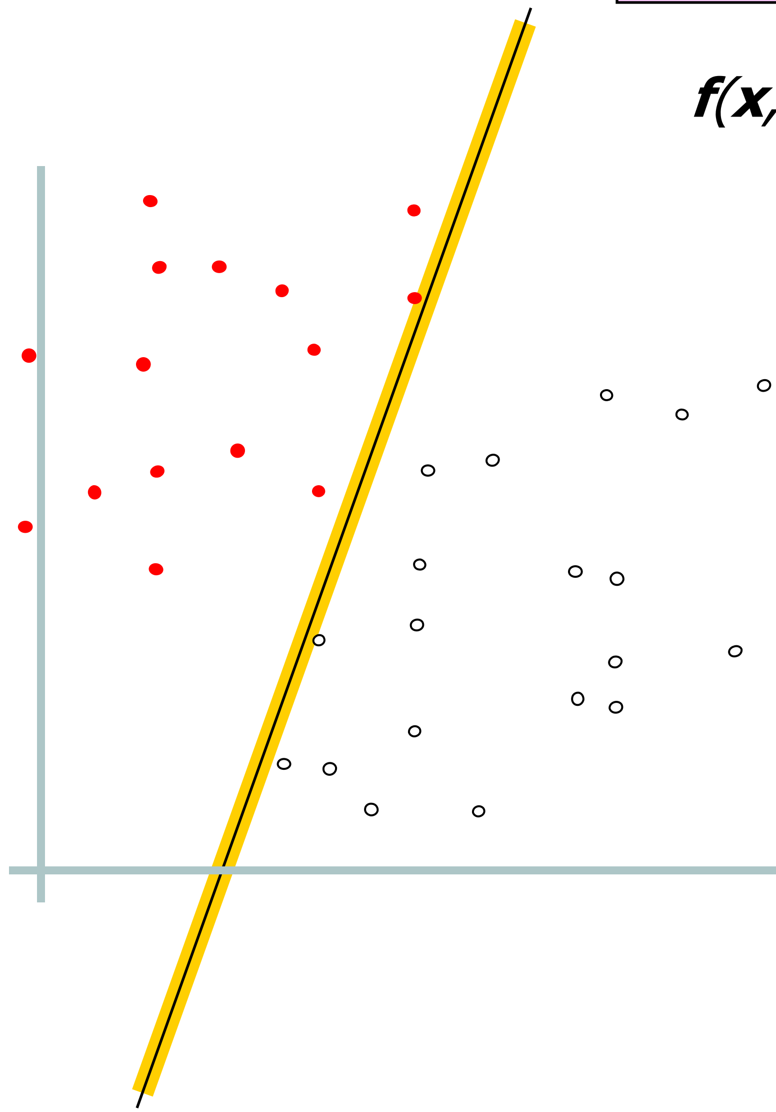
... which is best?

Classifier Margin



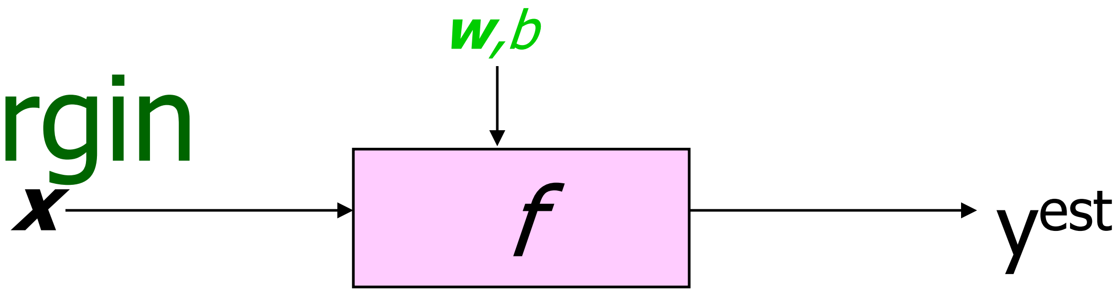
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w} \cdot \mathbf{x} + b)$

- • denotes +1
- ◦ denotes −1



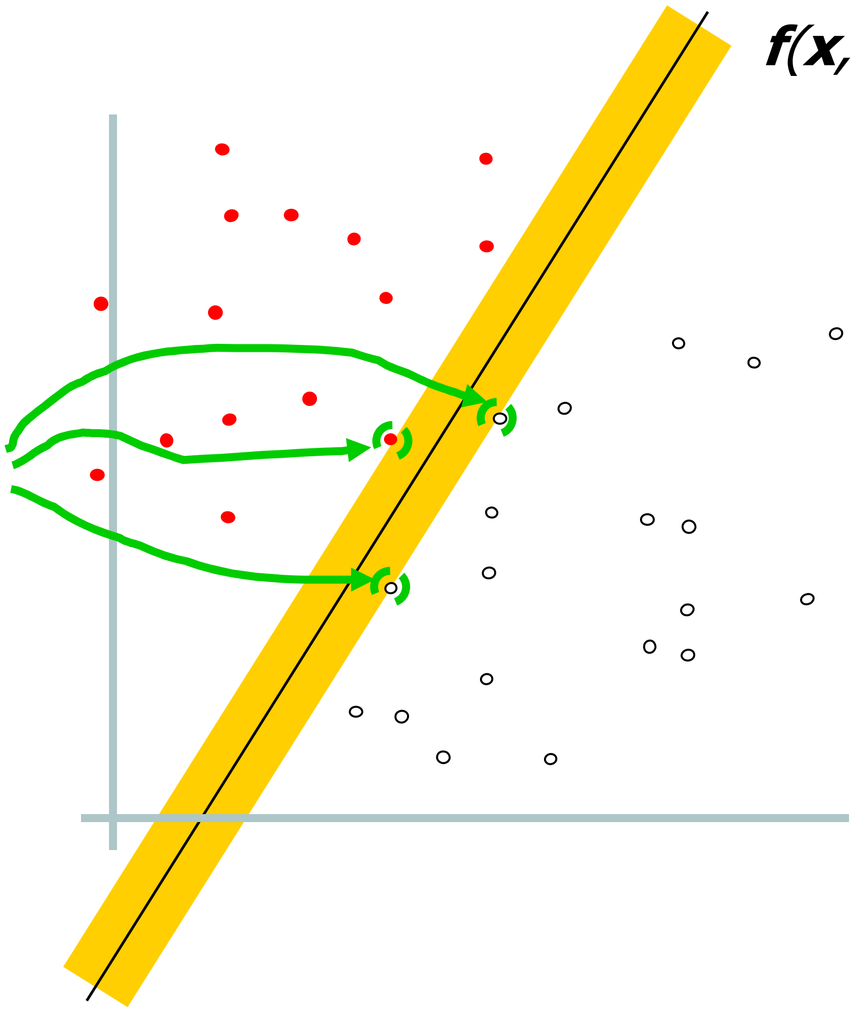
The margin of a linear classifier = the width that the boundary could be increased by, before hitting a datapoint

Maximum Margin



$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w} \cdot \mathbf{x} + b)$$

- • denotes +1
- ° denotes −1



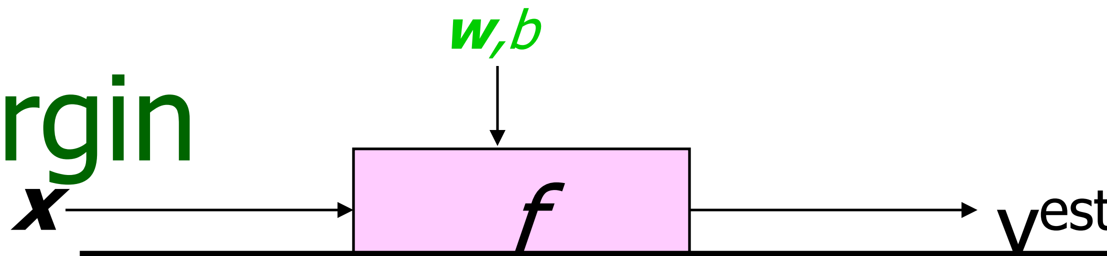
Support Vectors are datapoints that "touch" the margin

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

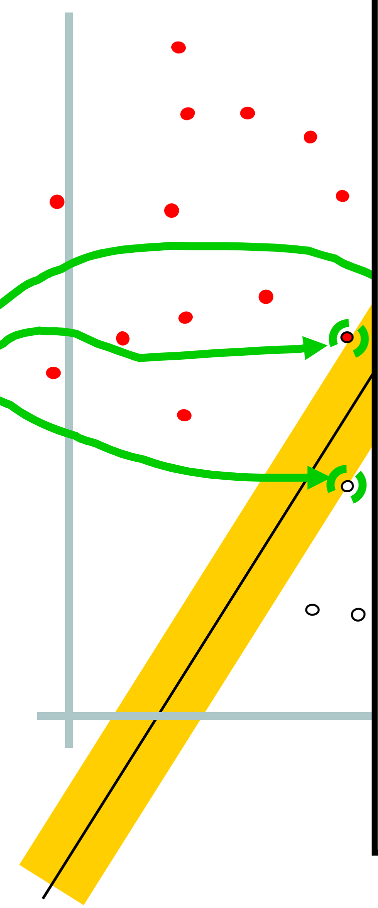
• the simplest kind of SVM – an LSVM

Linear SVM

Maximum Margin



- • denotes +1
- ◦ denotes −1



Support Vectors are datapoints that “touch” the margin

1. … this feels safest ...
2. If a small error in the location of the boundary (it’s been jolted in its perpendicular direction) this gives least chance of causing a misclassification.
3. LOO-CV is easy, since the model is immune to removal of any non-support-vector datapoints.
4. There’s some theory (using VC dimension) that is related to (but not the same as) the claim that this is a good thing.
5. Empirically it works *very very well*.

Goal of Max Margin Separator

Want a linear separator

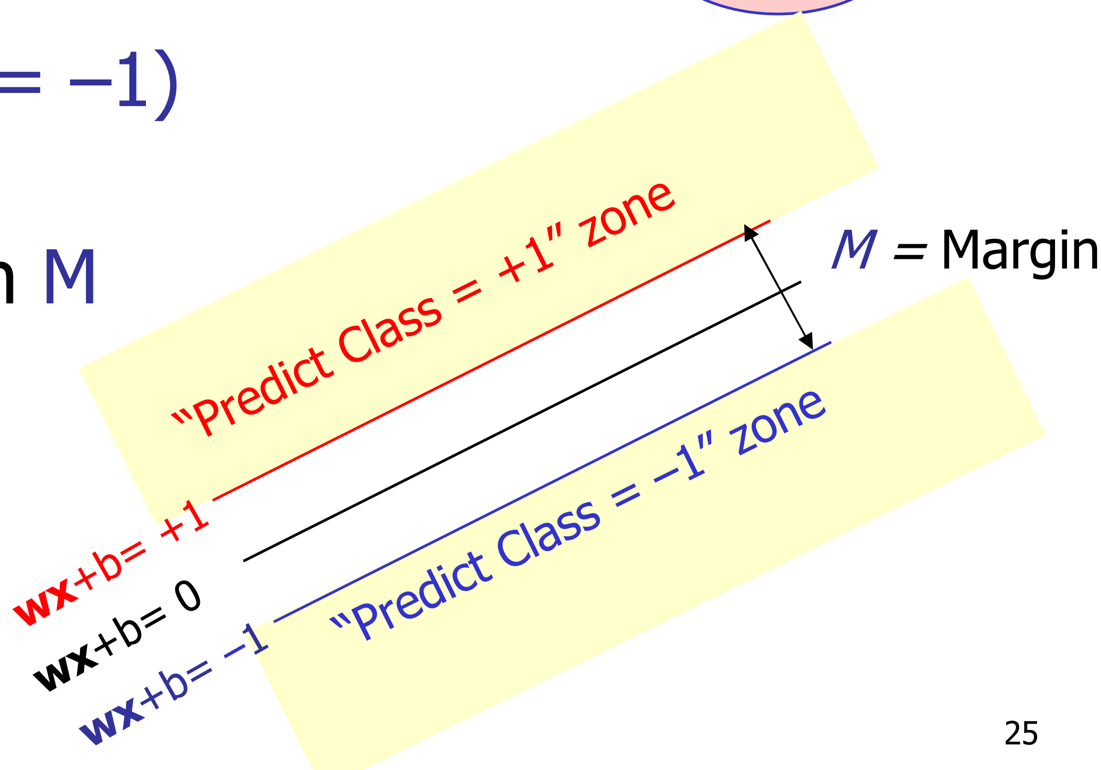
w, b for $y = \mathbf{w} \cdot \boldsymbol{x} + b$

s.t.

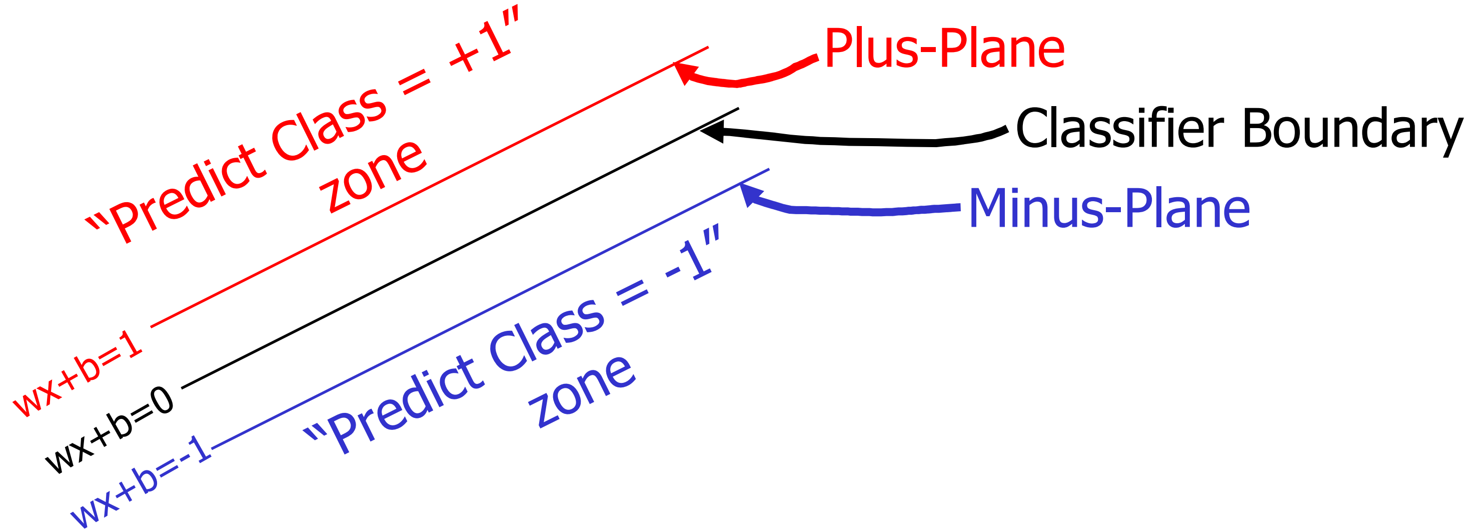
- For all +points $(\mathbf{x}_i, y_i=+1)$
  $\mathbf{w} \cdot \boldsymbol{x}_i + b \geq +1$
- For all –points $(\mathbf{x}_i, y_i= -1)$
  $\mathbf{w} \cdot \boldsymbol{x}_i + b \leq -1$
- Maximizes the margin M

Why 1?

Any >0 constant works, as scales. 1 is convenient…



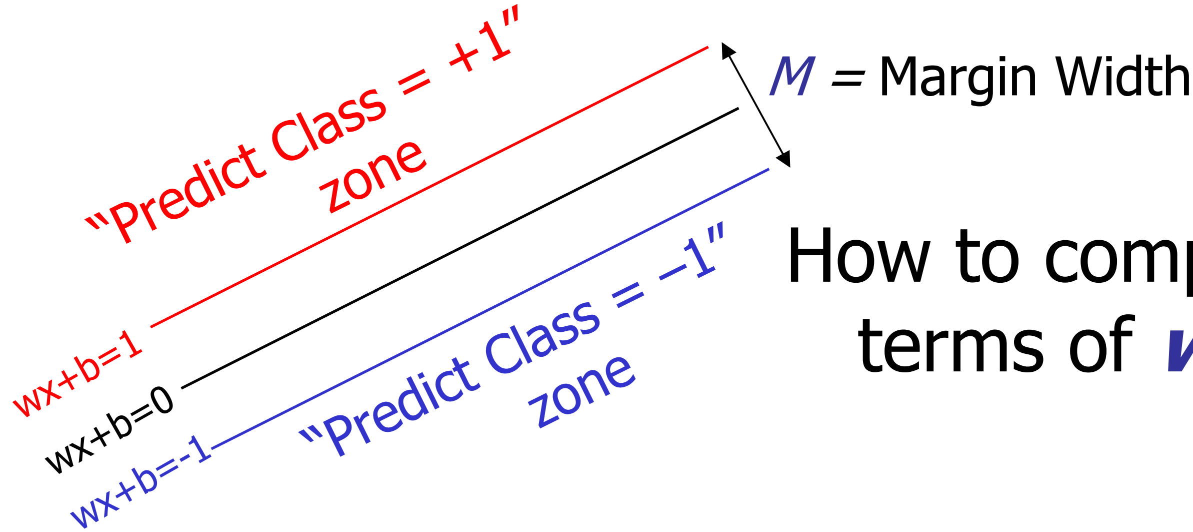
Specifying a Line and a Margin



- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Classify as..	+1	if	$\mathbf{w} \cdot \mathbf{x} + b \geq 1$
	−1	if	$\mathbf{w} \cdot \mathbf{x} + b \leq -1$
	Universe explodes	if	$-1 < \mathbf{w} \cdot \mathbf{x} + b < 1$

Computing the Margin Width



How to compute M in terms of $\mathbf{w}$ and b?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

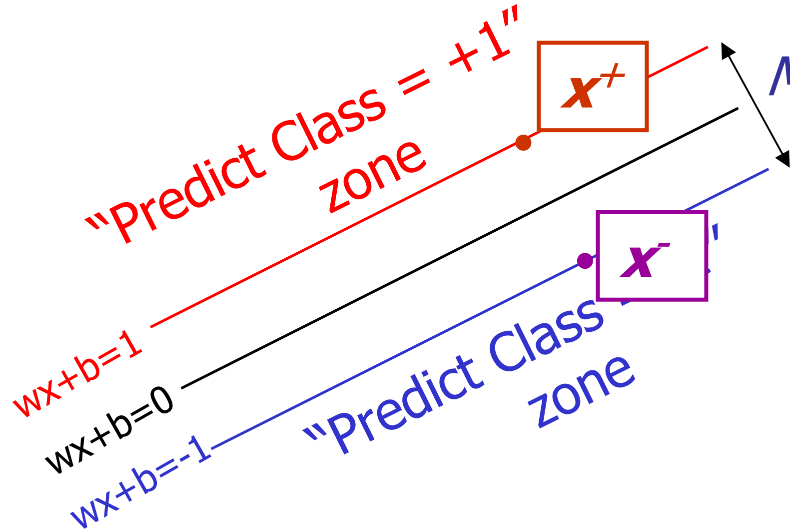
Claim: The vector **w** is perpendicular to the Plus Plane. Why?

- Definitions: "vector" ≡ "point"
- $\mathbf{x}_1$ perpendicular to $\mathbf{x}_2$ iff $\mathbf{x}_1 \cdot \mathbf{x}_2 = 0$

Let **u** and **v** be two vectors on the Plus Plane. What is $\mathbf{w} \cdot (\mathbf{u} - \mathbf{v})$?

w is also ⊥ Minus Plane

Computing the margin width



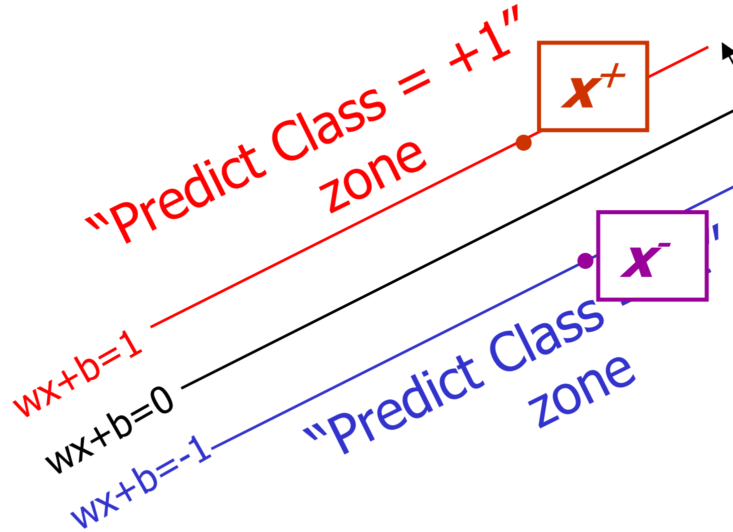
M = Margin Width

How to compute M in terms of $\mathbf{w}$ and b?

- Plus-plane = $\{\ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1\ \}$
- Minus-plane = $\{\ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1\ \}$
- The vector **w** is perpendicular to the Plus Plane
- $\mathbf{x}^-$ = any point on the minus plane
- $\mathbf{x}^+$ =the point in plus-plane closest to $\mathbf{x}^-$
- Claim: $\mathbf{x}^+ = \mathbf{x}^- + \lambda\, \mathbf{w}$ for some value of $\lambda \in \Re^+$. Why?

> Any location in $\Re^m$: not necessarily a datapoint

Computing the margin width



M = Margin Width

Line from x^- to x^+ is ⊥ to the planes.

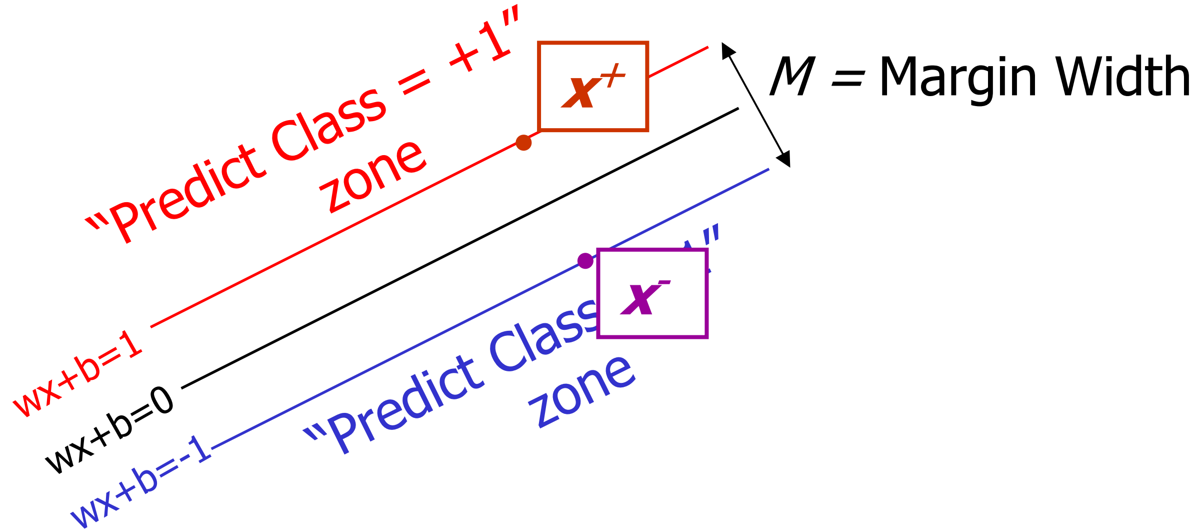
So to get from x^- to x^+, travel some distance in the direction of w

- Plus-plane = $\{ x : w \cdot x + b =$
- Minus-plane = $\{ x : w \cdot x + b = -1 \}$
- ular to the Plus Plane
- x^- = any poin the minus plane
- x^+ =the point n plus-plane closest to x^-
- Claim: $x^+ = x^- + \lambda w$ for some value of $\lambda \in \Re^+$. Why?

So ... $x^+ - x^- = \lambda w$

A ation in $\Re^m$: n ecessarily a atapoint

Computing the margin width

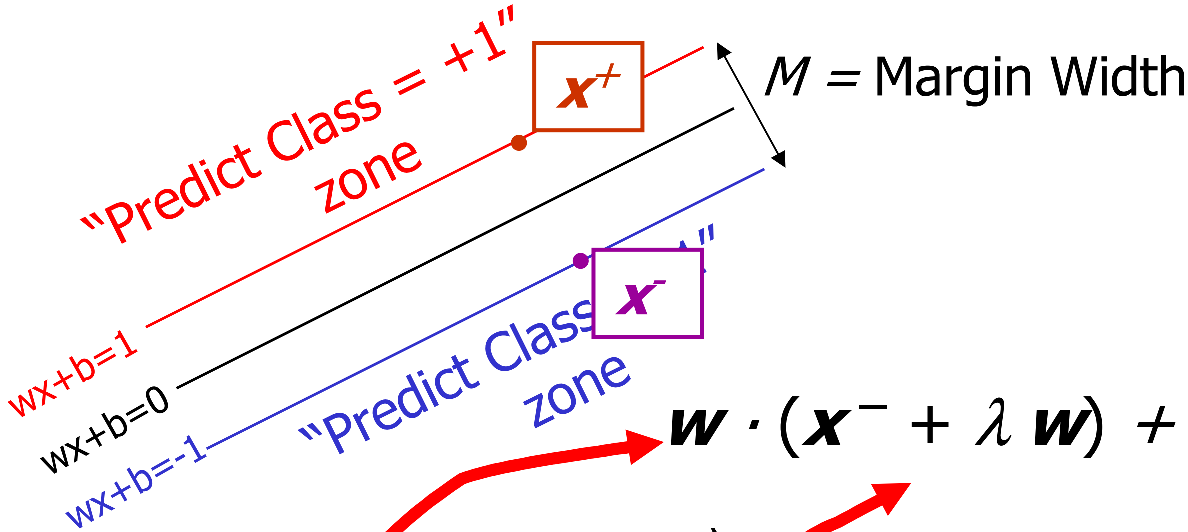


Given…

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ - \boldsymbol{x}^- = \lambda \, \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = M$

… easy to get M in terms of $\boldsymbol{w}$ and b

Computing the margin width



Given…

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$
- $\mathbf{x}^+ - \mathbf{x}^- = \lambda \, \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

… easy to get M in terms of $\mathbf{w}$ and b

$$\mathbf{w} \cdot (\mathbf{x}^- + \lambda \, \mathbf{w}) + b = 1$$

$$\Rightarrow$$

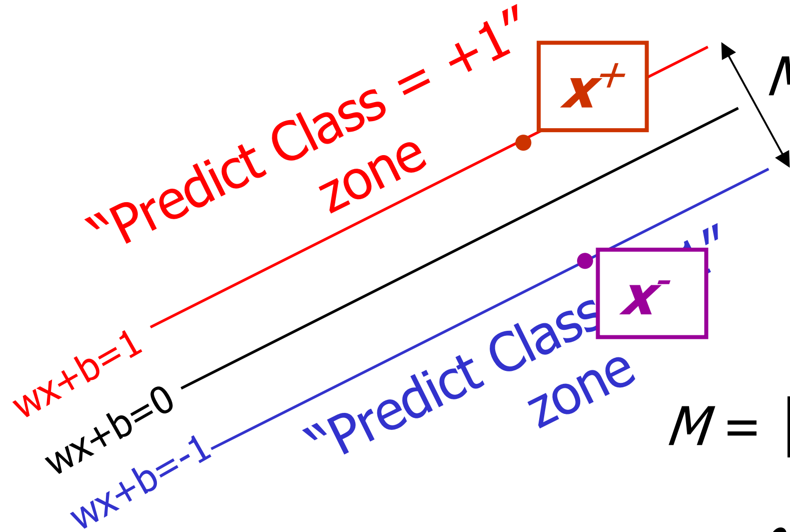
$$(\mathbf{w} \cdot \mathbf{x}^- + b) + \lambda \, \mathbf{w} \cdot \mathbf{w} = 1$$

$$\Rightarrow$$

$$-1 + \lambda \, \mathbf{w} \cdot \mathbf{w} = 1$$

$$\Rightarrow \quad \lambda = \frac{2}{\mathbf{w} \cdot \mathbf{w}}$$

Computing the margin width



M = Margin Width = $\dfrac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$

Given ...

- $\boldsymbol{w} \cdot \boldsymbol{x}^{+} + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^{-} + b = -1$
- $\boldsymbol{x}^{+} = \boldsymbol{x}^{-} + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^{+} - \boldsymbol{x}^{-}| = M$
- $\lambda = \dfrac{2}{\mathbf{w}\cdot\mathbf{w}}$

$$M = |\boldsymbol{x}^{+} - \boldsymbol{x}^{-}| = |\lambda \boldsymbol{w}| =$$

$$= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w}\cdot\mathbf{w}}$$

$$= \frac{2\sqrt{\mathbf{w}\cdot\mathbf{w}}}{\mathbf{w}\cdot\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$$

Yay! Just maximize $\dfrac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$

...≡ minimize **w·w**

Wait…OMG, I forgot the data!

Goal of Max Margin Separator

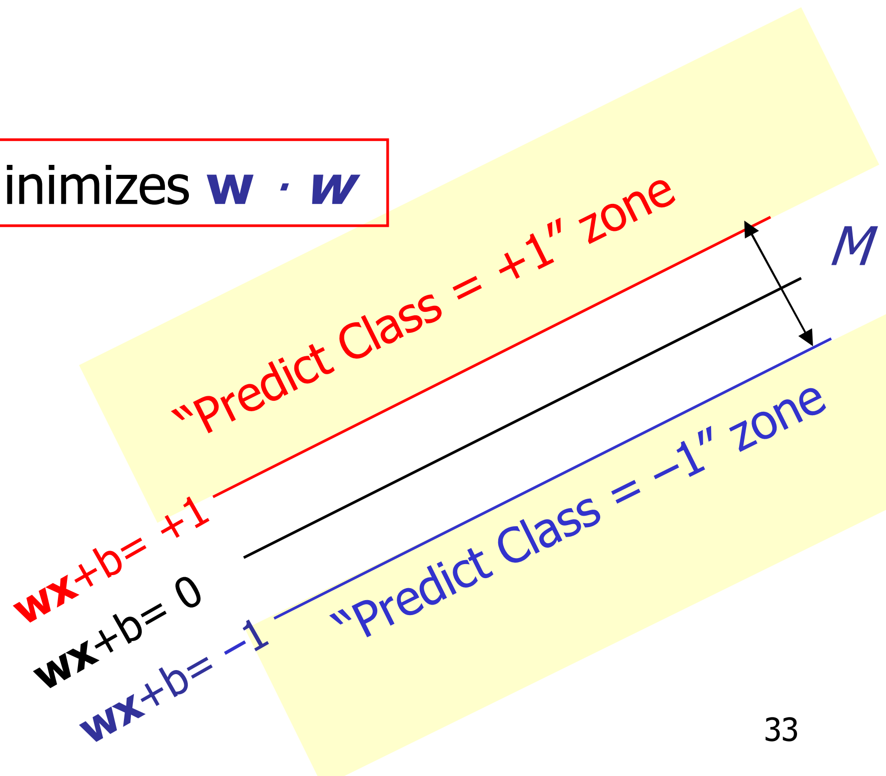
Want a linear separator

w, b for $y = \mathbf{w} \cdot \boldsymbol{x} + b$

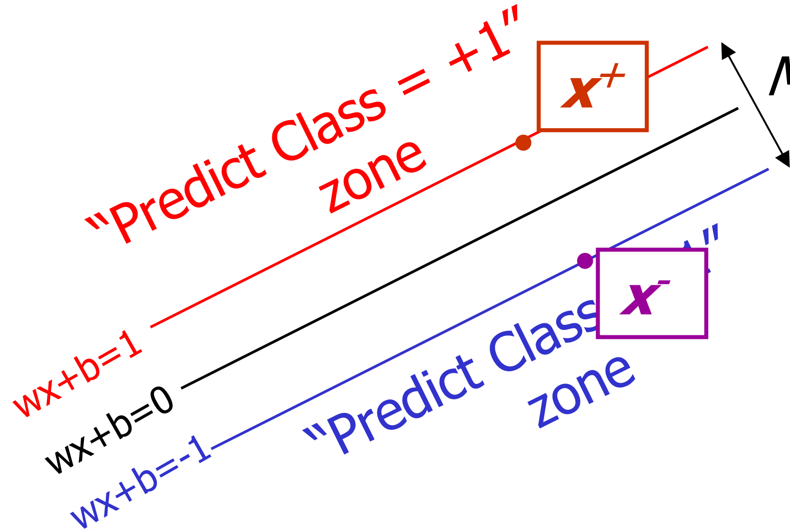
s.t.

- For all +points $(\mathbf{x}_i, y_i=+1)$
  $\mathbf{w} \cdot \boldsymbol{x}_i + b \geq +1$
- For all –points $(\mathbf{x}_i, y_i= -1)$
  $\mathbf{w} \cdot \boldsymbol{x}_i + b \leq -1$
- ~~Maximizes the margin~~ M

Minimizes $\mathbf{w} \cdot \boldsymbol{w}$



Learning the Maximum Margin Classifier



M = Margin Width = $\dfrac{2}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$

Given ***w*** and *b* we can

- Compute whether all data points are in correct half-planes
- Compute the width of the margin

But… need a program to search the space of **w**'s and *b*'s to find the widest margin that matches all the datapoints. *How?*

Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton's Method?

Rewrite Problem

Minimize $\frac{1}{2}\, \boldsymbol{w} \cdot \boldsymbol{w}$

s.t.

$$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \geq 1 \quad \text{if } y_k = 1$$
$$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \leq -1 \quad \text{if } y_k = -1$$

$$y_k(\boldsymbol{w} \cdot \boldsymbol{x}_k + b) - 1 \geq 0$$

Lagrange Multiplier

Equivalent optimization…

$$L(\mathbf{w}, b, \lambda) = \frac{1}{2}\, |\mathbf{w}|^2 - \sum_k \lambda_k \left[\, y_k\, (\mathbf{w} \cdot \mathbf{x}_k + b) - 1 \,\right]$$

$\min_{\mathbf{w},b}\ L(\mathbf{w}, b, \lambda)$

s.t. $\lambda \geq 0$

KKT:

$$y_k\, (\mathbf{w} \cdot \mathbf{x}_k + b) > 1 \ \Rightarrow\ \lambda_k = 0$$

Solving Constrained Optimization

$\min_{\mathbf{w},b} L(\mathbf{w}, b, \lambda) \quad = \frac{1}{2} |\mathbf{w}|^2 - \sum_k \lambda_k [\, y_k (\mathbf{w} \cdot \mathbf{x}_k + b) - 1 \,]$

s.t. $\lambda \geq 0$

Setting derivatives to 0…

- $\mathbf{w} = \sum_k \lambda_k \, y_k \, \mathbf{x}_k$
- $0 = \sum_k \lambda_k \, y_k$

Substitute back into L(..):

Find $\lambda \geq \mathbf{0}$ that minimizes

$$£(\lambda) = \sum_k \lambda_k - \frac{1}{2} \sum_k \sum_m \lambda_k \lambda_m \, y_k \, y_m (\mathbf{x}_k \cdot \mathbf{x}_m)$$

Learning via Quadratic Programming

- QP is a well-studied class of optimization alg's that
  - maximize a **quadratic function** of some real-valued variables
  - subject to **linear constraints**

- Popular ML approach:
  - Describe your learning problem as optimization…
  - …and give it to somebody else to solve!

Quadratic Programming – in general

Find $\arg\min_{\mathbf{w}} \quad c + \mathbf{d}^T \mathbf{w} + \frac{\mathbf{w}^T \mathbf{K} \mathbf{w}}{2}$ ← Quadratic criterion

Note $\boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^T \boldsymbol{x}$

Subject to

$$
\left.\begin{aligned}
a_{11}w_1 + a_{12}w_2 + \ldots + a_{1m}w_m &\le b_1 \\
a_{21}w_1 + a_{22}w_2 + \ldots + a_{2m}w_m &\le b_2 \\
&\vdots \\
a_{n1}w_1 + a_{n2}w_2 + \ldots + a_{nm}w_m &\le b_n
\end{aligned}\right\}
$$

n additional linear inequality constraints

and to

$$
\left.\begin{aligned}
a_{(n+1)1}w_1 + a_{(n+1)2}w_2 + \ldots + a_{(n+1)m}w_m &= b_{(n+1)} \\
a_{(n+2)1}w_1 + a_{(n+2)2}w_2 + \ldots + a_{(n+2)m}w_m &= b_{(n+2)} \\
&\vdots \\
a_{(n+e)1}w_1 + a_{(n+e)2}w_2 + \ldots + a_{(n+e)m}w_m &= b_{(n+e)}
\end{aligned}\right\}
$$

e additional linear equality constraints

Quadratic Programming – in general

Find $\underset{\mathbf{w}}{\arg\min} \quad c + \mathbf{d}^T\mathbf{w} + \frac{\mathbf{w}^T\mathbf{K}\mathbf{w}}{2}$ ← Quadratic criterion

Note ***w·x = w^{T}x***

Subject to $a_{11}w_1 + \ldots$

…additional linear inequality constraints

∃ alg's for finding such constrained quadratic optima **much more efficiently** and **reliably** than gradient ascent!

(But they are very fiddly… you probably don't want to write one yourself)

and to

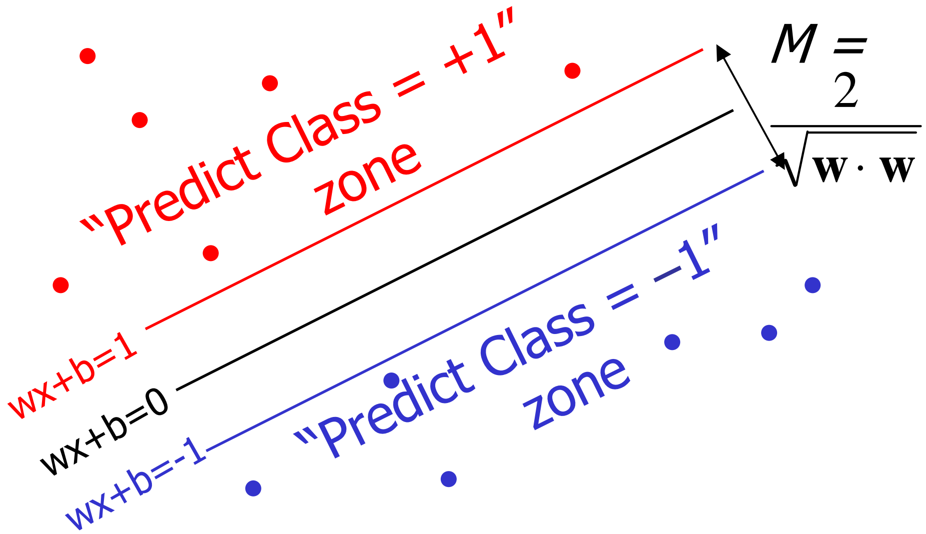
$\ldots = b_{(n+1)}$

$\ldots w_m = b_{(n+2)}$

$$a_{(n+e)1}w_1 + a_{(n+e)2}w_2 + \ldots + a_{(n+e)m}w_m = b_{(n+e)}$$

e additional linear equality constraints

Learning the Maximum Margin Classifier



Given guess of $\mathbf{w}$, b, can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

R datapoints, $\{[\mathbf{x}_k, y_k]\}$ where $y_k \in \{ +1, -1 \}$

What is quadratic optimization criterion?

Minimize $\mathbf{w}\cdot\mathbf{w}$

How many constraints? R

What should they be?

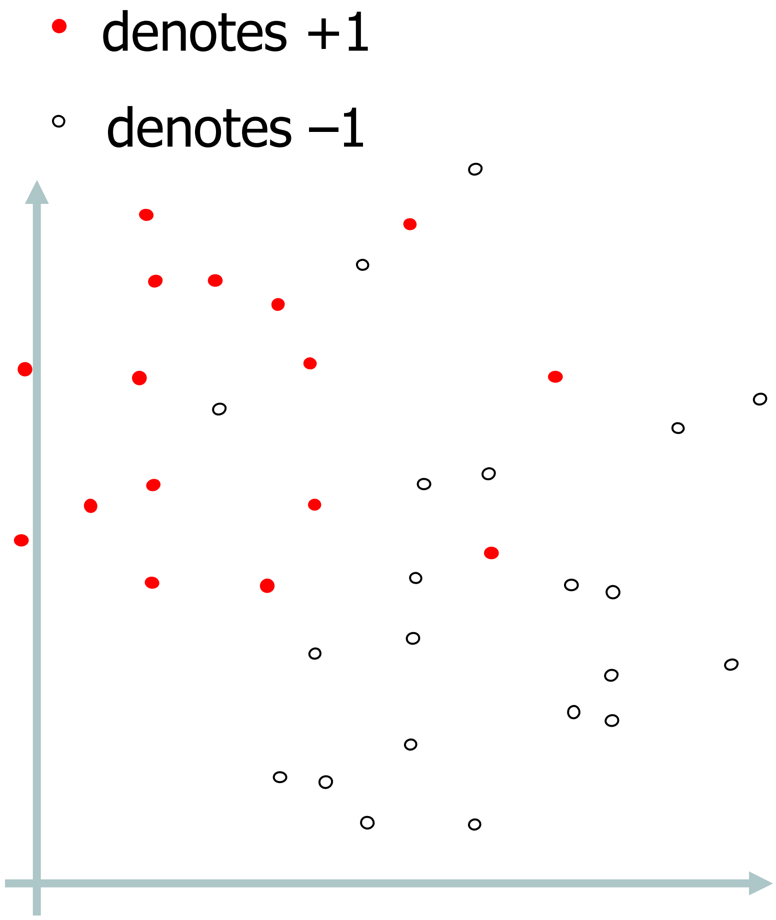
$\mathbf{w} \cdot \mathbf{x}_k + b \geq 1 \quad \text{if } y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq -1 \quad \text{if } y_k = -1$

Uh-oh!

This is going to be a problem!

What should we do?

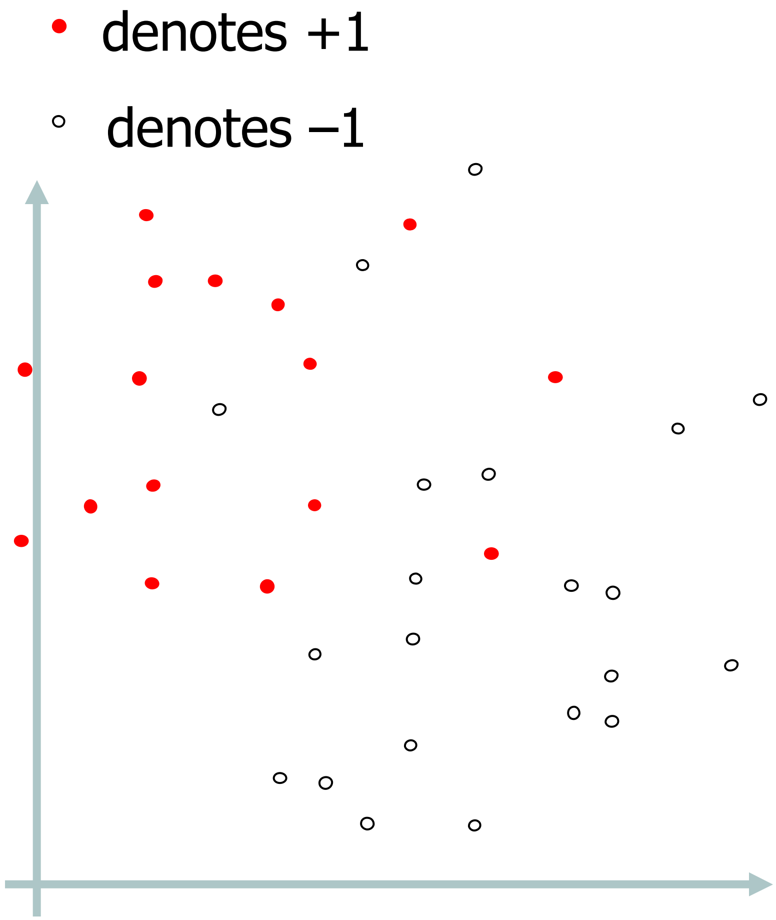


Idea 1:

Find minimum ***w·w***, while minimizing number of training set errors.

Problemette: Minimizing *TWO* things is ill-defined optimization

Uh-oh!



This is going to be a problem!

What should we do?

Idea 1.1:

Minimize

$$\mathbf{w}\cdot\mathbf{w} + C\,(\#\text{train errors})$$

Tradeoff parameter

But… a ***serious*** practical problem dooms this approach

Uh-oh!

This is going to be a problem!

What should we do?

Idea 1.1:

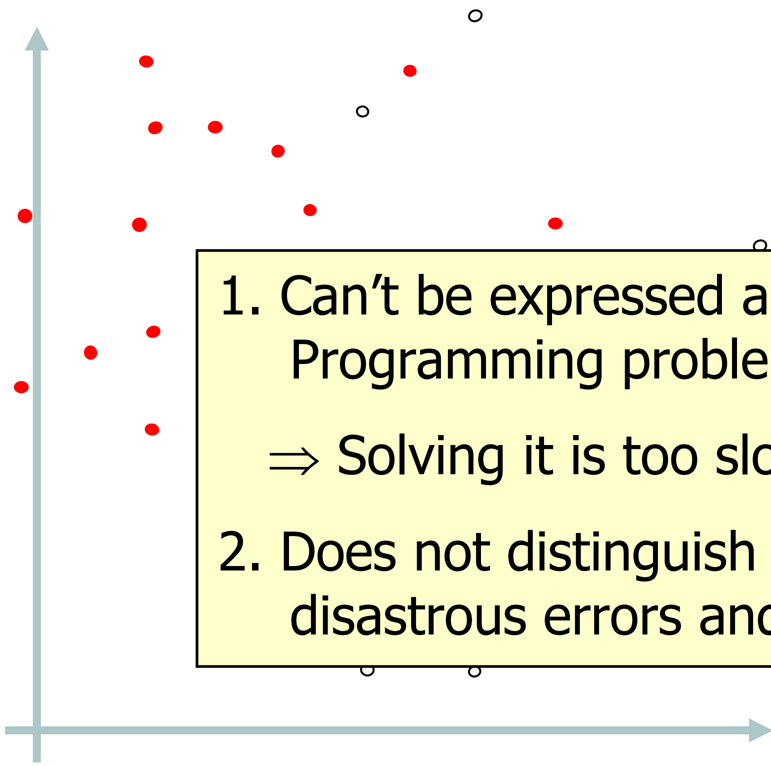
Minimize

$\mathbf{w} \cdot \mathbf{w} + C$ *(#train errors)*

Tradeoff parameter

• denotes +1

∘ denotes −1



1. Can’t be expressed as a Quadratic Programming problem.

   ⇒ Solving it is too slow.

2. Does not distinguish between disastrous errors and near misses

?? any other ideas ??

Uh-oh!

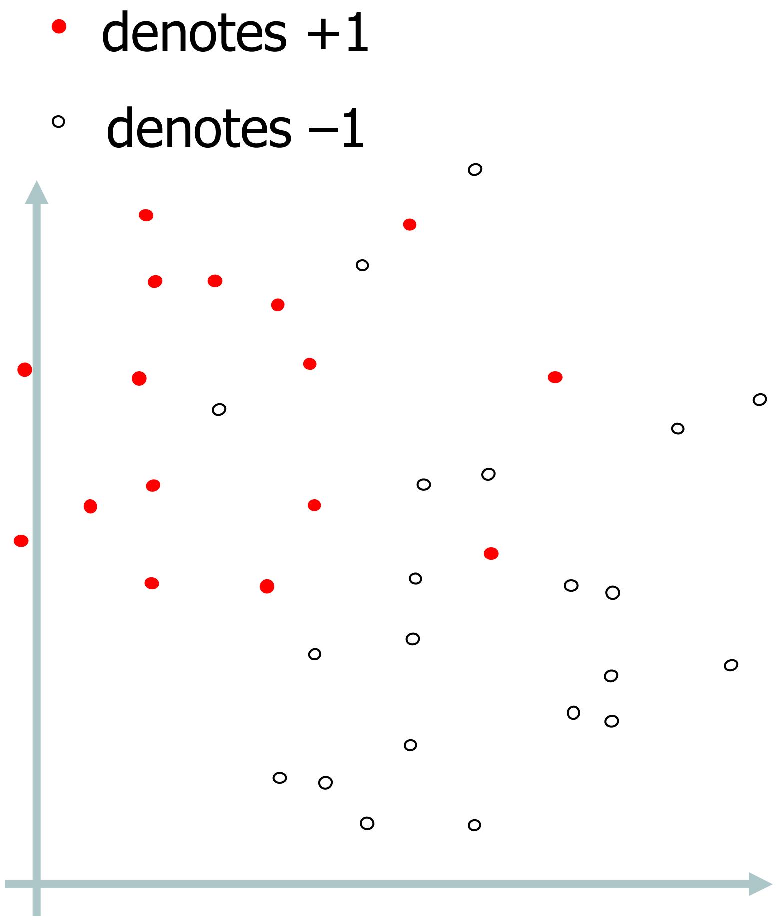
This is going to be a problem!

What should we do?

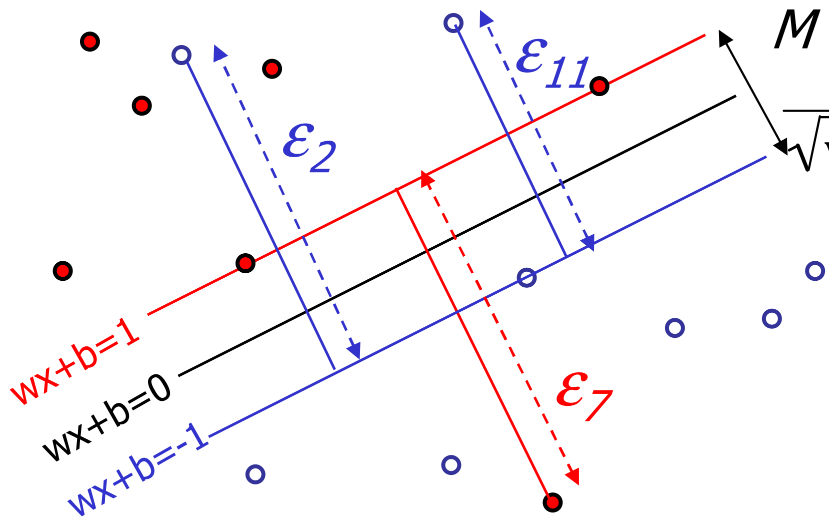
Idea 2.0:

Minimize

$\boldsymbol{w}\cdot\boldsymbol{w}$ + *C (distance from incorrectly labeled points to their correct place)*



Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$

Given guess of $\boldsymbol{w}$, b, can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

R datapoints, $\{[\mathbf{x}_k, y_k]\}$ where $y_k \in \{ +1, -1 \}$

What is quadratic optimization criterion?

Minimize $\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

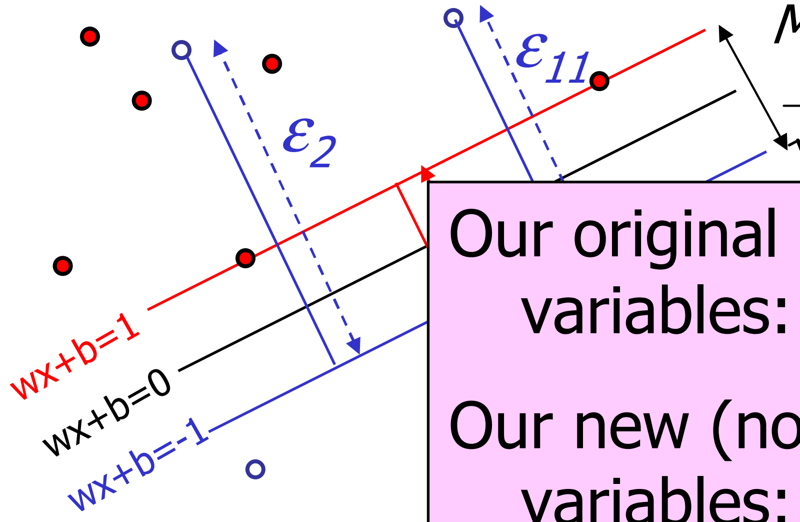
How many constraints? R

What should they be?

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \geq (1 - \varepsilon_k)$ if $y_k = 1$

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b \leq (-1 + \varepsilon_k)$ if $y_k = -1$

Learning Maximum Margin with Noise



$M = \dfrac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$

Given gu… can

- Compute whether all data …

…dth

…y_k)

where $y_k = \{+1, -1\}$

> m = # input dimensions

> Our original (noiseless data) QP had $m+1$ variables: $w_1, w_2, \ldots w_m$, and b.
>
> Our new (noisy data) QP has $m+1+R$ variables: $w_1, w_2, \ldots w_m, b, \varepsilon_k, \varepsilon_1, \ldots \varepsilon_R$

What is quadratic optimization criterion?

Minimize $\dfrac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constraints … R

> R= # records

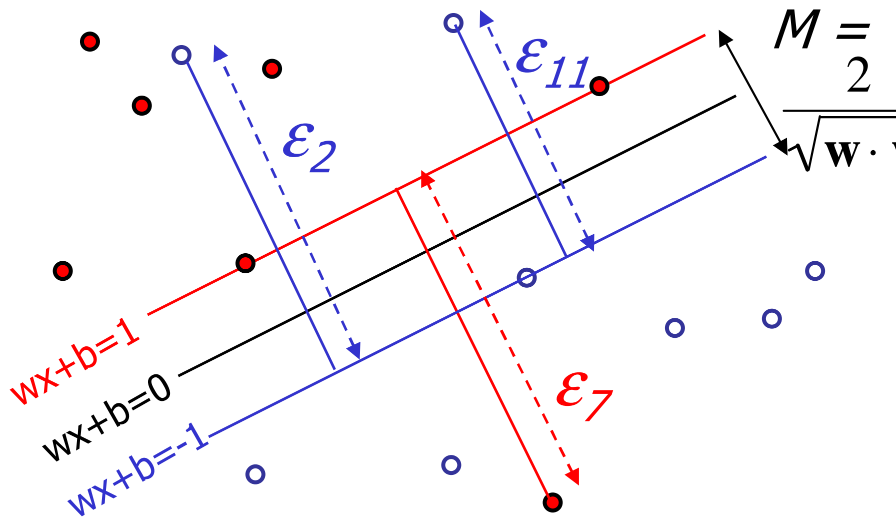
What should they be?

$\mathbf{w}\cdot\mathbf{x}_k + b \geq (1-\varepsilon_k) \quad \text{if } y_k = 1$

$\mathbf{w}\cdot\mathbf{x}_k + b \leq (-1+\varepsilon_k) \quad \text{if } y_k = -1$

There’s a bug in this QP. Can you spot it?

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}$

Given guess of ***w*** , *b*, can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

R datapoints, $\{[\mathbf{x}_k, y_k]\}$ where $y_k \in \{ +1, -1 \}$

What is quadratic optimization criterion?

Minimize $$\frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C\sum_{k=1}^{R} \varepsilon_k$$

Called "slack variables"

How many constraints? *2R*

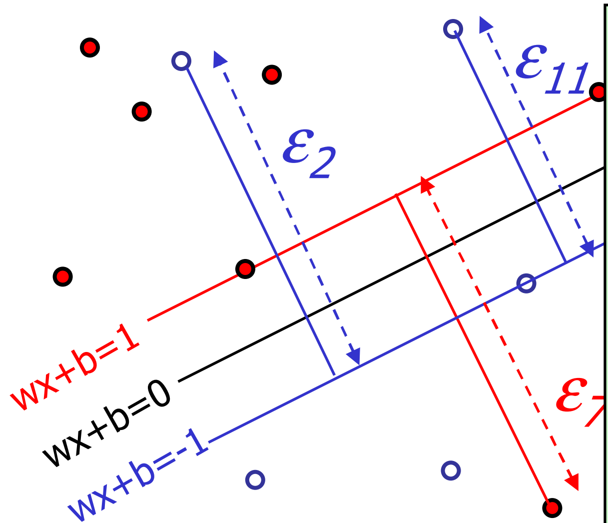
What should they be?

$\mathbf{w} \cdot \mathbf{x}_k + b \geq (1 - \varepsilon_k)$ if $y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq (-1 + \varepsilon_k)$ if $y_k = -1$

$\varepsilon_k \geq 0$ for all k

Learning Maximum Margin with Noise



Big ***C*** ⇒ "Fit the training data as much as possible!"
(at the expense of maximizing margin)

Small ***C*** ⇒ "Maximize the margin as much as possible!"
(at the expense of fitting the training data)

, *b*, can

her all data

e correct

argin width

$_k, y_k]\}$

+1, −1 }

What is quadratic optimization criterion

How many constraints? *2R*

What should they be?

Minimize $$\frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

$\mathbf{w} \cdot \mathbf{x}_k + b \geq (1 - \varepsilon_k)$ if $y_k = 1$

$\mathbf{w} \cdot \mathbf{x}_k + b \leq (-1 + \varepsilon_k)$ if $y_k = -1$

$\varepsilon_k \geq 0$ for all k

Called "slack variables"

Solving Constrained Optimization

$$\min_{\mathbf{w},b} L(\mathbf{w}, b, \lambda, \varepsilon) = \tfrac{1}{2}\,|\mathbf{w}|^2 + C \textstyle\sum_k \varepsilon_k - \sum_k \lambda_k [\, y_k(\mathbf{w} \cdot \mathbf{x}_k + b) - (1-\varepsilon_k)\,]$$

s.t. $\lambda, \varepsilon \geq 0$

To incorporate slack variables ε_k

Just add constraint:

$$0 \leq \lambda_k \leq C$$

Setting derivatives to 0…

- $\mathbf{w} = \sum_k \lambda_k y_k \mathbf{x}_k$
- $0 = \sum_k \lambda_k y_k$

Actually:

$$£(\lambda) = \ldots + \textstyle\sum_k (C - \lambda_k)\varepsilon_k$$

But $\varepsilon_k > 0 \Rightarrow \lambda_k = C$ …

Substitute back into L(..):

Find $\lambda \geq \mathbf{0}$ that minimizes

$$£(\lambda) = \textstyle\sum_k \lambda_k - \tfrac{1}{2} \sum_k \sum_m \lambda_k \lambda_m y_k y_m (\mathbf{x}_k \cdot \mathbf{x}_m)$$

An Equivalent QP

Maximize $\displaystyle \max_{\lambda_k} \sum_{k=1}^{R} \lambda_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \lambda_k \lambda_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \leq \lambda_k \leq C \quad \forall k$ $\qquad \displaystyle \sum_{k=1}^{R} \lambda_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \lambda_k y_k \mathbf{x}_k$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K \cdot \mathbf{w}_K$$

$$\text{where} \quad K = \arg\max_k \alpha_k$$

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w* · *x* + *b*)**

An Equivalent QP

Maximize $\sum_{k=1}^{R} \lambda_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R} \lambda_k \lambda_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$
λ_k

Subject to these constraints: $0 \le \lambda_k \le C \quad \forall$ … $\sum_{k=1}^{R}$ …

$\mathbf{x}_k$ only appears in dot product!

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \lambda_k y_k \mathbf{x}_k$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K \cdot \mathbf{w}$$

where $K = \arg\max_k \alpha$ …

Datapoints with $\lambda_k > 0$ == support vectors

Then classify with:

$f(x, w, b)$ … $sign($ … $\mathbf{x} + b)$

...note this sum only needs to be over the support vectors.

(probably << R)

An Equivalent QP

Maximize over λ_k: $\displaystyle \sum_{k=1}^{R} \lambda_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R} \lambda_k \lambda_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constr...

...dot product!

Why use this equivalent QP?

- QP packages can optimize it more quickly
- Stay tuned…

Then de...

$\mathbf{w} = \sum_{k=1} \lambda ...$

$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K \cdot \mathbf{w}$

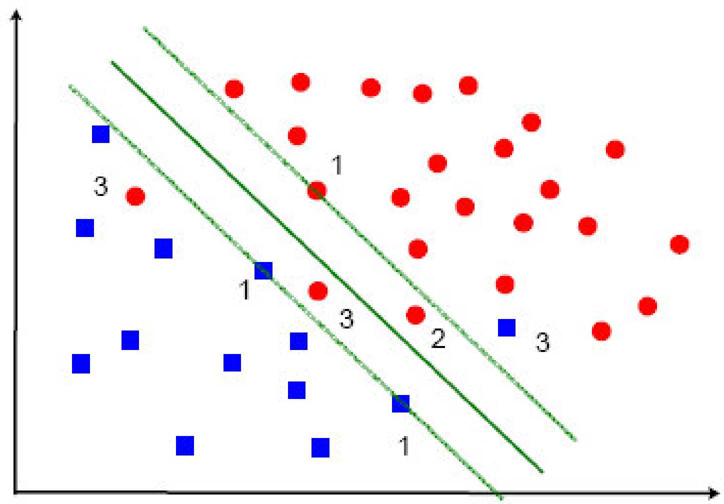
where $K = \arg\max_k \alpha ...$

...(w · x + b)

...notice this sum only needs to be over the support vectors.

(probably << R)

Types of Support Vectors



Support Vectors:

1	margin s.v.	$\xi_i = 0$	Correct
2	non-margin s.v.	$\xi_i < 1$	Correct (in margin)
3	non-margin s.v.	$\xi_i > 1$	Error

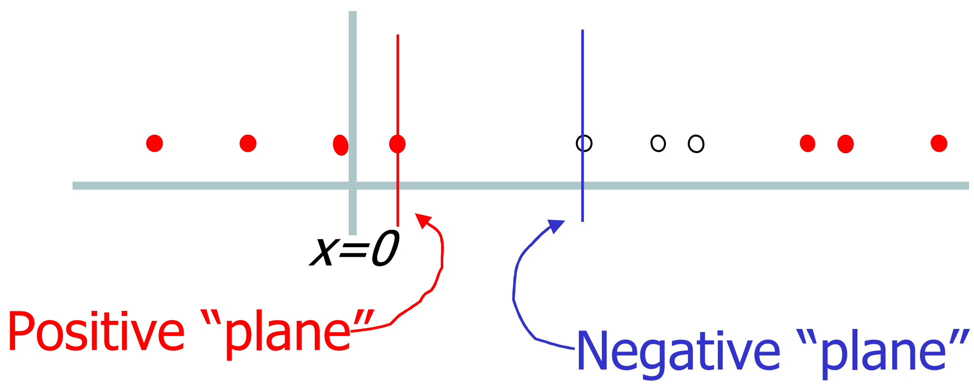
What do we have?

- Method for learning a
  maximum-margin linear classifier
  when the data are …
  - "Linearly separable" –
    ∃ line that gets 0 training error
  - Not linearly separable – i.e. no such line.
- If not linearly separable, must trade-off between maximizing margin and minimizing "stuff-is-on-the-wrong-side-ness"
  - … OR DO WE?? Kernels!

Hard 1-dimensional Dataset

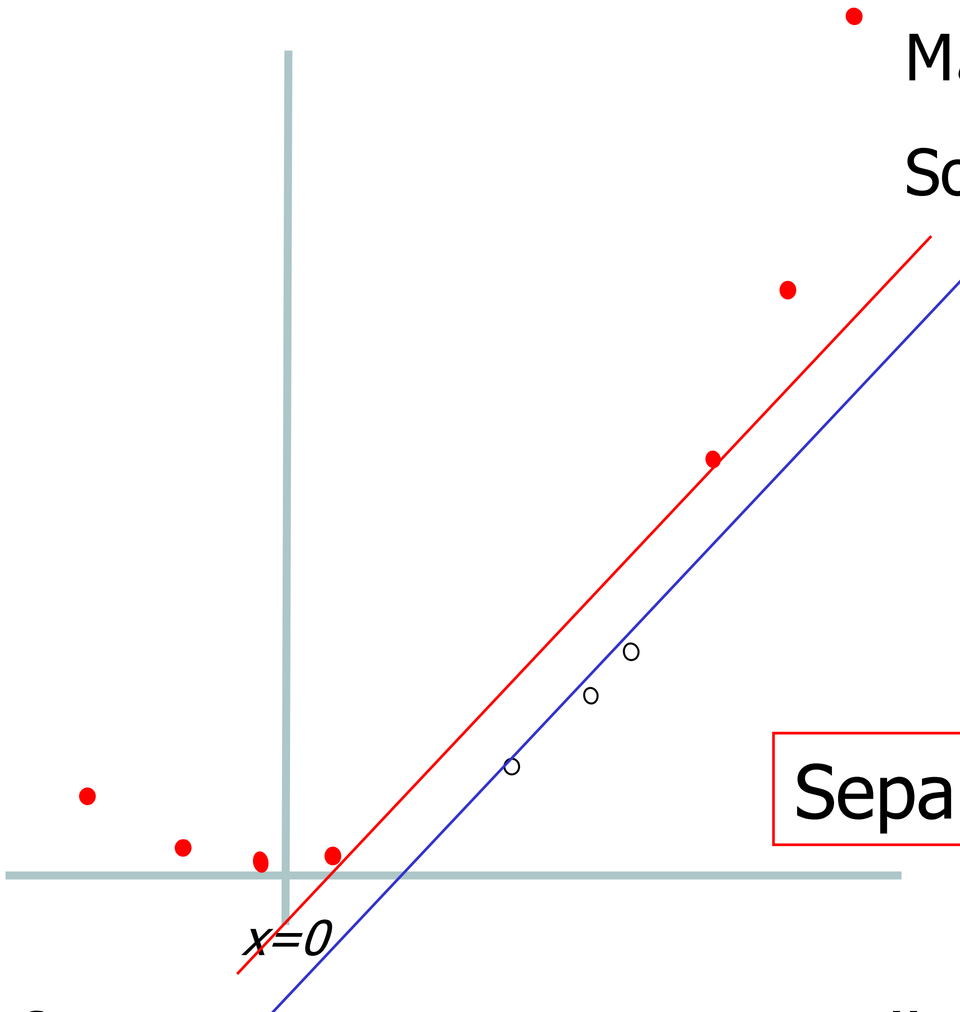
What would SVMs do with this data?

Not a big surprise



Doesn't look like slack variables will save us this time…

Hard 1-dimensional Dataset



M*ake up a new feature!*

Sort of…

… computed from original feature(s)

$$\mathbf{z}_k = (x_k, x_k^2)$$

Separable! MAGIC!

New features are sometimes called *basis functions.*

Now drop this "augmented" data into our linear SVM.

… New Features from Old …

- Here: mapped $\Re \to \Re^2$ by Φ: $x \to [x, x^2]$
  - Found "extra dimensions" $\Rightarrow$ linearly separable!
- In general,
  - Start with vector $\mathbf{x} \in \Re^k$
  - Want to add in x_1^2 , x_2^2, …
  - Probably want other terms – eg $x_2 \cdot x_7$, …
  - Which ones to include?

    Why not ALL OF THEM?

    (If $\Re^r$ linearly-separable, then any SUPERSET is)
- $(x_1, x_2, x_3) \to$
  $(1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3)$
  - $\Re^3 \to \Re^{10}$
  - In general,

$$m \to 1 + m + m + \binom{m}{2} = \frac{(m+2)(m+1)}{2} \approx \frac{m^2}{2}$$

Implied Algorithm

- Training: Given R training instances, each in $\Re^m$
  1. Map each $\Re^m$ –tuple $\mathbf{x}_i$ to $\Re^{m*m/2}$ –tuple $\Phi(\mathbf{x}_i)$
  2. Learn SVM classifier wrt these $\Phi(\mathbf{x}_i)$ tuples

- Performance: Given new $\Re^m$ –tuple $\mathbf{x}$
  1. Map this $\mathbf{x}$ to $\Re^{m*m/2}$ –tuple $\Phi(\mathbf{x})$
  2. Apply learned SVM classifier to $\Phi(\mathbf{x})$

One more trick!

- Issue:
  - This $\Phi(.)$ operation is expensive!! – $O(m^2)$
  - What if want $\Phi'(.)$ that deals with "x^3", or "x^4", or …

See lectures by B Poczos!

Quadratic Basis Functions

Skip from here to "VC-dimension of an SVM"

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ : \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ : \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ : \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ : \\ \sqrt{2}x_1x_m \\ : \\ \sqrt{2}x_{m-1}x_m \end{pmatrix}$$

- Constant Term: 1
- Linear Terms: $\sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_m$
- Pure Quadratic Terms: $x_1^2, x_2^2, \ldots, x_m^2$
- Quadratic Cross-Terms: $\sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{m-1}x_m$

What about those $\sqrt{2}$??

… stay tuned

Quadratic Dot Products

$$\Phi(\mathbf{a})\bullet\Phi(\mathbf{b}) = \begin{pmatrix} 1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ \vdots \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \vdots \\ \sqrt{2}a_{m-1}a_m \end{pmatrix} \bullet \begin{pmatrix} 1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ \vdots \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \vdots \\ \sqrt{2}b_{m-1}b_m \end{pmatrix}$$

$$1 + \sum_{i=1}^{m} 2a_ib_i + \sum_{i=1}^{m} a_i^2b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$

$$1 + 2\sum_{i=1}^{m} a_i b_i + \sum_{i=1}^{m} (a_i b_i)^2 + \sum_{i=1}^{m} \sum_{j=i+1}^{m} 2 a_i a_j b_i b_j$$

Now consider another fn of ***a*** and ***b*** :

$$(\mathbf{a} \cdot \mathbf{b} + 1)^2$$

$$= (\mathbf{a} \cdot \mathbf{b})^2 + 2\mathbf{a} \cdot \mathbf{b} + 1$$

$$= \left(\sum_{i=1}^{m} a_i b_i \right)^2 + 2\sum_{i=1}^{m} a_i b_i + 1$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

They're the same!

And this is only O(m) to compute… not O(m^2)

Higher Order Polynomials

$$Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$$

Poly-nomial	$\phi(\boldsymbol{x})$	Cost to build Q_{kl} matrix: *traditional*	Cost if 100 inputs	$\phi(\boldsymbol{a})\cdot\phi(\boldsymbol{b})$	Cost to build Q_{kl} matrix: *sneaky*	Cost if 100 inputs
Quadratic	All $m^2/2$ terms up to degree 2	$m^2 R^2 /4$	$2\,500\ R^2$	$(\mathbf{a}\cdot\mathbf{b}+1)^2$	$m R^2 / 2$	$50\ R^2$
Cubic	All $m^3/6$ terms up to degree 3	$m^3 R^2 /12$	$83\,000\ R^2$	$(\mathbf{a}\cdot\mathbf{b}+1)^3$	$m R^2 / 2$	$50\ R^2$
Quartic	All $m^4/24$ terms up to degree 4	$m^4 R^2 /48$	$1\,960\,000\ R^2$	$(\mathbf{a}\cdot\mathbf{b}+1)^4$	$m R^2 / 2$	$50\ R^2$

Original QP

Maximize $\underset{\alpha_k}{\text{}}$ $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\quad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K \cdot \mathbf{w}_K$$

$$\text{where} \quad K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w*** · ***x*** + *b*)

QP using Basis Functions

Maximize $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l(\Phi(\mathbf{x}_k)\cdot\Phi(\mathbf{x}_l))$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\quad \sum_{k=1}^{R}\alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k)$$

$$b = y_K(1-\varepsilon_K) - \Phi(\mathbf{x}_K)\mathbf{w}$$

$$\text{where} \quad K = \arg\max_k \alpha_k$$

Then classify with:

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

QP using Basis Functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_l))$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

$\phi(\mathbf{x}_k)$ only appears within dot product!

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k)$$

$$b = y_K (1 - \varepsilon_K) - \Phi(\mathbf{x}_K) \mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

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

$= sgn(\sum_k \alpha_k y_k \phi(\mathbf{x}_k) \cdot \phi(\mathbf{x}) + b)$

$= sgn(\sum_k \alpha_k y_k [\phi(\mathbf{x}_k) \cdot \phi(\mathbf{x})] + b)$

QP with Quintic basis functions

This matrix requires $R^2/2$ dot products.

In 100-d, each dot product requires 103 ops, … not 75 million

But still worries…

$$Q_{kl} = y_k y_l (\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_l))$$

The use of Maximum Margin magically reduces this problem

Subject to these constraints: $0 \le \alpha_k \ldots$

- Overfitting due to enormous number of terms
- The evaluation phase (doing a predictions on a test instance **x**) seems expensive…
  as **w**·φ(**x**) needs 75 million operations

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k)$$

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x})$$

$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only $S\,m$ operations (S=#support vectors)

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

$$= sgn(\textstyle\sum_k \alpha_k y_k \phi(\mathbf{x}_k) \cdot \phi(\mathbf{x}) + b)$$

$$= sgn(\textstyle\sum_k \alpha_k y_k [\phi(\mathbf{x}_k) \cdot \phi(\mathbf{x})] + b)$$

The "Kernel Trick"!

maximize$_\alpha$ $\quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

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

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

- Never represent features explicitly
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features

… at classification time

- For a new input **x**, if we need to represent Φ(**x**), we are in trouble!
- Recall classifier: sign(**w**.Φ(**x**)+b)
- Using kernels we are cool!

$$K(\mathbf{u},\mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

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

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

Classifying using SVMs with Kernels

- Choose a set of features and kernel function
- Solve dual problem to obtain support vectors α_i
- At classification time, compute:

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

for any k where $C > \alpha_k > 0$

Classify as

$$sgn(\textstyle\sum_k \alpha_k y_k [\phi(\mathbf{x}_k) \cdot \phi(\mathbf{x})] + b)$$

$$= sgn(\textstyle\sum_k \alpha_k y_k \mathrm{K}(\mathbf{x}_k, \mathbf{x}) + b)$$

What makes a valid kernel?

- In general, K matrix must be symmetric, positive semidefinite
- A *sufficient* (but not necessary) condition is for K to behave like a distance metric
  - Nonnegative
  - K(x,x)=0
  - Symmetric
  - Obeys triangle inequality
- Fancy kernels can be constructed by combining simple ones

Common Kernels

- Polynomials of degree d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

- Gaussian kernels

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||}{2\sigma^2}\right)$$

Equivalent to $\phi(x)$ of infinite dimensionality!

Source of Kernels?

- Can generate new kernels from old:
- If $k_1(\mathbf{x},\mathbf{x}')$, $k_2(\mathbf{x},\mathbf{x}')$ are kernels, then so is:
  - $\mathbf{x}^T A \mathbf{x}'$ – A any positive semidefinite matrix
  - $c\ k_1(\mathbf{x},\mathbf{x}')$ – $c \in \Re^+$
  - $f(\mathbf{x})\ k_1(\mathbf{x},\mathbf{x}')\ f(\mathbf{x}')$ – $f(.)$ any function
  - $q(\ k_1(\mathbf{x},\mathbf{x}')\ )$ – $q(.)$ any poly function w/ coeff's ≥ 0
  - $\exp(\ k_1(\mathbf{x},\mathbf{x}')\ )$
  - $k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}')$
  - $k_a(\mathbf{x}_a,\mathbf{x}'_a) + k_b(\mathbf{x}_b,\mathbf{x}'_b)$ –
    $\mathbf{x} = (\mathbf{x}_a\ \mathbf{x}_b)$ and $k_a(.,.)$ kernel over "a" space,
    $k_b(.,.)$ kernel over "b" space
  - …

Overfitting?

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting

VC-dimension of an SVM

- Very very very loosely speaking... under some assumptions, an upper bound on the VC dimension is:

$$\left\lceil \frac{\text{Diameter}}{\text{Margin}} \right\rceil$$

- where
  - *Diameter* = diameter of the smallest sphere that can enclose all the high-dimensional term-vectors derived from the training set.
  - *Margin* = smallest margin we'll let the SVM use
- Used in SRM (Structural Risk Minimization) for choosing the polynomial degree, RBF σ, etc.
  - But most people just use Cross-Validation...

SVM Performance

- Anecdotally SVMs do work very very well indeed.
  - *Eg1:* The best-known classifier on a well-studied hand-written-character recognition benchmark
  - *Eg2:* Many people doing practical real-world work claim that SVMs have saved them…
    when their other favorite classifiers did poorly.

- Lots of excitement and religious fervor about SVMs as of 2001…

- Still… some practitioners are a little skeptical…

Doing Multi-Class Classification

- SVMs can only handle two-class outputs
  (i.e. a categorical output variable with arity 2)
- What can be done?
- Answer: with output arity N, learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - :
  - SVM N learns "Output==N" vs "Output != N"
- Then, to predict the output for a new input:
  - just predict with each SVM and
  - select the class w/ largest margin
    [whose prediction is **furthest** into the positive region]

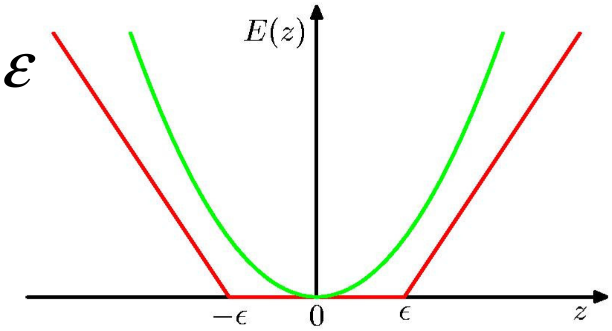
SVM Regression

- Typical loss function: $$C\sum_n (y_n - t_n)^2 + \frac{1}{2}\|w\|^2$$

… penalty whenever $y_n \neq t_n$

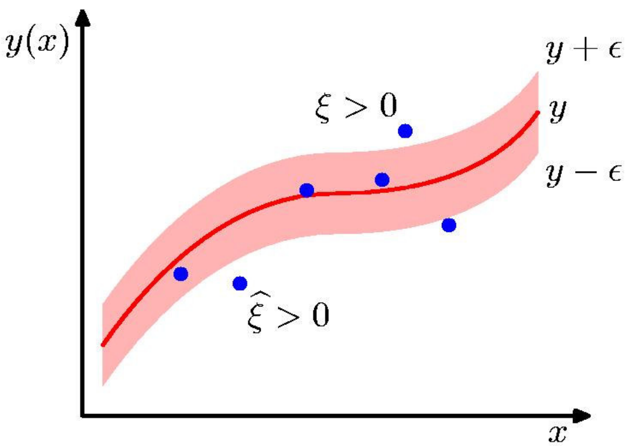
- To be sparse… don't worry if "close enough"

  - $E_\varepsilon(y, t) = \begin{cases} 0 & \textit{if } |y - t| < \varepsilon \\ |y - t| - \varepsilon & \textit{otherwise} \end{cases}$



- … loss function

$$C\sum_n E_\varepsilon(y_n, t_n) + \frac{1}{2}\|w\|^2$$

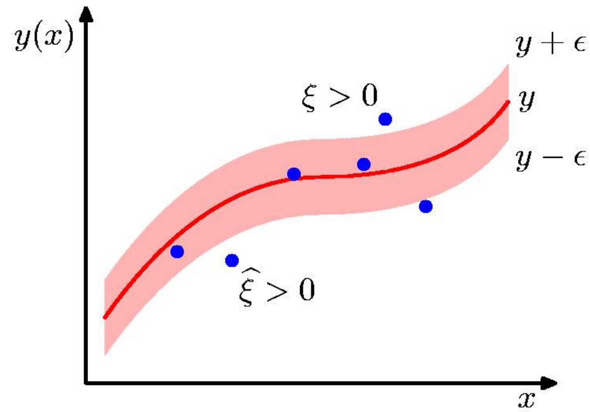


- No penalty if in ε-tube

SVM Regression

$$C\sum_n E_\varepsilon(y_n, t_n) + \frac{1}{2}\|w\|^2$$

$$E_\varepsilon(y, t) = \begin{cases} 0 & \text{if } |y - t| < \varepsilon \\ |y - t| - \varepsilon & \text{otherwise} \end{cases}$$



- No penalty if $y_n - \varepsilon \le t_n \le y_n + \varepsilon$
- Slack variables: $\{\xi_{n+}, \xi_{n-}\}$
  - $t_n \le y_n + \varepsilon + \xi_{n+}$
  - $t_n \ge y_n - \varepsilon - \xi_{n-}$
- Error function:

$$C\sum_n (\xi_{n+} + \xi_{n-}) + \frac{1}{2}\|w\|^2$$

- … use Lagrange Multipliers $\{ a_{n+}, a_{n-}, \mu_{n+}, \mu_{n-} \} \ge 0$

$$\min L(\ldots) = C\sum_n (\xi_{n+} + \xi_{n-}) + \frac{1}{2}\|w\|^2 - \sum_n (\mu_{n+}\xi_{n+} + \mu_{n-}\xi_{n-})$$
$$- \sum_n a_{n+}(\varepsilon + \xi_{n+} + y_n - t_n) - \sum_n a_{n-}(\varepsilon + \xi_{n-} - y_n + t_n)$$

SVM Regression, con't

$$L(\ldots) = C\sum_n(\xi_{n+} + \xi_{n-}) + \frac{1}{2}\|w\|^2 - \sum_n(\mu_{n+}\xi_{n+} + \mu_{n-}\xi_{n-})$$
$$- \sum_n a_{n+}(\varepsilon + \xi_{n+} + y_n - t_n) - \sum_n a_{n-}(\varepsilon + \xi_{n-} - y_n + t_n)$$

- Set derivatives to 0, solve for $\{\xi_{n+}, \xi_{n-}, \mu_{n+}, \mu_{n-}\}$ …

$$\min_{\vec{a}_+,\vec{a}_-} \tilde{L}(\vec{a}_+,\vec{a}_-) = -\frac{1}{2}\sum_n\sum_m(a_{n+} - a_{n-})(a_{m+} - a_{m-})k(x_n, x_m)$$
$$- \varepsilon\sum_n(a_{n+} - a_{n-}) + \sum_n(a_{n+} - a_{n-})t_n$$

s.t. $0 \le a_{n+} \le C$ $\quad 0 \le a_{n-} \le C$

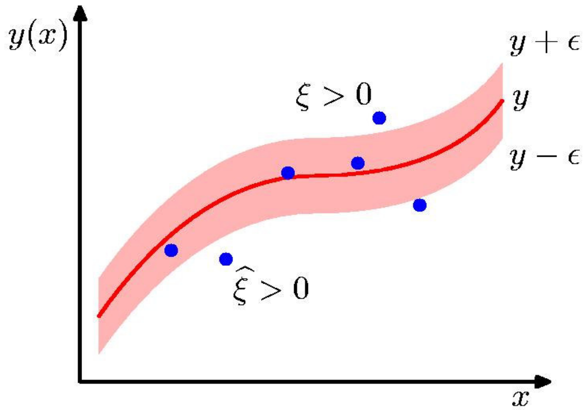
Quadratic Program!!

- Prediction for new **x** :

$$y(x) = \sum_n(a_{n+} - a_{n-})\, k(x_n, x)$$

SVM Regression, con't

$$y(x) = \sum_n (a_{n+} - a_{n-})\, k(x_n, x)$$



- Can ignore $\mathbf{x}_n$ unless either $a_{n+}>0$ or $a_{n-}>0$
  - $a_{n+}>0$ only if $t_n = y_n + \varepsilon + \xi_{n+}$
    ie, if on upper boundary of ε-tube ($\xi_{n+}=0$)
    or above ($\xi_{n+}>0$)
  - $a_{n-}>0$ only if $t_n = y_n - \varepsilon - \xi_{n-}$
    ie, if on lower boundary of ε-tube ($\xi_{n-}=0$)
    or below ($\xi_{n-}>0$)

Kernels in Logistic Regression

See Poczos lecture

$$P(Y=1 \mid x, \mathbf{w}) \;=\; \frac{1}{1+e^{-(\mathbf{w}\cdot\Phi(\mathbf{x})+b)}}$$

- Define weights in terms of support vectors:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$$

$$P(Y=1 \mid x, \mathbf{w}) \;=\; \frac{1}{1+e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x})+b)}}$$

$$= \frac{1}{1+e^{-(\sum_i \alpha_i K(\mathbf{x},\mathbf{x}_i)+b)}}$$

- Derive simple gradient descent rule on α_i

Difference between SVMs and Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	**Yes!**	No
Solution sparse	Often yes!	Almost always no!
Semantics of output	"Margin"	Real probabilities

SVM Implementations

- Sequential Minimal Optimization, SMO [Platt]
  - efficient implementation of SVMs
  - in Weka

- SVMlight
  - http://svmlight.joachims.org/

- Run time:
  - typically quadratic in the number of data points
  - perhaps less if # support vectors is small

References

- An excellent tutorial on VC-dimension and Support Vector Machines:

  C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html

- The VC/SRM/SVM Bible:

  Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998.

  BUT YOU SHOULD PROBABLY READ ALMOST ANYTHING ELSE ABOUT SVMS FIRST.

Key SVM Ideas

- Maximize the margin between + and – examples
  - connects to PAC theory
- Sparse:
  Only the support vectors contribute to solution
- Penalize errors in non-separable case
- Kernels map examples into a new, usually nonlinear space
  - Implicitly do dot products in this new space
    (in the "dual" form of the SVM program)

  - Kernels are separate from SVMs
    … but they combine very nicely with SVMs

Summary I

Advantages

- Systematic implementation through quadratic programming
  - ∃ *very efficient* implementations
- Excellent data-dependent *generalization bounds*
- *Regularization* built into cost function
- Statistical performance is *independent* of dim. of feature space

- Theoretically related to widely studied fields of regularization theory and sparse approximation
- Fully adaptive procedures available for determining hyper-parameters

Summary II

Drawbacks

- Treatment of non-separable case somewhat heuristic
- Number of support vectors may depend strongly on the kernel type and the hyper-parameters
- Systematic choice of kernels is difficult (prior information)
  - … some ideas exist
- Optimization may require clever heuristics for large problems

Summary III

Extensions

- Online algorithms
- Systematic choice of kernels using generative statistical models
- Applications to
  - Clustering
  - Non-linear principal component analysis
  - Independent component analysis
- Generalization bounds constantly improving
  - (some even practically useful!)

What You Should Know

- Definition of a maximum margin classifier
- Sparse version: (Linear) SVMs
- What QP can do for you
  (even if you don't know how it works)
- How Maximum Margin = a QP problem
- How to deal with noisy (non-separable) data
  - Slack variable
- How to permit "non-linear boundaries"
  - Kernel trick
- How SVM Kernel functions permit us to pretend we're working with a zillion features

What really happens

- Johnny Machine Learning gets a dataset

- Wants to try SVMs
  - Linear: “Not bad, but I think it could be better.”
  - Adjusts C to trade off margin vs. slack
  - Still not satisfied: Tries kernels, typically polynomial. Starts with quadratic, then goes up to about degree 5.

- Johnny goes to Machine Learning conference
  - Johnny: “Wow, a quartic kernel with C=2.375 works great!”
  - Audience member: “Why did you pick those, Johnny?”
  - Johnny: “Cross validation told me to!”

Understanding LOO

- LOO estimates probability that
  a classifier trained on n-1 points
  gets the n^{th} point right
- For largish n, LOO is ≈ an average of n such draws
- For SVM with k support vectors, n training points
  - At least n-k draws will produce the same classifier
  - At least this many will get the next point, right
- Suggests empirical error of our SVM should be as low as k/n …

Relevance Vector Machine

- Bayesian Version of SVM
- Provides probabilities on outputs
- Tends to produce sparser solutions
- Requires non-linear optimization
- Can be slow