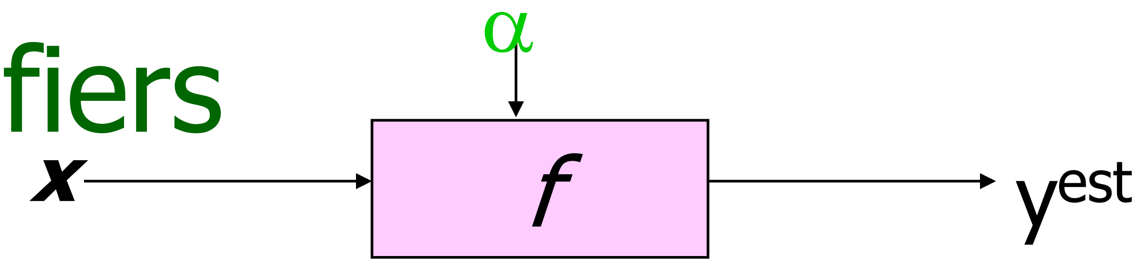
Support Vector Machines

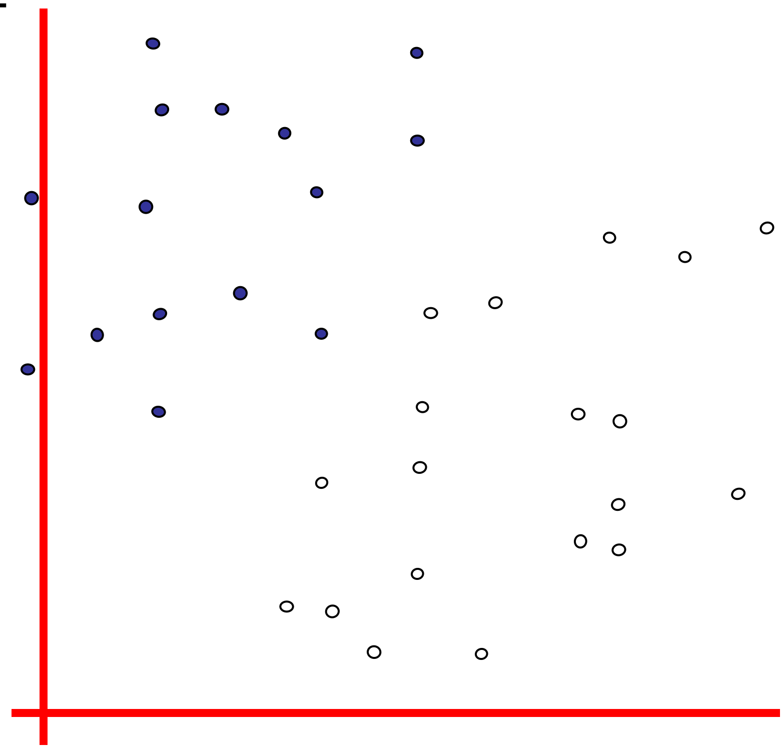
Andrew W. Moore
Professor
School of Computer Science
Carnegie Mellon University
www.cs.cmu.edu/~awm
awm@cs.cmu.edu
412-268-7599

Linear Classifiers



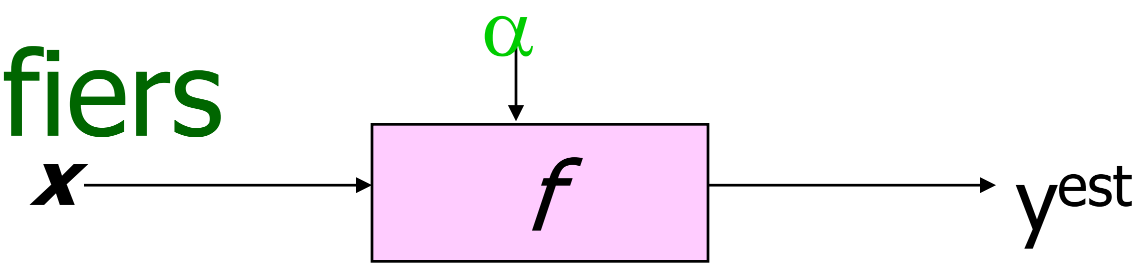
$$f(\mathbf{x},\mathbf{w},b) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} - b)$$

- ● denotes +1
- ○ denotes -1

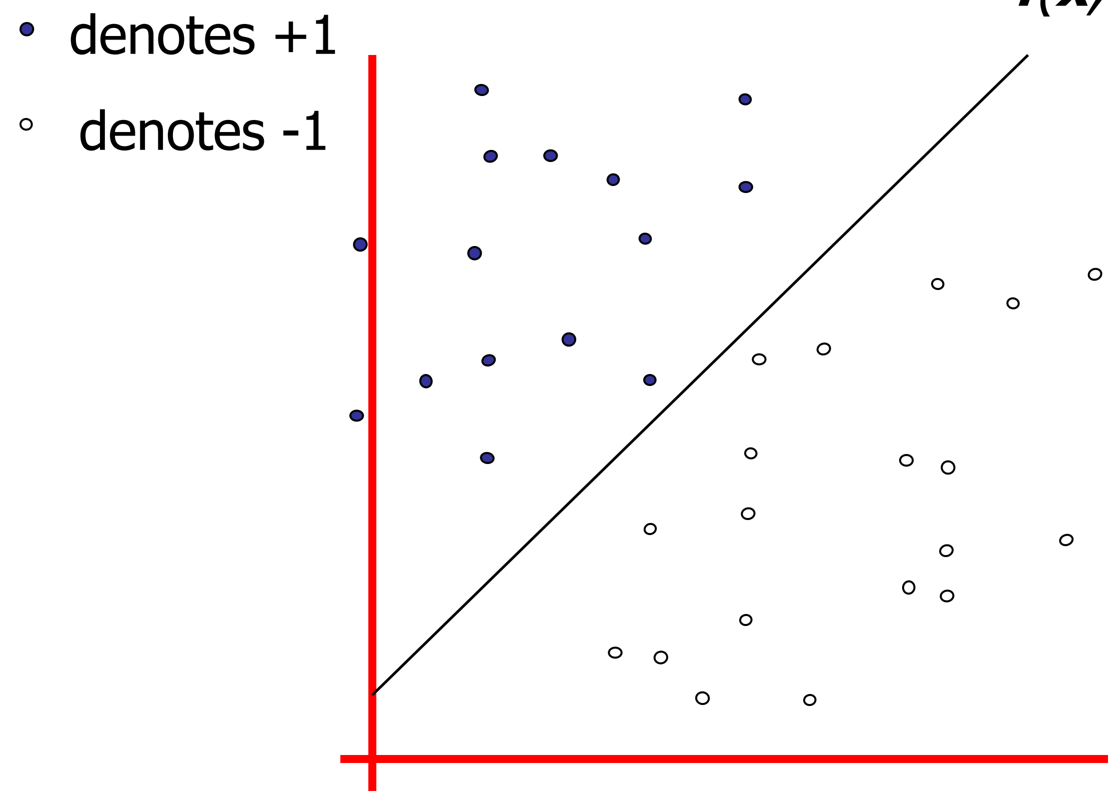


How would you classify this data?

Linear Classifiers

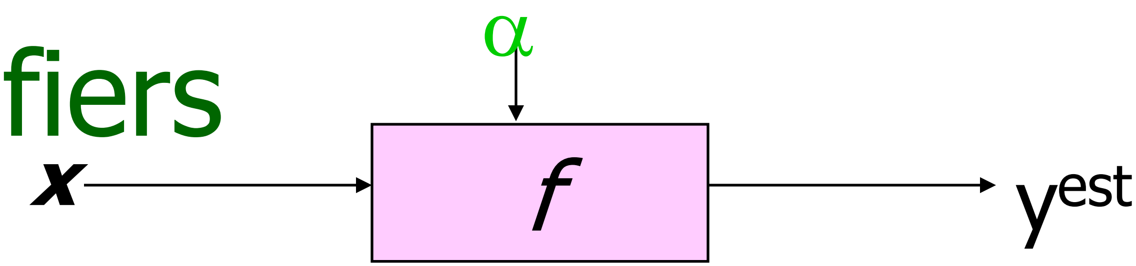


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$



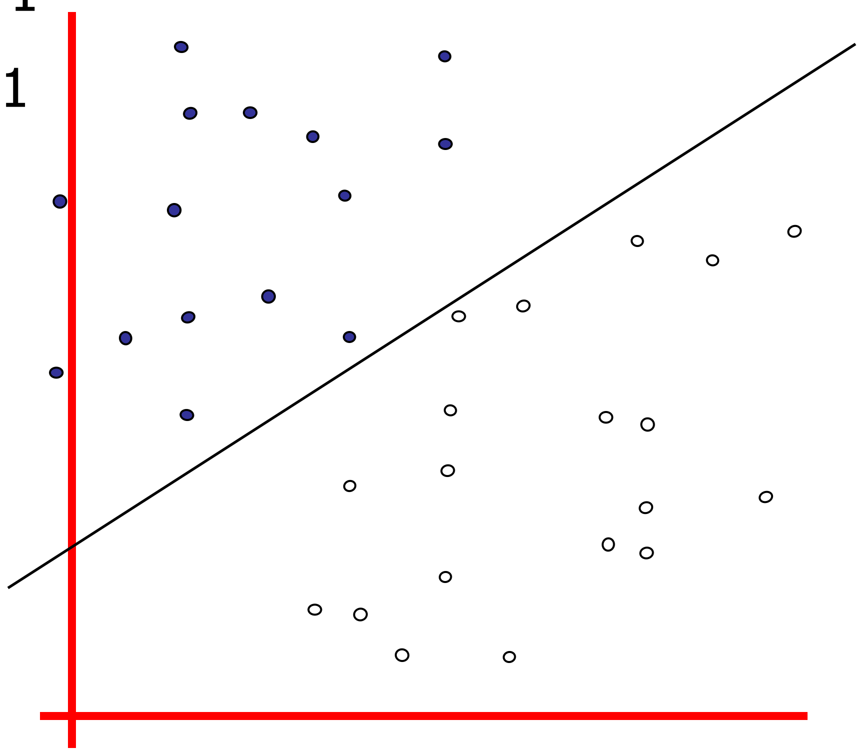
How would you classify this data?

Linear Classifiers



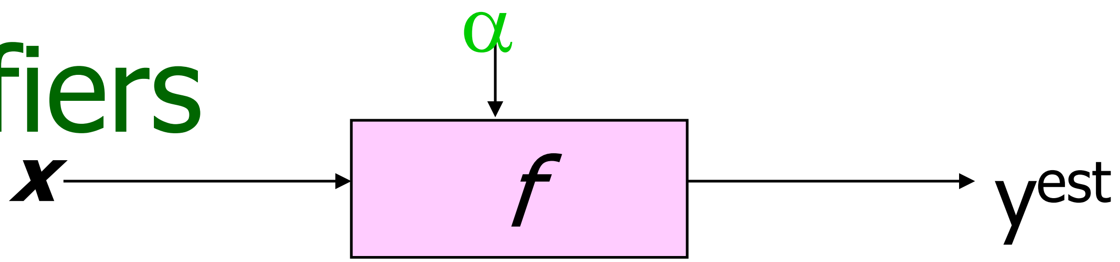
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\,\mathbf{x} - b)$$

- ● denotes +1
- ○ denotes -1



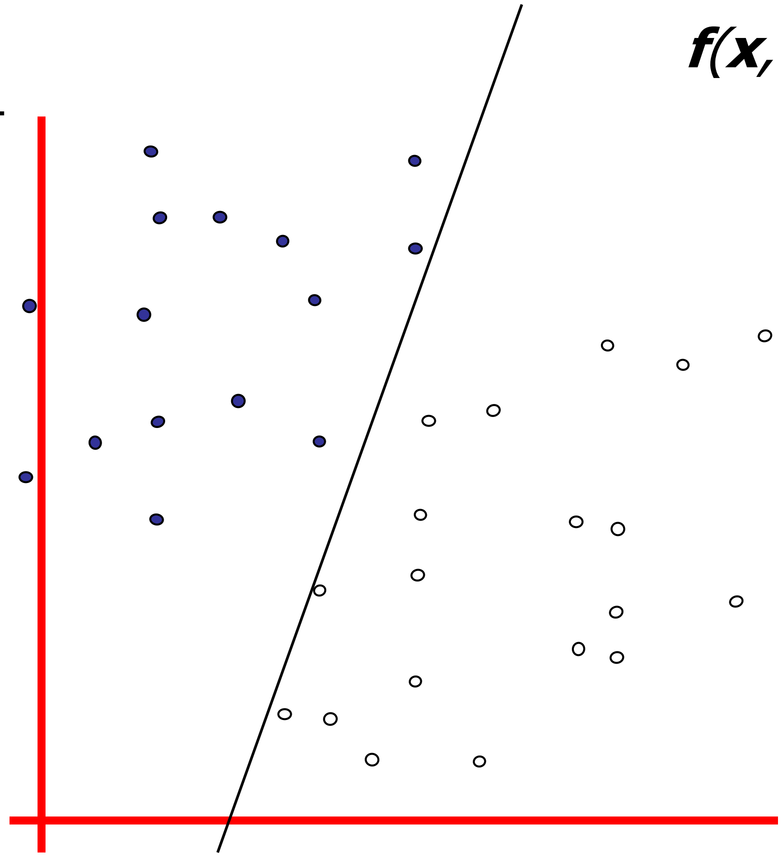
How would you classify this data?

Linear Classifiers



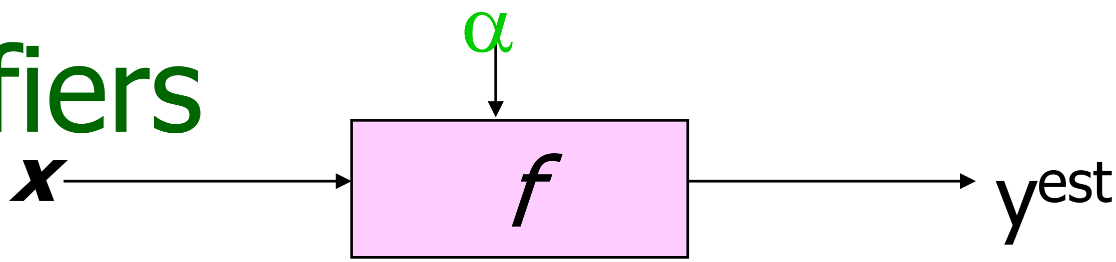
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$

- denotes +1
- ◦ denotes -1



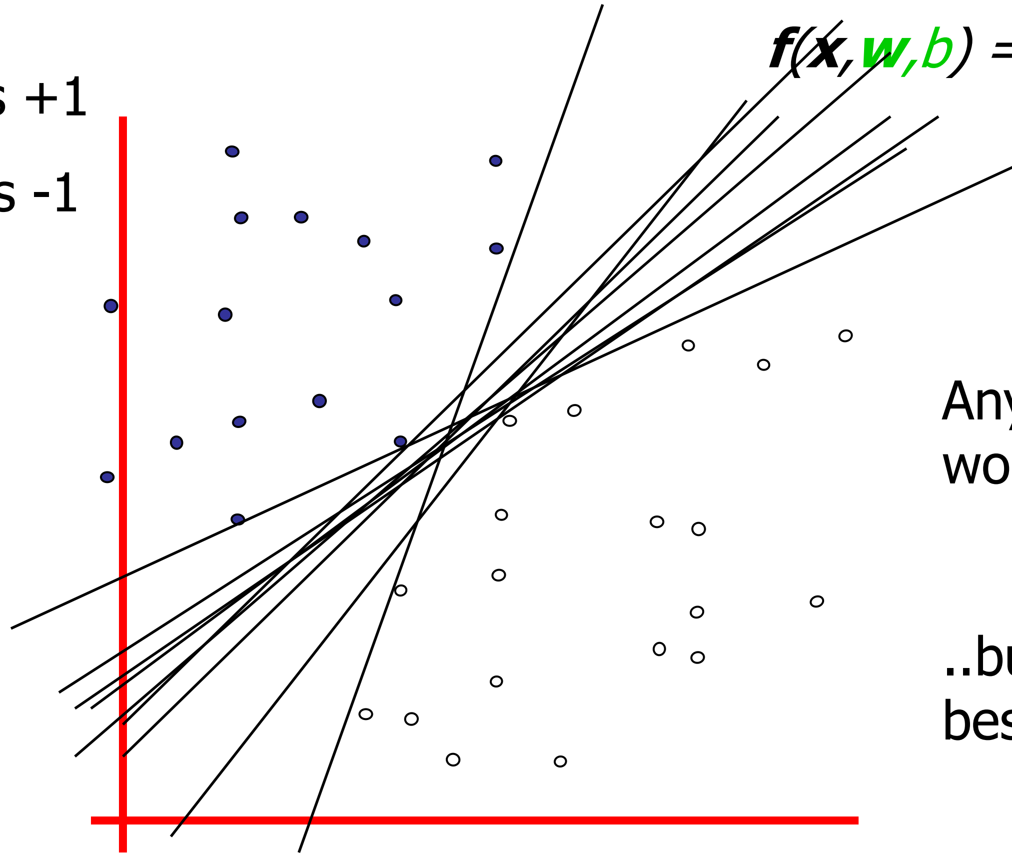
How would you classify this data?

Linear Classifiers



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

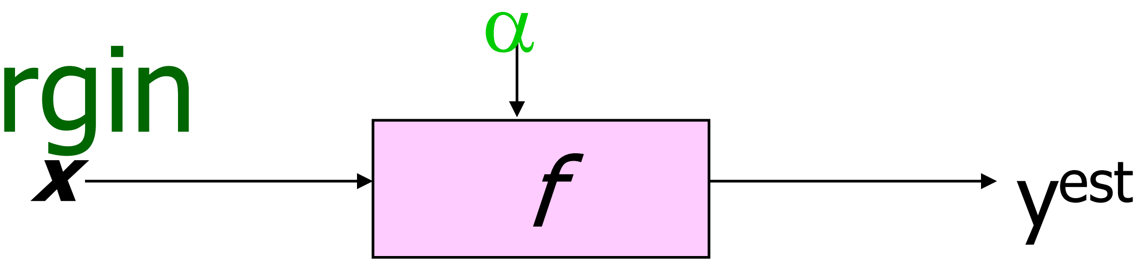
- denotes +1
- denotes -1



Any of these would be fine..

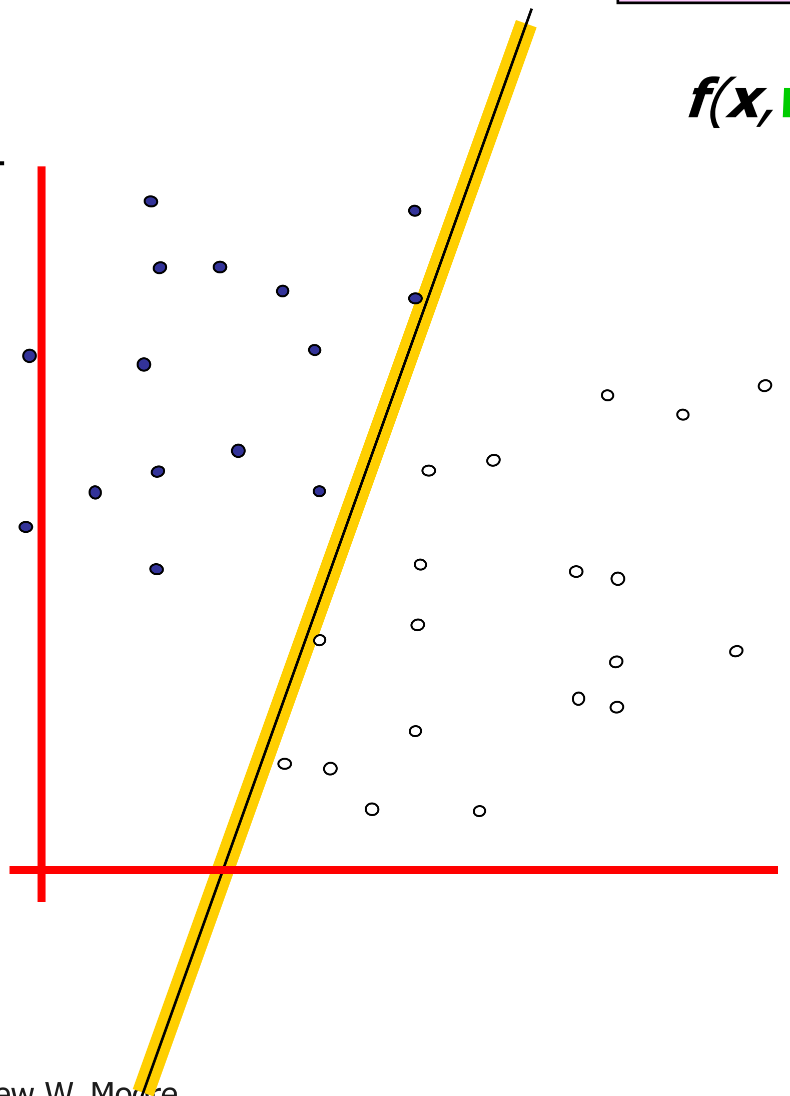
..but which is best?

Classifier Margin



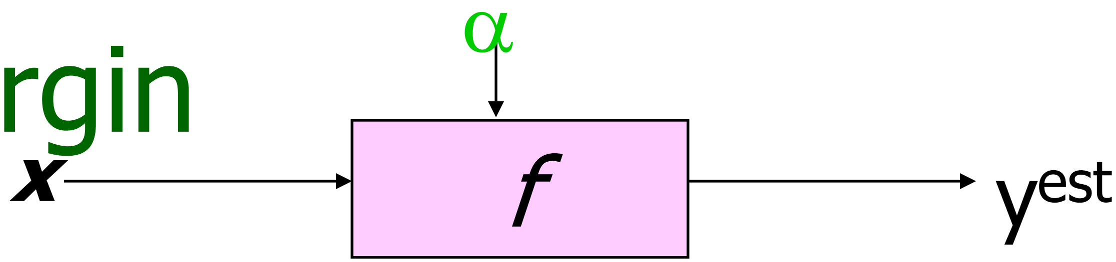
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$

- ● denotes +1
- ○ denotes -1

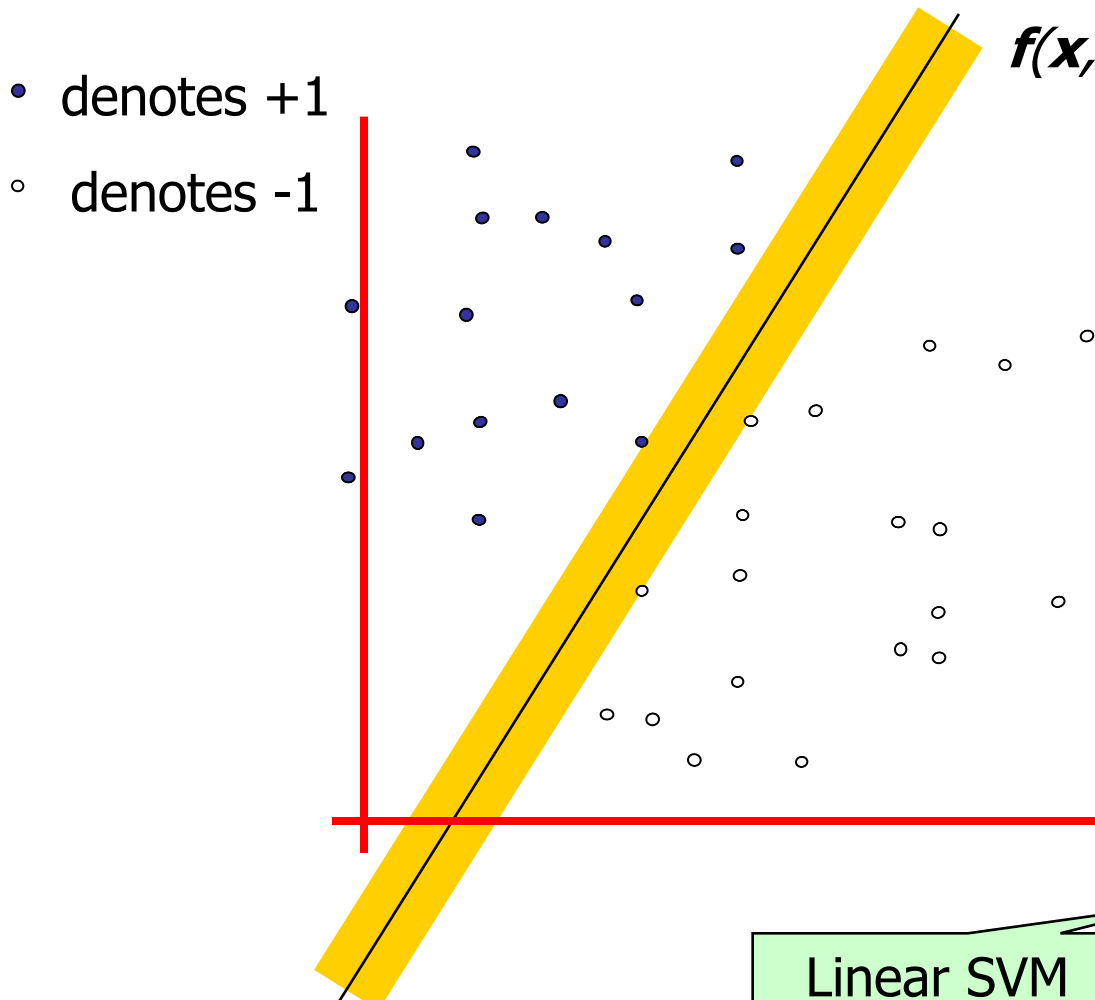


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

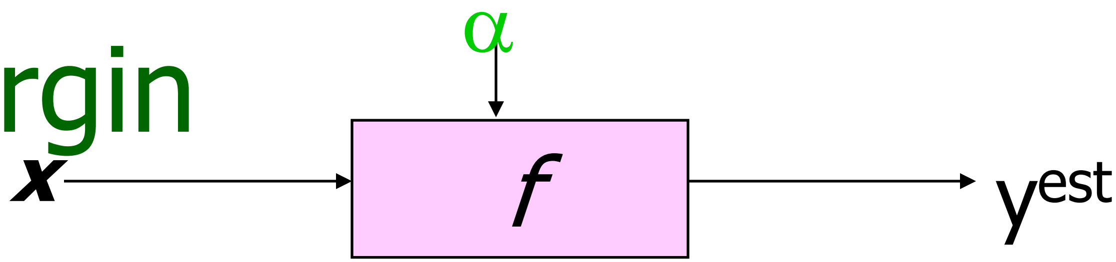


The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

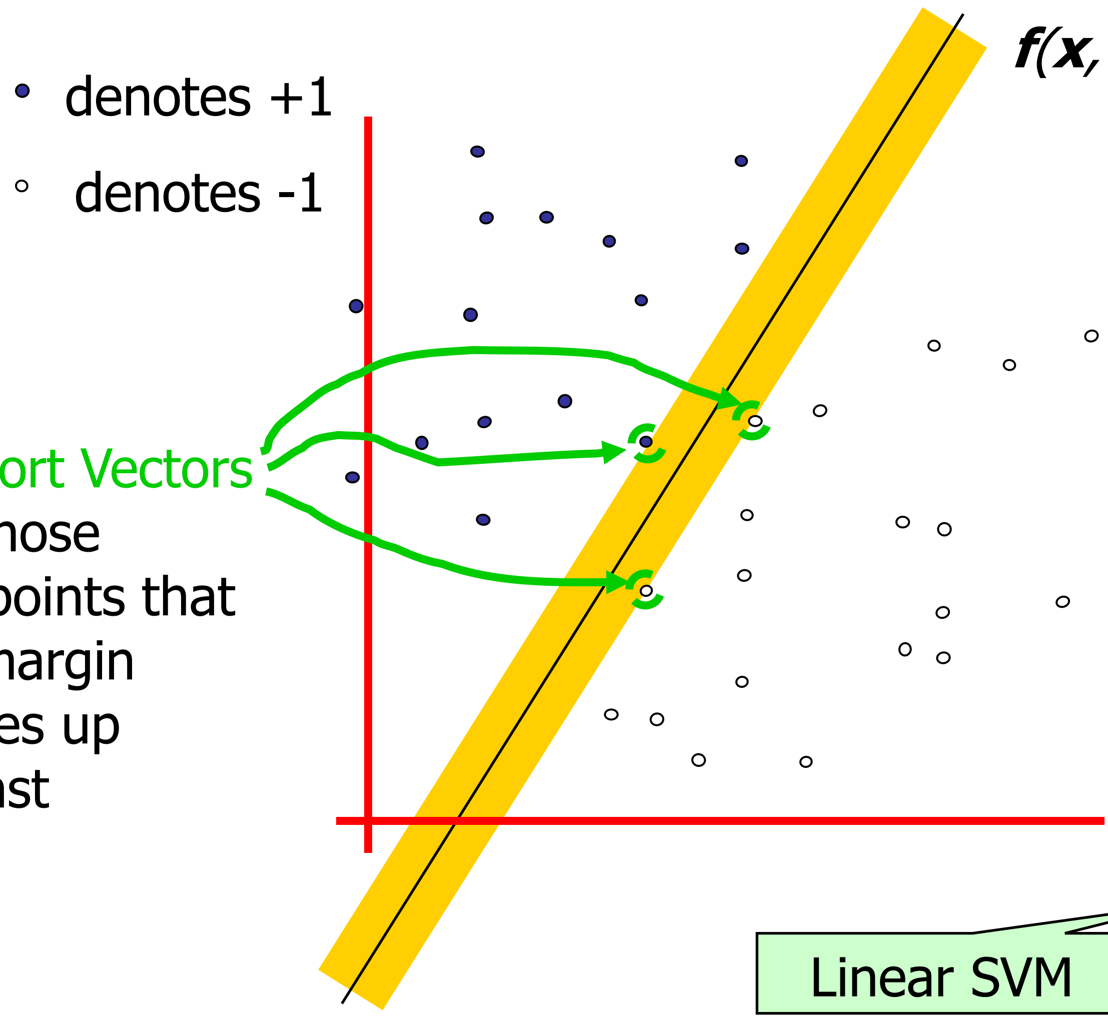
This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Maximum Margin



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



Support Vectors are those datapoints that the margin pushes up against

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

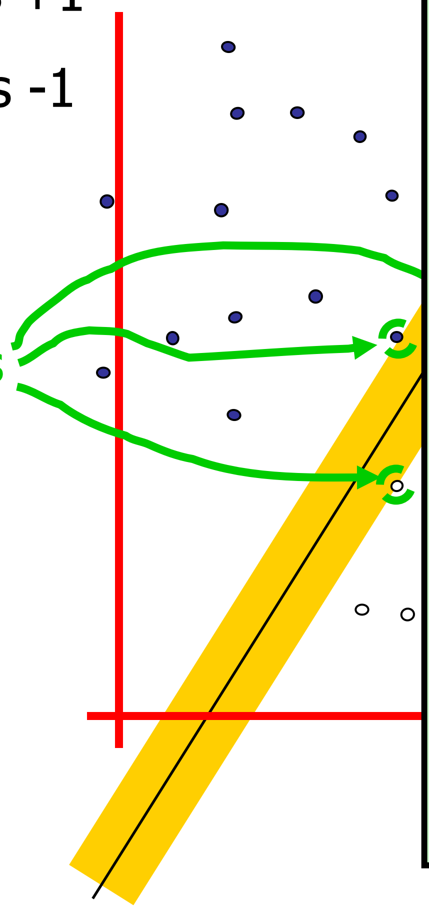
This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Why Maximum Margin?

• denotes +1

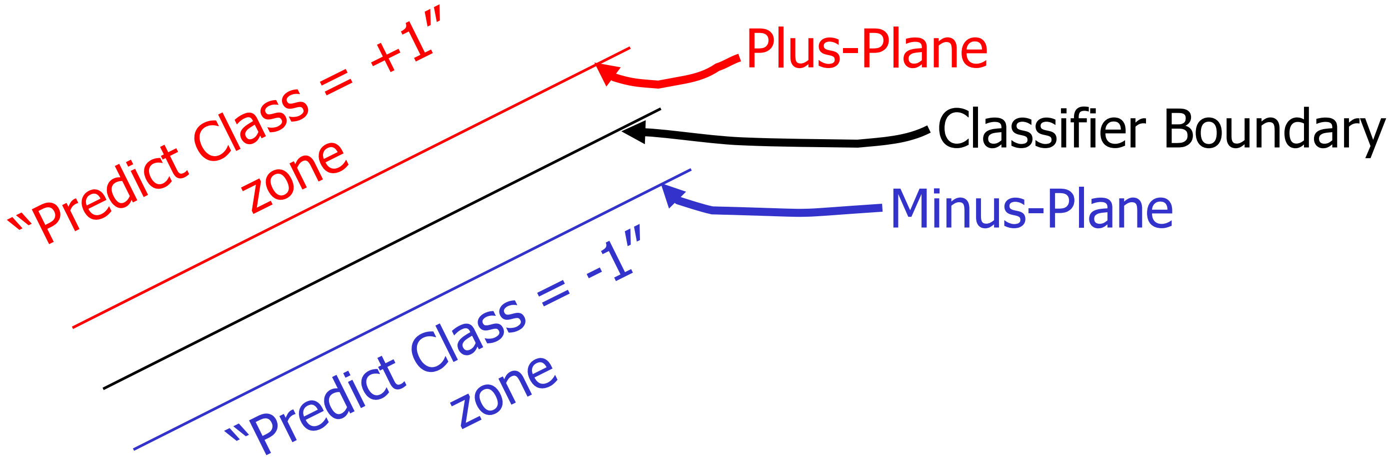
° denotes -1



Support Vectors are those datapoints that the margin pushes up against

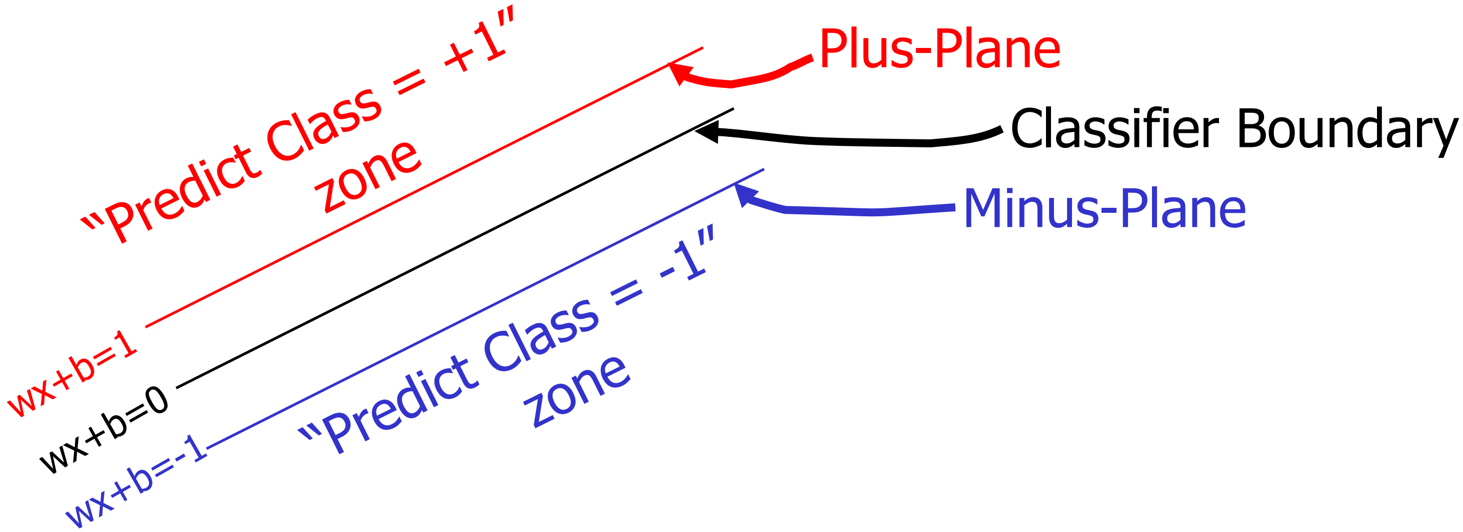
1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4.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

Specifying a line and margin



- How do we represent this mathematically?
- …in *m* input dimensions?

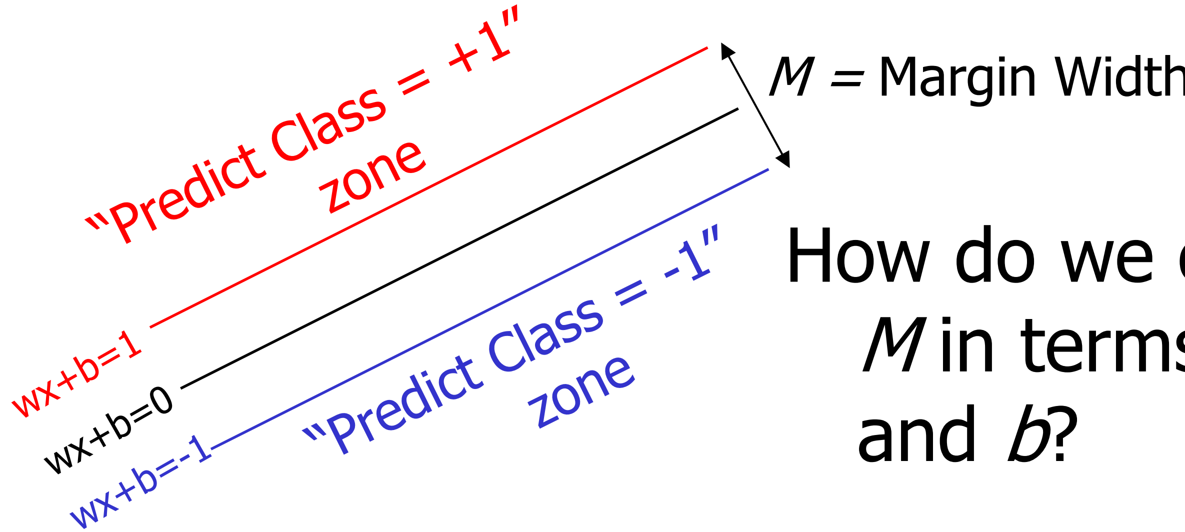
Specifying a line and margin



- 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	$\mathbf{w} \cdot \mathbf{x} + b >= 1$
	-1	if	$\mathbf{w} \cdot \mathbf{x} + b <= -1$
	Universe explodes	if	$-1 < \mathbf{w} \cdot \mathbf{x} + b < 1$

Computing the margin width

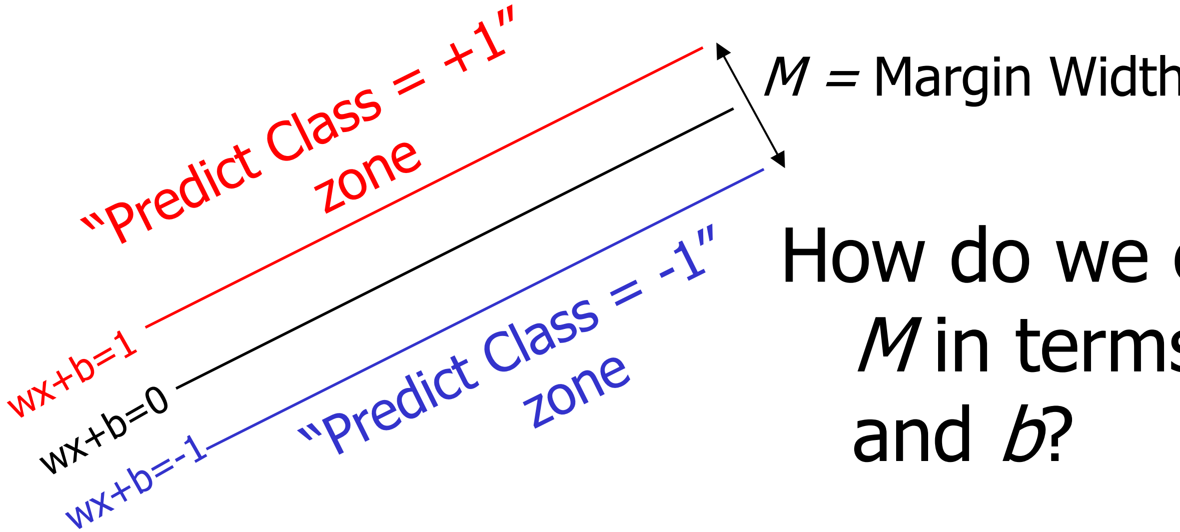


How do we compute *M* in terms of ***w*** and *b*?

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

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

Computing the margin width



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

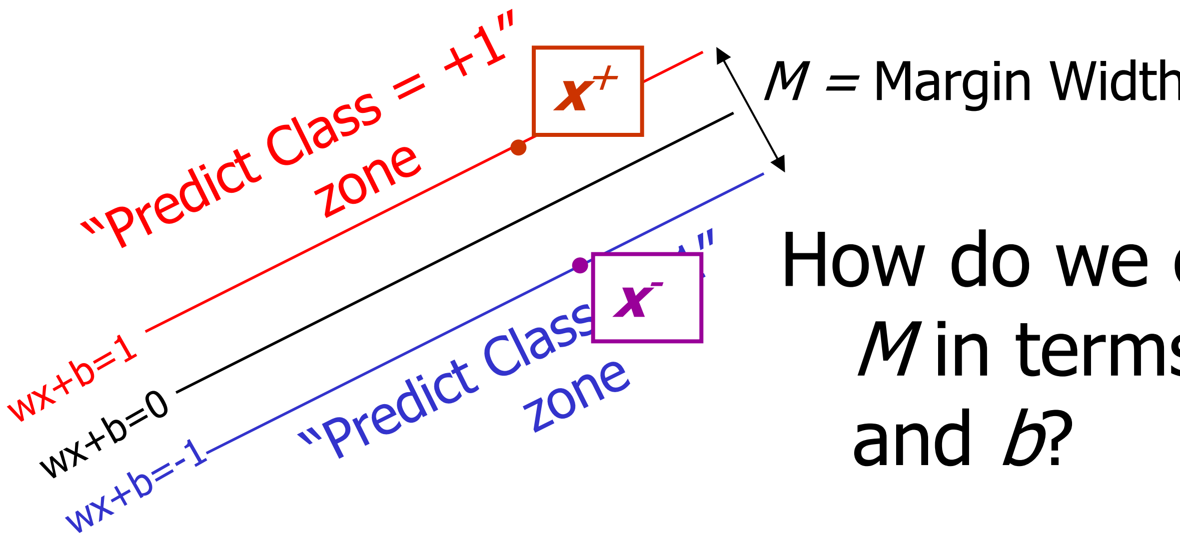
- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = -1 \}$

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

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

And so of course the vector **w** is also perpendicular to the Minus Plane

Computing the margin width

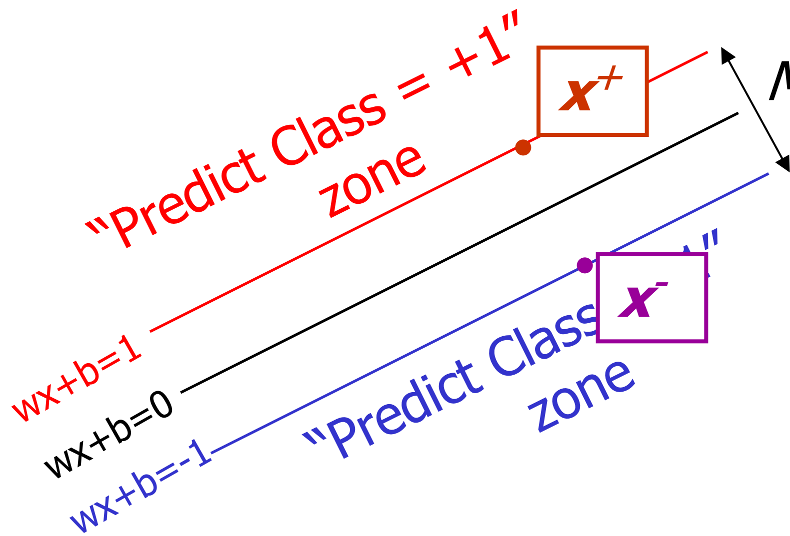


How do we compute M in terms of $\boldsymbol{w}$ and b?

- Plus-plane = $\{\, \mathbf{x} : \boldsymbol{w} \,.\, \mathbf{x} + b = +1 \,\}$
- Minus-plane = $\{\, \mathbf{x} : \boldsymbol{w} \,.\, \mathbf{x} + b = -1 \,\}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\boldsymbol{x}^-$ be any point on the minus plane
- Let $\boldsymbol{x}^+$ be the closest plus-plane-point to $\boldsymbol{x}^-$.

Any location in R^m: not necessarily a datapoint

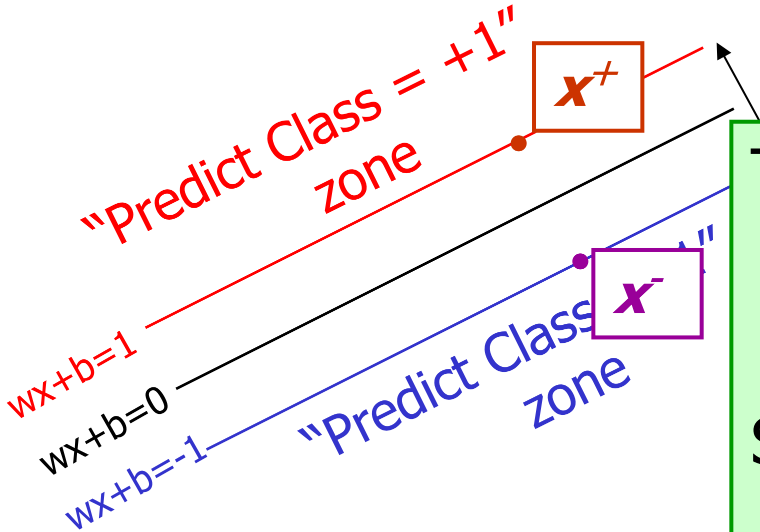
Computing the margin width



How do we compute M in terms of $\boldsymbol{w}$ and b?

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

Computing the margin width

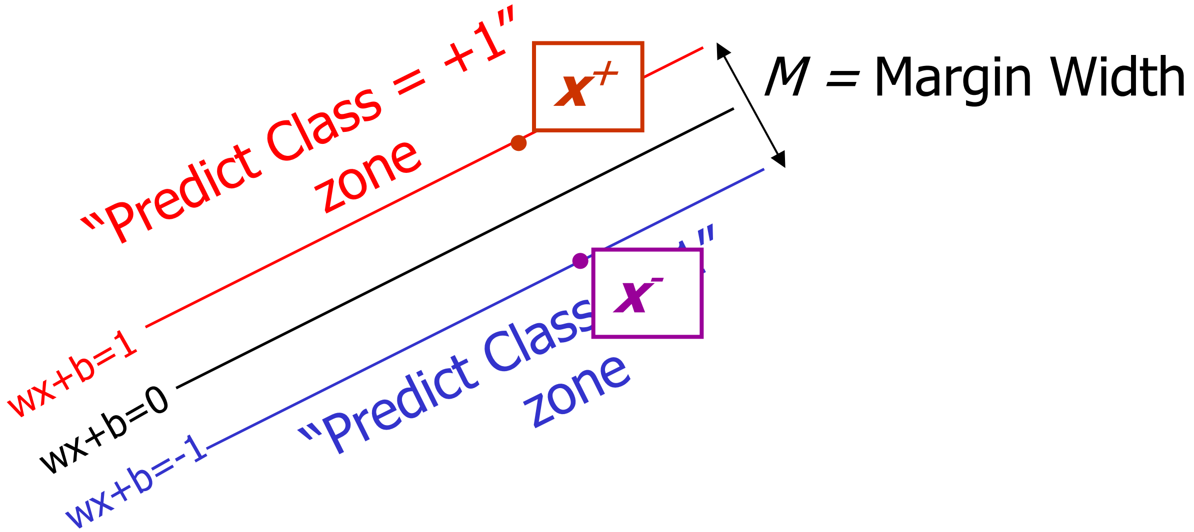


The line from $\boldsymbol{x^-}$ to $\boldsymbol{x^+}$ is perpendicular to the planes.

So to get from $\boldsymbol{x^-}$ to $\boldsymbol{x^+}$ travel some distance in direction $\boldsymbol{w}$.

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

Computing the margin width

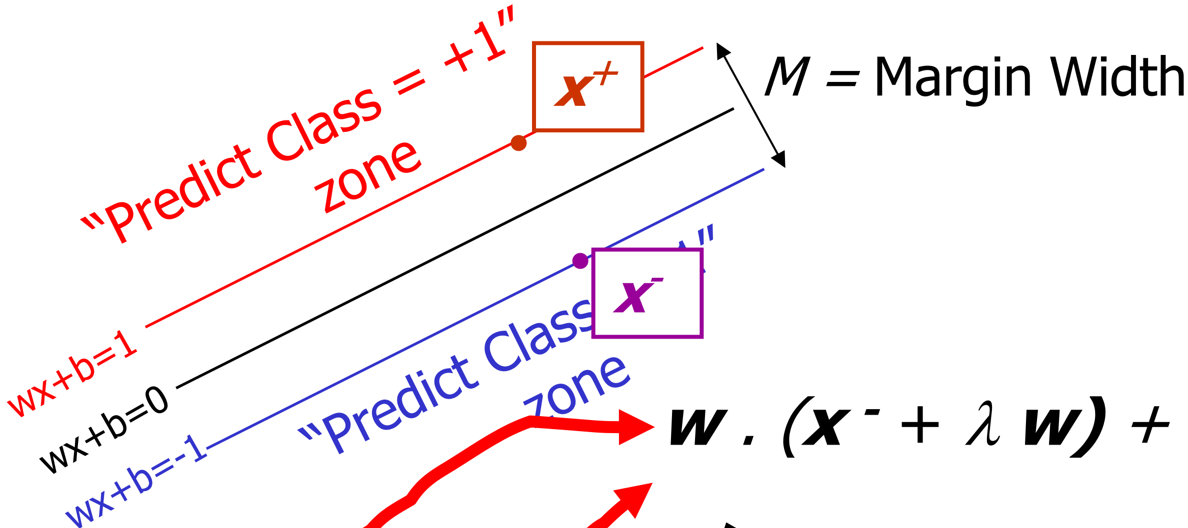


What we know:

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

It's now easy to get M in terms of $\mathbf{w}$ and b

Computing the margin width



What we know:

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

It's now easy to get M in terms of $\boldsymbol{w}$ and b

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

$\Rightarrow$

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

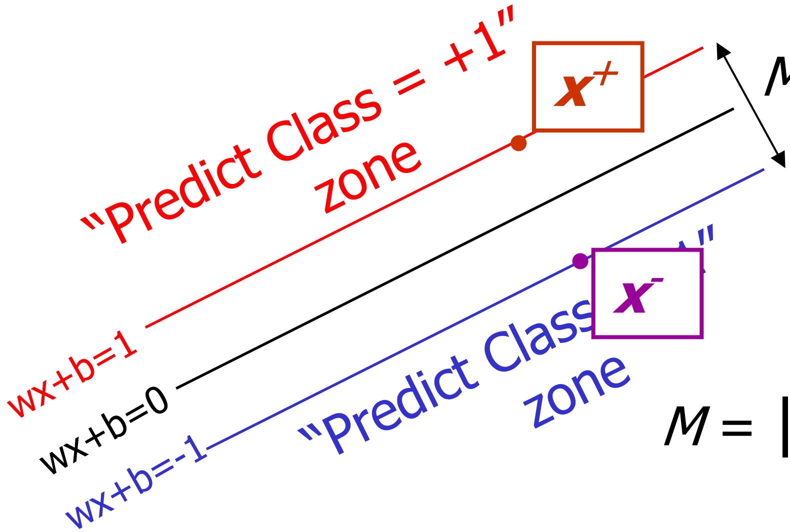
$\Rightarrow$

$$-1 + \lambda \boldsymbol{w} \cdot \boldsymbol{w} = 1$$

$\Rightarrow$

$$\lambda = \frac{2}{\mathbf{w.w}}$$

Computing the margin width



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

What we know:

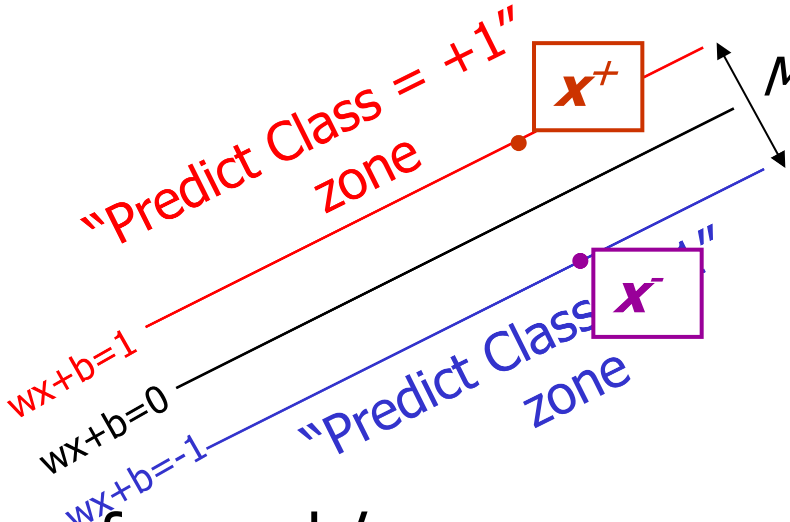
- $\mathbf{w}\,.\,\mathbf{x}^{+} + b = +1$
- $\mathbf{w}\,.\,\mathbf{x}^{-} + b = -1$
- $\mathbf{x}^{+} = \mathbf{x}^{-} + \lambda\,\mathbf{w}$
- $|\mathbf{x}^{+} - \mathbf{x}^{-}| = M$
- $$\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$$

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

$$= \lambda\,|\mathbf{w}| = \lambda\sqrt{\mathbf{w}.\mathbf{w}}$$

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

Learning the Maximum Margin Classifier



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

Given a guess of $\boldsymbol{w}$ and b we can

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

So now we just need to write 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?

Learning via Quadratic Programming

- QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

Quadratic Programming

Find $\underset{\mathbf{u}}{\arg\max}\ \ c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ← Quadratic criterion

Subject to

$$
\begin{aligned}
a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m &\le b_1 \\
a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m &\le b_2 \\
&\vdots \\
a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m &\le b_n
\end{aligned}
\Bigg\}
$$

n additional linear inequality constraints

And subject to

$$
\begin{aligned}
a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m &= b_{(n+1)} \\
a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m &= b_{(n+2)} \\
&\vdots \\
a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m &= b_{(n+e)}
\end{aligned}
\Bigg\}
$$

e additional linear equality constraints

Quadratic Programming

Find $\arg\max_{\mathbf{u}} \; c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R \mathbf{u}}{2}$ ← Quadratic criterion

Subject to u_1 …

…dditi…nal linear …equality …nstraints

And subj…

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

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

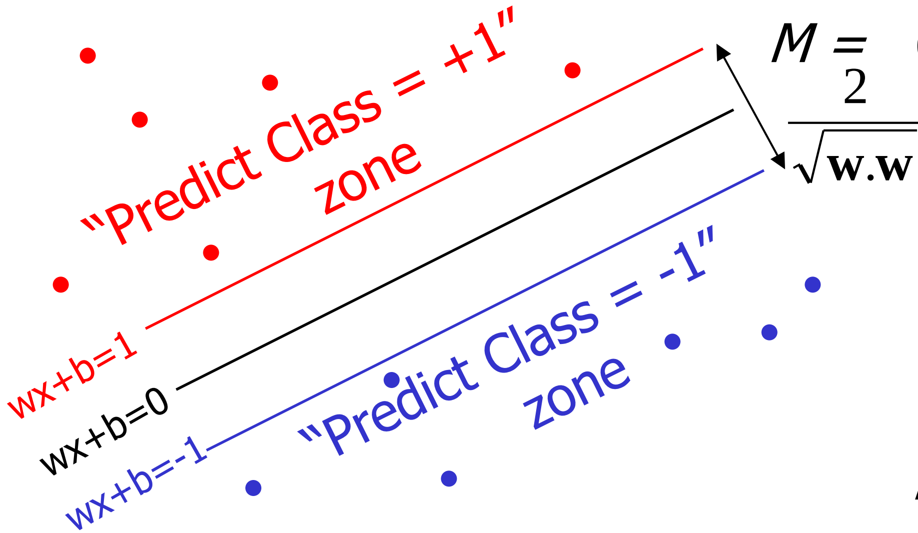
$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}$

e additional linear equality constraints

There exist algorithms 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)

Learning the Maximum Margin Classifier



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

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

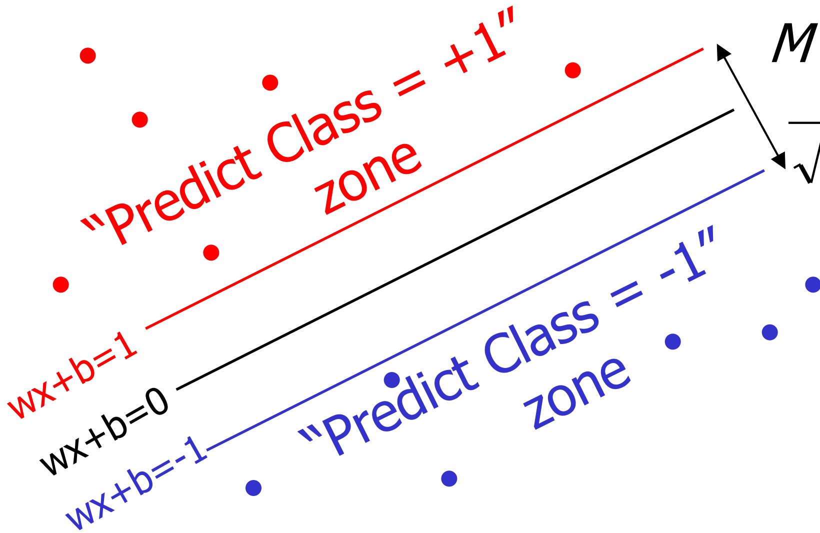
Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Learning the Maximum Margin Classifier



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

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

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

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\boldsymbol{w}.\boldsymbol{w}$

How many constraints will we have? R

What should they be?

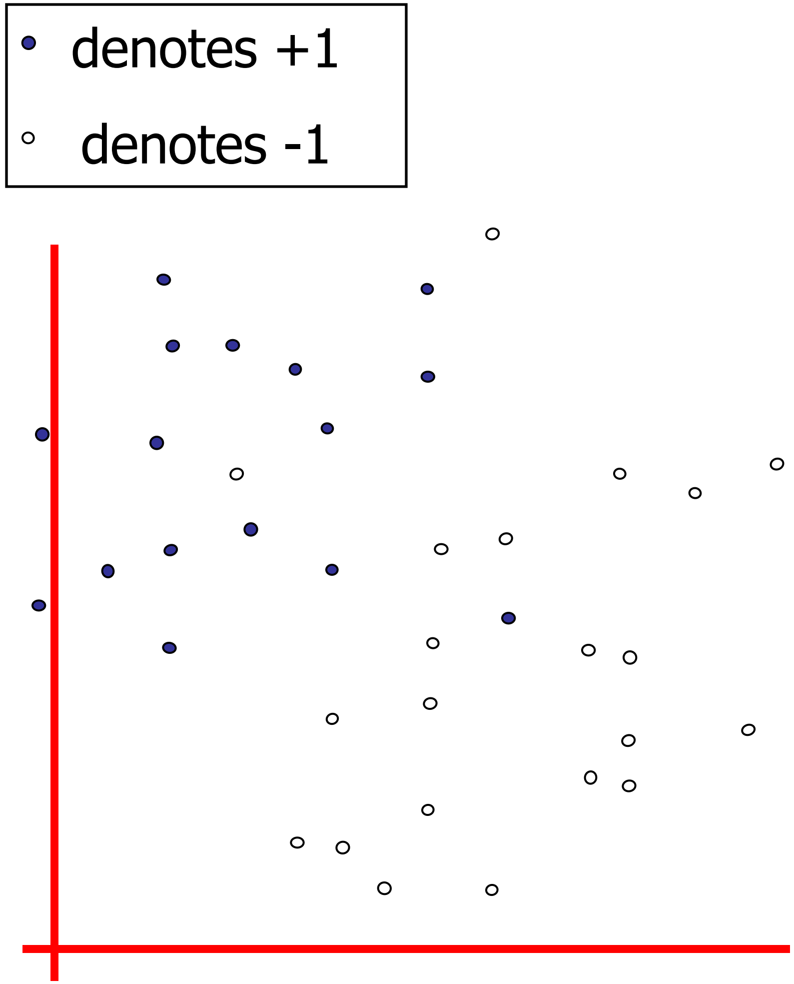
$\boldsymbol{w} . \boldsymbol{x}_k + b >= 1 \text{ if } y_k = 1$

$\boldsymbol{w} . \boldsymbol{x}_k + b <= -1 \text{ if } y_k = -1$

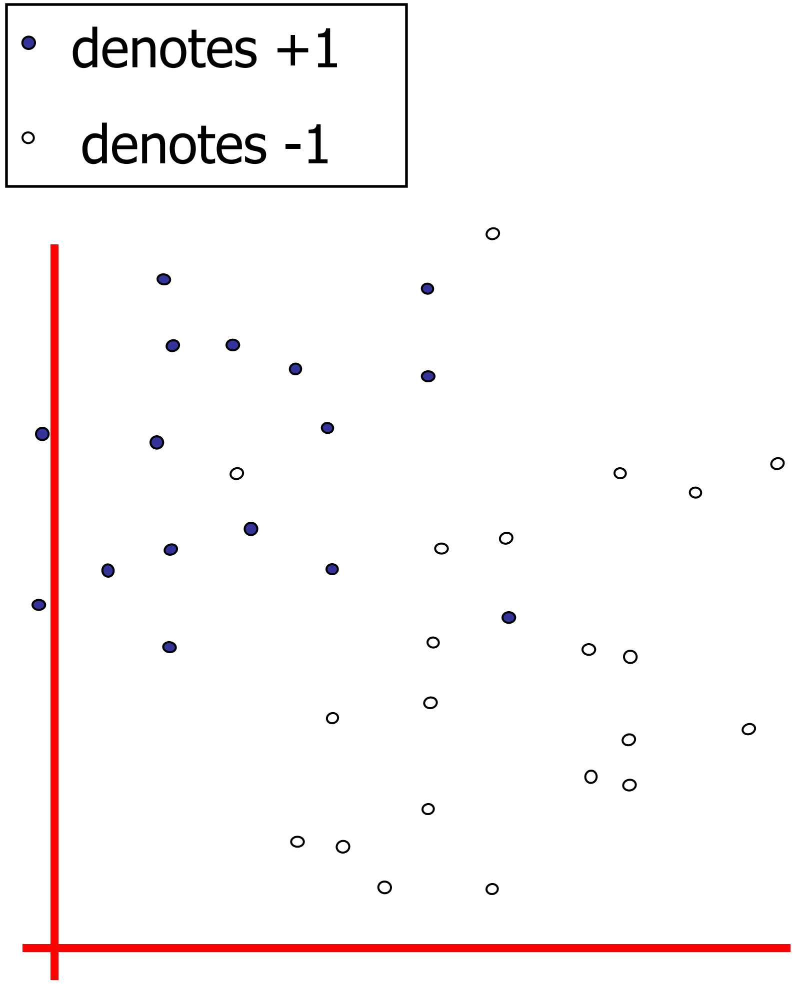
Uh-oh!

This is going to be a problem!

What should we do?



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: Two things to minimize makes for an ill-defined optimization

Uh-oh!

This is going to be a problem!

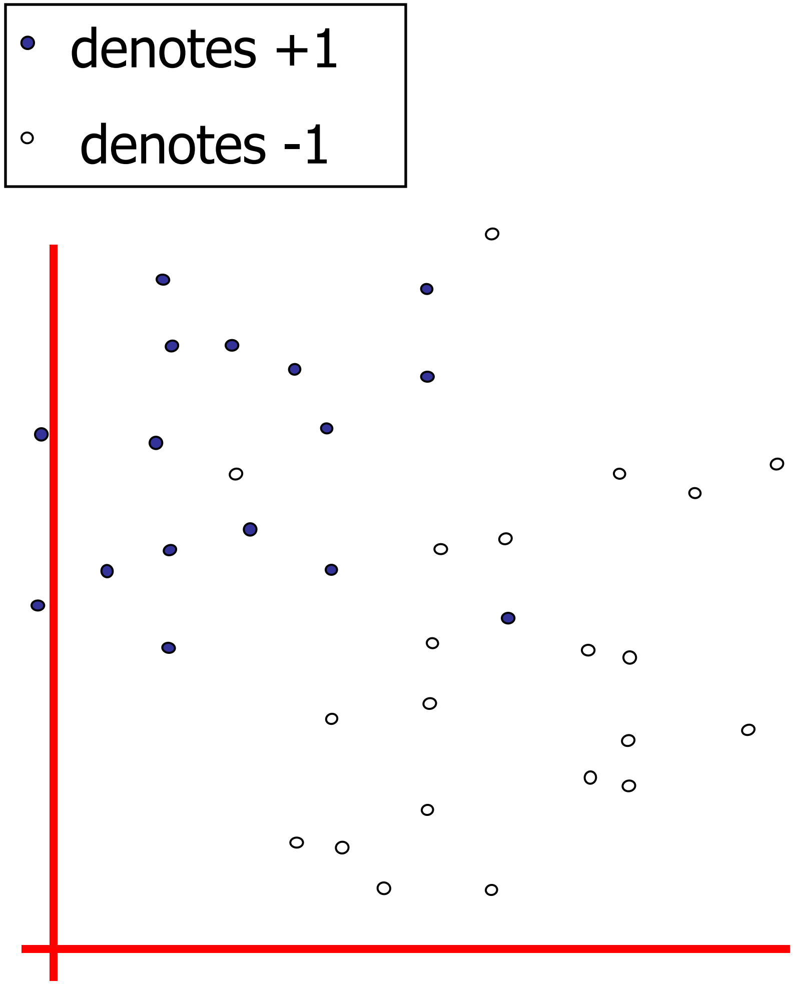
What should we do?

Idea 1.1:

Minimize

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

Tradeoff parameter



There's a serious practical problem that's about to make us reject this approach. Can you guess what it is?

Uh-oh!

This is going to be a problem!

What should we do?

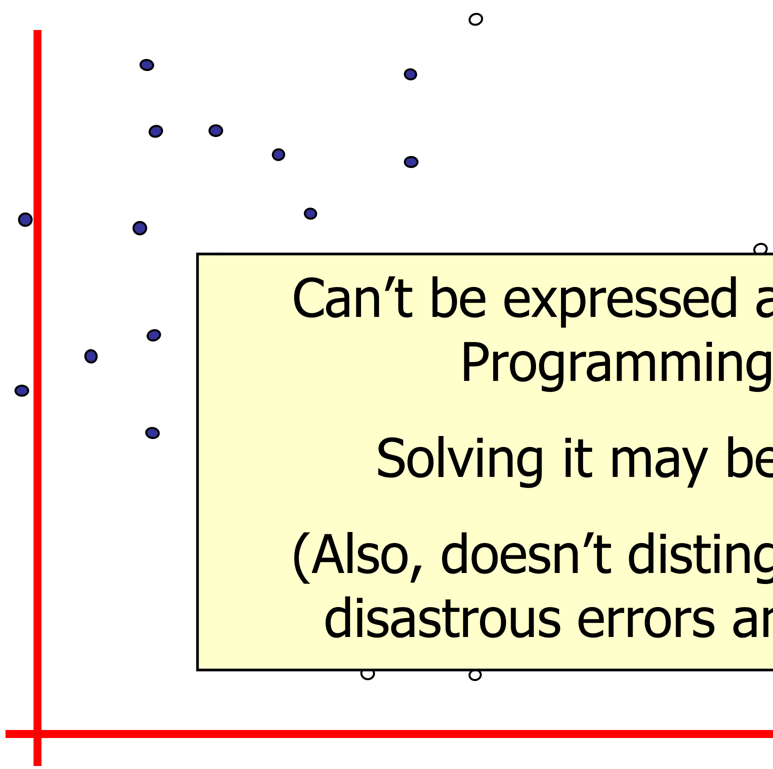
Idea 1.1:

Minimize

$\boldsymbol{w.w}$ + *C (#train errors)*

Tradeoff parameter

- denotes +1
- ◦ denotes -1



Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

(Also, doesn't distinguish between disastrous errors and near misses)

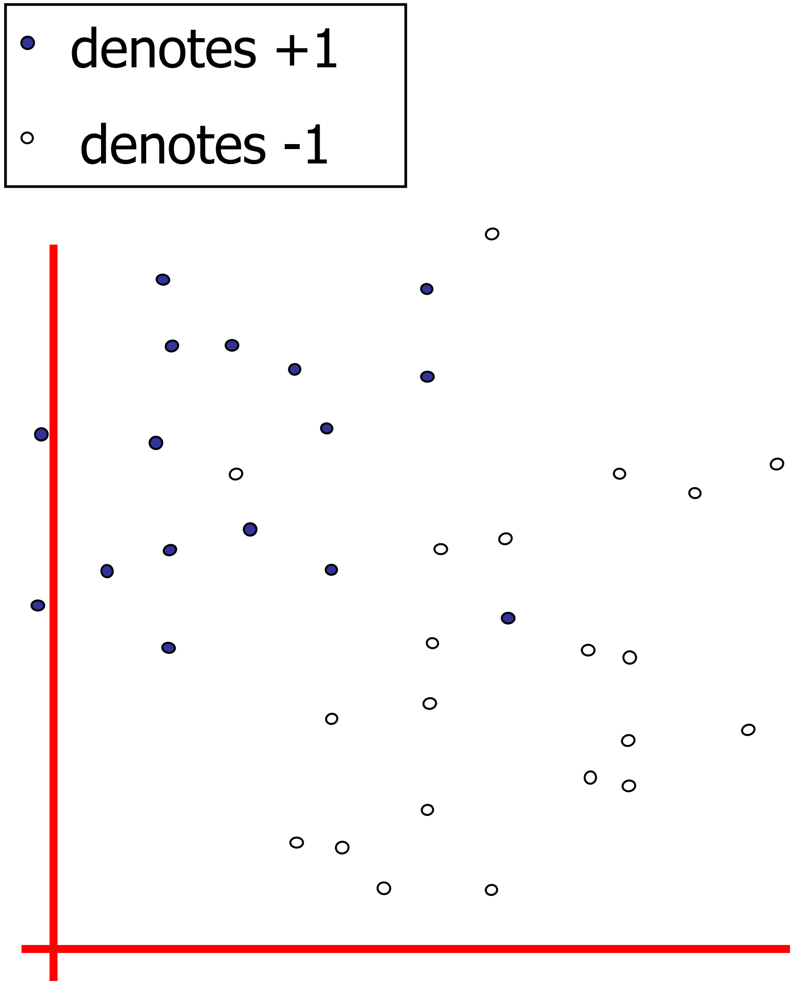
...erious... at'... ke

us reject th...

you guess wh...

So... any other ideas?

Uh-oh!



This is going to be a problem!

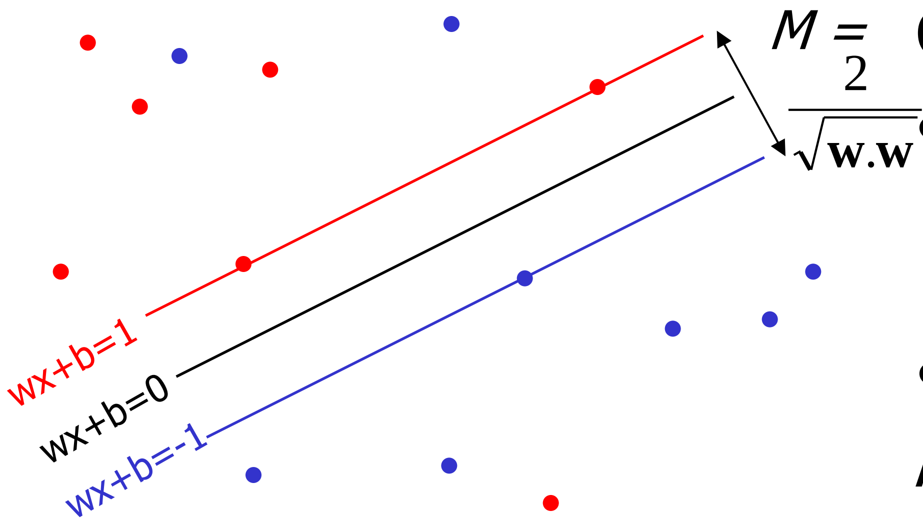
What should we do?

Idea 2.0:

Minimize

w.w *+ C (distance of error points to their correct place)*

Learning Maximum Margin with Noise



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

- Compute sum of distances of points to their correct zones
- Compute the margin width

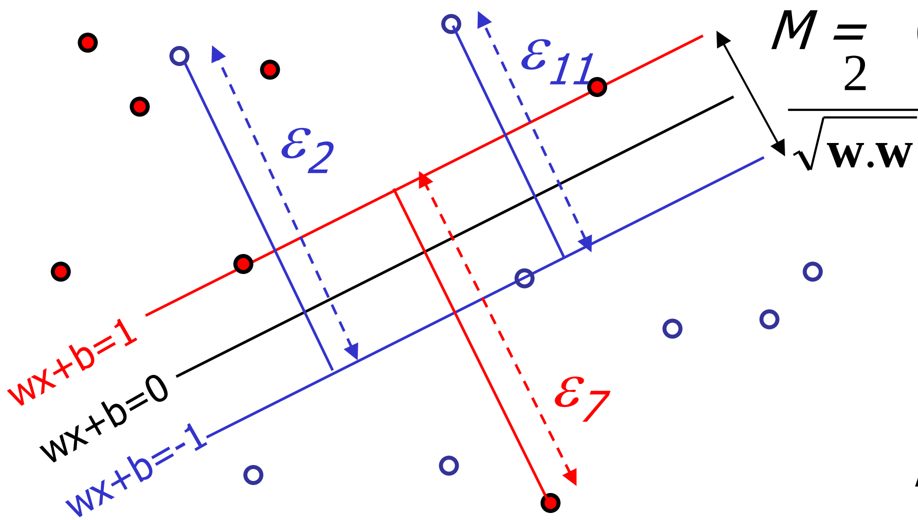
Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Learning Maximum Margin with Noise



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

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

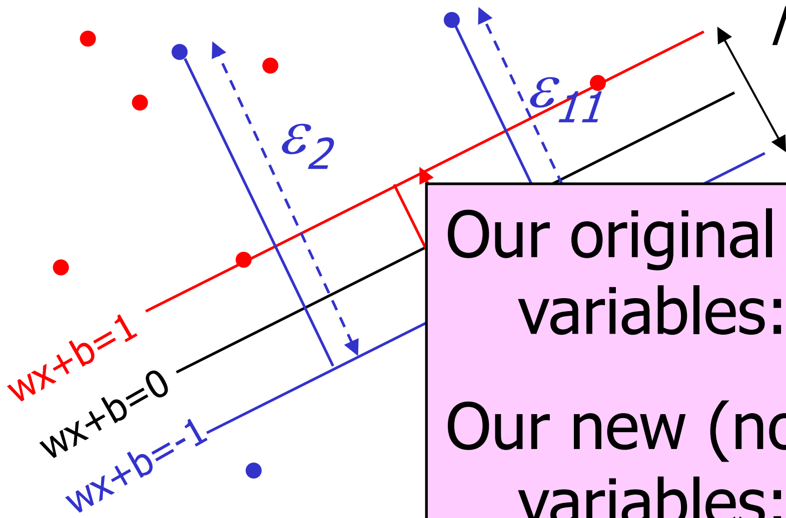
How many constraints will we have? R

What should they be?

$\boldsymbol{w} \, . \, \boldsymbol{x}_k + b >= 1\text{-}\varepsilon_k$ *if* $y_k = 1$

$\boldsymbol{w} \, . \, \boldsymbol{x}_k + b <= -1+\varepsilon_k$ *if* $y_k = -1$

Learning Maximum Margin with Noise



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

Given gu… we can

- Compute sum of distances …

…dth

$(\mathbf{x}_k, y_k)$ where y_k … 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: $w_1, w_2, \ldots w_m, b, \varepsilon_k, \varepsilon_1, \ldots \varepsilon_R$

R= # records

What should our quadratic optimization criterion be?

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

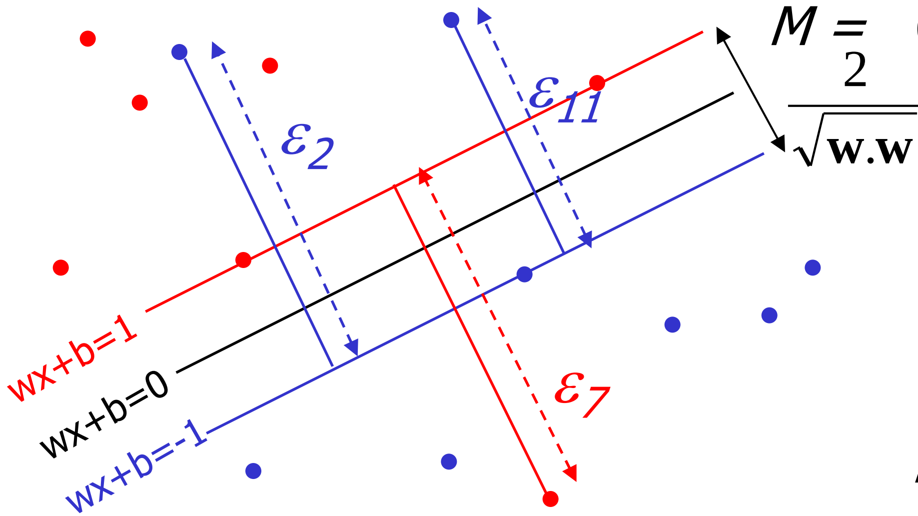
How many constraints … we have? R

What should they be?

$\mathbf{w} \,.\, \mathbf{x}_k + b >= 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} \,.\, \mathbf{x}_k + b <= -1+\varepsilon_k$ if $y_k = -1$

Learning Maximum Margin with Noise



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

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

How many constraints will we have? R

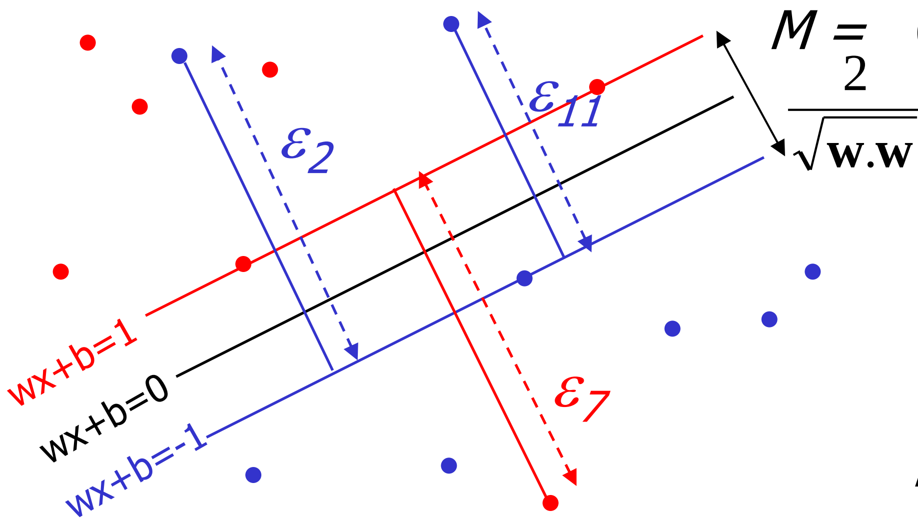
What should they be?

$\boldsymbol{w} \,.\, \mathbf{x}_k + b >= 1\text{-}\varepsilon_k$ if $y_k = 1$

$\boldsymbol{w} \,.\, \mathbf{x}_k + b <= -1+\varepsilon_k$ if $y_k = -1$

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

Learning Maximum Margin with Noise



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

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

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

How many constraints will we have? *2R*

What should they be?

$\mathbf{w} \,.\, \mathbf{x}_k + b >= 1 - \varepsilon_k$ if $y_k = 1$

$\mathbf{w} \,.\, \mathbf{x}_k + b <= -1 + \varepsilon_k$ if $y_k = -1$

$\varepsilon_k >= 0$ for all k

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

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:

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

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

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

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. *x* - *b*)**

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

Subject to these constraints:

$$0 \le \alpha_k \le C \quad \forall k \qquad \sum_{k=1}^{R} \alpha_k y_k = 0$$

Datapoints with $\alpha_k > 0$ will be the support vectors

Then define:

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

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

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

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

f(**x**,**w**,*b*) = *sign*(**w**. **x** - *b*)

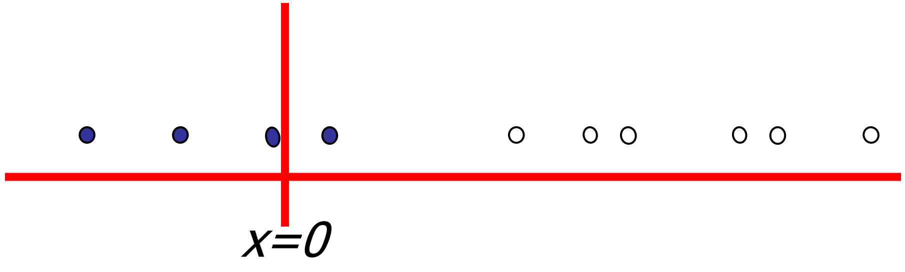
An Equivalent QP

Why did I tell you about this equivalent QP?

- It's a formulation that QP packages can optimize more quickly
- Because of further jaw-dropping developments you're about to learn.

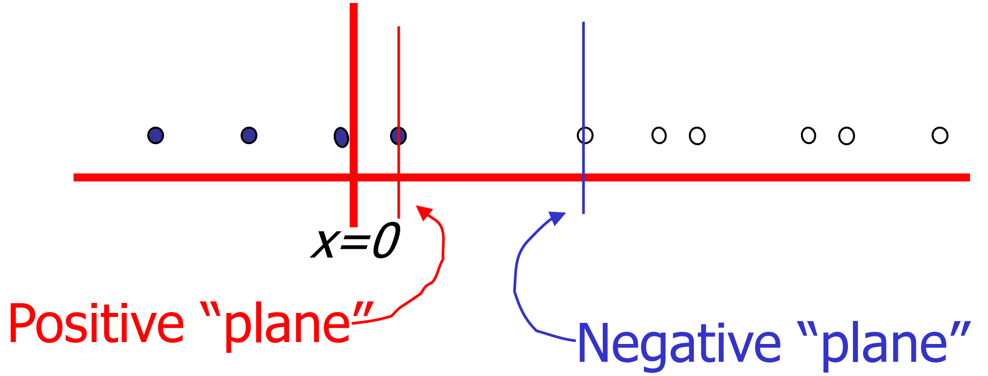
Suppose we're in 1-dimension

What would SVMs do with this data?

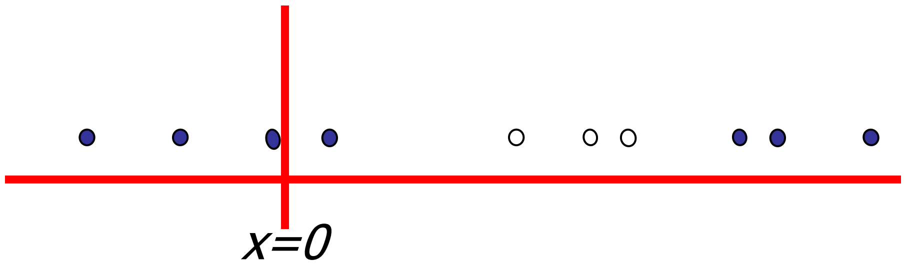


Suppose we're in 1-dimension

Not a big surprise



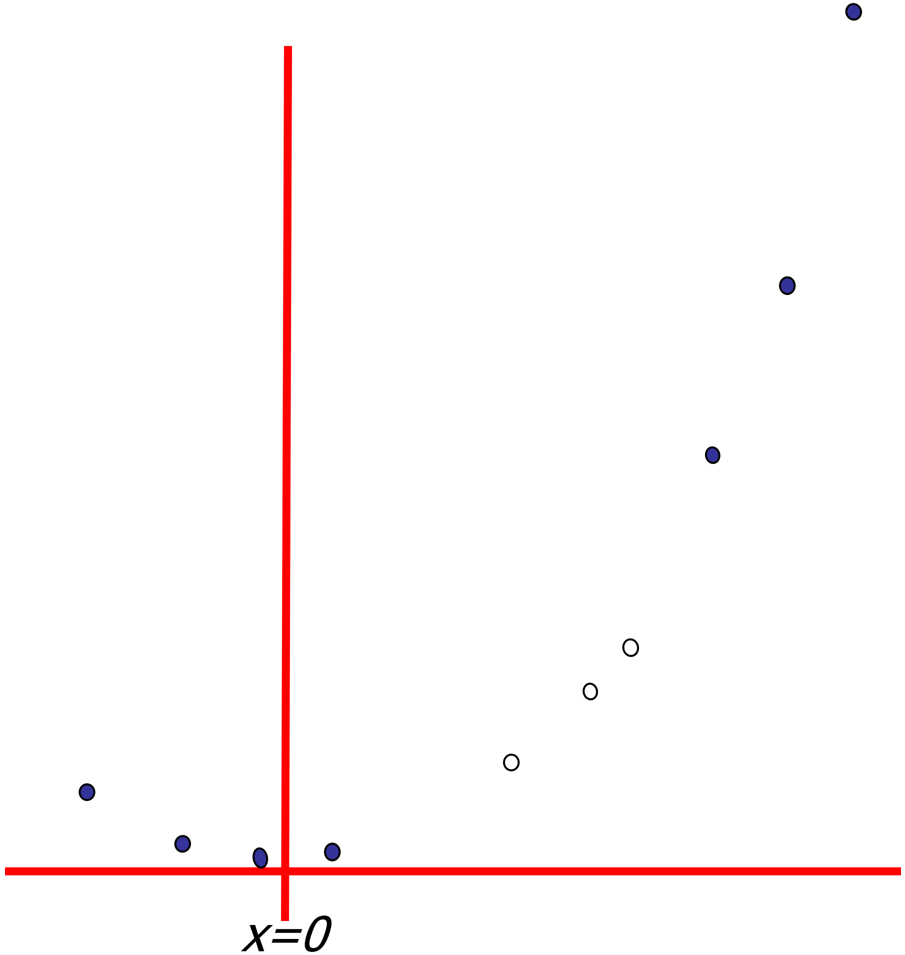
Harder 1-dimensional dataset



That's wiped the smirk off SVM's face.

What can be done about this?

Harder 1-dimensional dataset

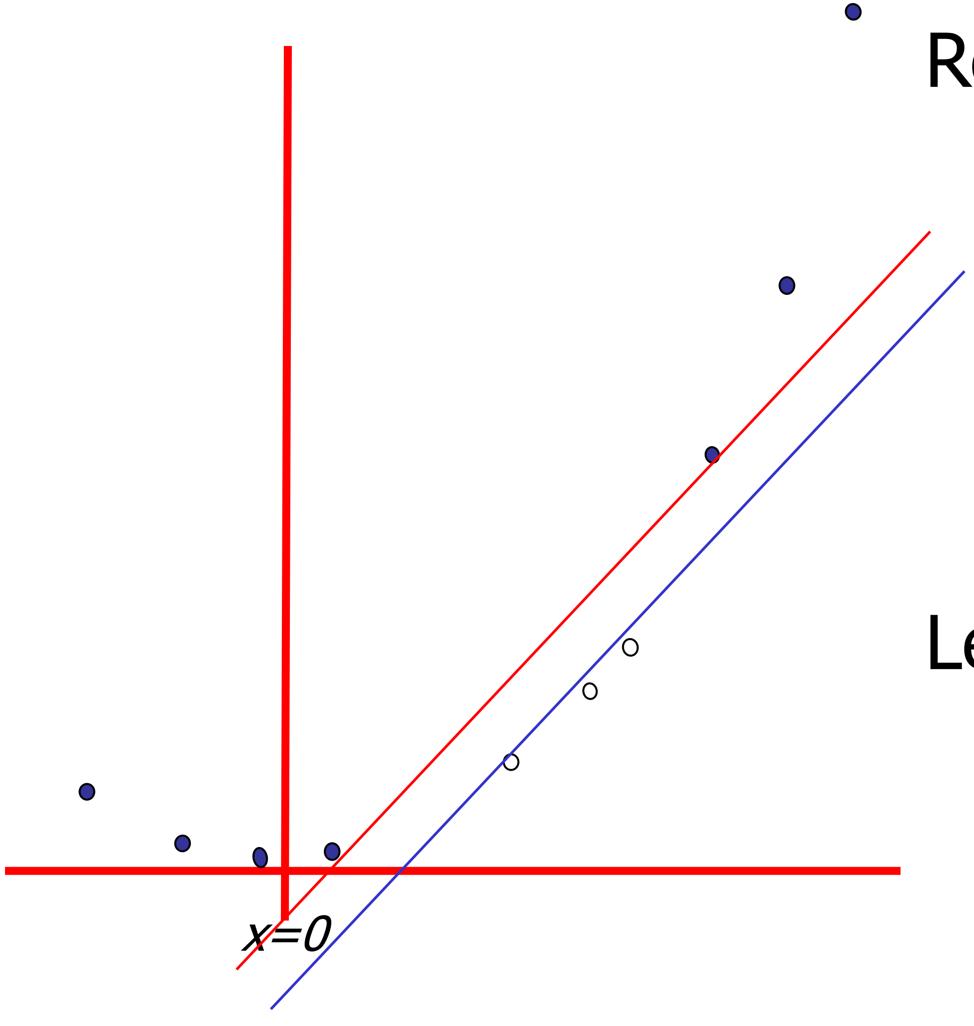


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

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

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

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

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \mathrm{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\mathrm{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

This is sensible.

Is that the end of the story?

No…there's one more trick!

Quadratic Basis Functions

$$\Phi(\mathbf{x}) = \left(\begin{array}{cl}
\left. 1 \right\} & \text{Constant Term} \\
\left. \begin{matrix} \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \end{matrix} \right\} & \text{Linear Terms} \\
\left. \begin{matrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \end{matrix} \right\} & \text{Pure Quadratic Terms} \\
\left. \begin{matrix} \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{matrix} \right\} & \text{Quadratic Cross-Terms}
\end{array} \right)$$

Number of terms (assuming m input dimensions) = (m+2)-choose-2

= (m+2)(m+1)/2

= (as near as makes no difference) $m^2/2$

You may be wondering what those $\sqrt{2}$'s are doing.

- You should be happy that they do no harm
- You'll find out why they're there soon.

QP with basis functions

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

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

Then define:

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

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

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

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. ϕ(*x*) - *b*)**

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

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

Then define:

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

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

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

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product requires $m^2/2$ additions and multiplications

The whole thing costs $R^2 m^2 /4$. Yeeks!

...or does it?

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

Quadratic Dot Products

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

$$1$$

$$+$$

$$\sum_{i=1}^{m} 2a_ib_i$$

$$+$$

$$\sum_{i=1}^{m} a_i^2 b_i^2$$

$$+$$

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

Quadratic Dot Products

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

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

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

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

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

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

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

Quadratic Dot Products

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

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

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

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

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

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

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

They're the same!

And this is only O(m) to compute!

QP with Quadratic basi

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

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

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product now only requires *m* additions and multiplications

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

Then define:

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

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

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

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ϕ(***x***) - *b*)

Higher Order Polynomials

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

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. *What are they?*

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

$\forall k$

$$\sum_{k=1}^{R} \alpha_k y_k = 0$$

constraints:

Then define:

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

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

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

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. ϕ(*x*) - *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

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

$\forall k \qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

constraints:

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive *(why?)*

Then define:

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

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

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

Then classify with:

$\boldsymbol{f}(\boldsymbol{x},\boldsymbol{w},b) = sign(\boldsymbol{w}.\,\phi(\boldsymbol{x}) - b)$

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

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

The use of Maximum Margin magically makes this not a problem

$\forall k \qquad \sum \alpha_k y_k = 0$

constraints:

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Because each **w**. ϕ(**x**) (see below) needs 75 million operations. *What can be done?*

Then define:

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

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

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

Then classify with:

***f*(x,w,*b*) = *sign*(w. ϕ(x) - *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

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

The use of Maximum Margin magically makes this not a problem

constraints:

$$\forall k \qquad \sum \alpha_k y_k = 0$$

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Because each **w**. ϕ(**x**) (see below) needs 75 million operations. What can be done?

Then define:

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

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

Only *Sm* operations (*S*=#support vectors)

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. ϕ(*x*) - *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

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

The use of Maximum Margin magically makes this not a problem

$\forall k$ $\sum \alpha_k y_k = 0$

constraints:

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Then define:

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

Because each **w**. $\phi(\mathbf{x})$ (see below) needs 75 million operations. What can be done?

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

Only *Sm* operations (*S*=#support vectors)

Th

f(

When you see this many callout bubbles on a slide it's time to wrap the author in a blanket, gently take him away and murmur "someone's been at the PowerPoint for too long."

QP with Quintic basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ wh

Andrew's opinion of why SVMs don't overfit as much as you'd think:

No matter what the basis function, there are really only up to R parameters: $\alpha_1, \alpha_2 .. \alpha_R$, and usually most are set to zero by the Maximum Margin.

Asking for small **w.w** is like "weight decay" in Neural Nets and like Ridge Regression parameters in Linear regression and like the use of Priors in Bayesian Regression---all designed to smooth the function and reduce overfitting.

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

Then define:

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

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

Only *Sm* operations (*S*=#support vectors)

Then classify with:

***f*(*x*,*w*,*b*) = *sign*(*w*. φ(*x*) - *b*)**

SVM Kernel Functions

- $K(\mathbf{a},\mathbf{b})=(\mathbf{a} \cdot \mathbf{b} +1)^d$ is an example of an SVM Kernel Function
- Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function
  - Radial-Basis-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

  - Neural-net-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \tanh(\kappa \mathbf{a}.\mathbf{b} - \delta)$$

σ, κ and δ are magic parameters that must be chosen by a model selection method such as CV or VCSRM*

*see last lecture

VC-dimension of an SVM

- Very very very loosely speaking there is some theory which under some different assumptions puts an upper bound on the VC dimension as

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

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

SVM Performance

- Anecdotally they work very very well indeed.
- Example: They are currently the best-known classifier on a well-studied hand-written-character recognition benchmark
- Another Example: Andrew knows several reliable people doing practical real-world work who claim that SVMs have saved them when their other favorite classifiers did poorly.
- There is a lot of excitement and religious fervor about SVMs as of 2001.
- Despite this, some practitioners (including your lecturer) 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 find out which one puts the prediction the furthest into the positive region.

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

What You Should Know

- Linear SVMs
- The definition of a maximum margin classifier
- What QP can do for you (but, for this class, you don’t need to know how it does it)
- How Maximum Margin can be turned into a QP problem
- How we deal with noisy (non-separable) data
- How we permit non-linear boundaries
- How SVM Kernel functions permit us to pretend we’re working with ultra-high-dimensional basis-function terms