

Machine Learning (CS 567)

Fall 2008

Time: T-Th 5:00pm - 6:20pm

Location: GFS 118

Instructor: Sofus A. Macskassy (macskass@usc.edu)

Office: SAL 216

Office hours: by appointment

Teaching assistant: Cheol Han (cheolhan@usc.edu)

Office: SAL 229

Office hours: M 2-3pm, W 11-12

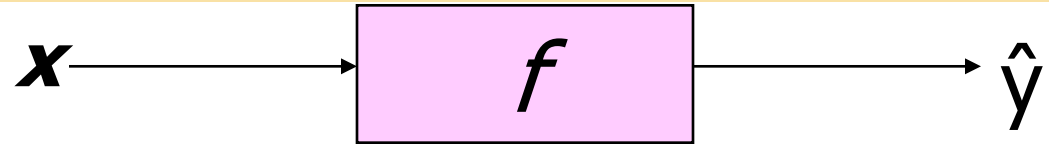
Class web page:

<http://www-scf.usc.edu/~csci567/index.html>

Support Vector Machines

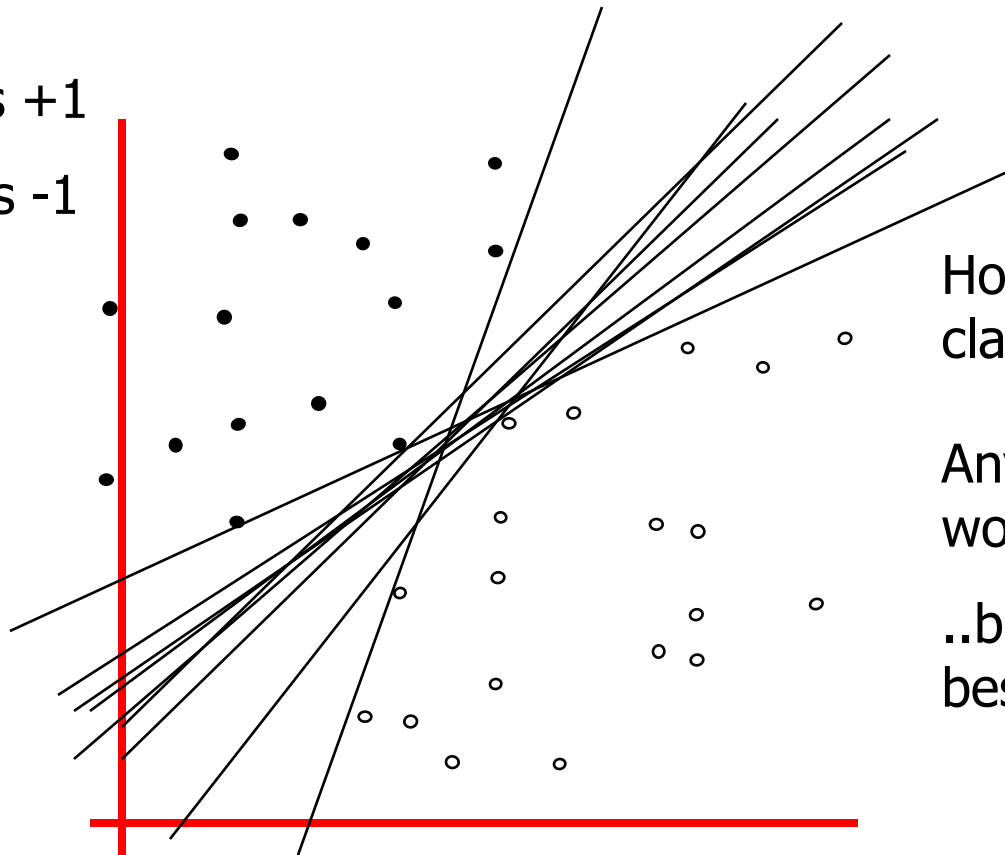
- Hypothesis Space
 - variable size
 - deterministic
 - continuous parameters
- Learning Algorithm
 - linear and quadratic programming
 - eager
 - batch
- SVMs combine three important ideas
 - Apply optimization algorithms from Operations Research (Linear Programming and Quadratic Programming)
 - Implicit feature transformation using kernels
 - Control of overfitting by maximizing the margin

Linear Classifiers



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

- denotes +1
- denotes -1

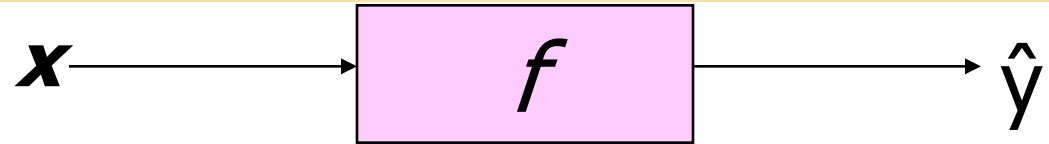


How would you classify this data?

Any of these would be fine..

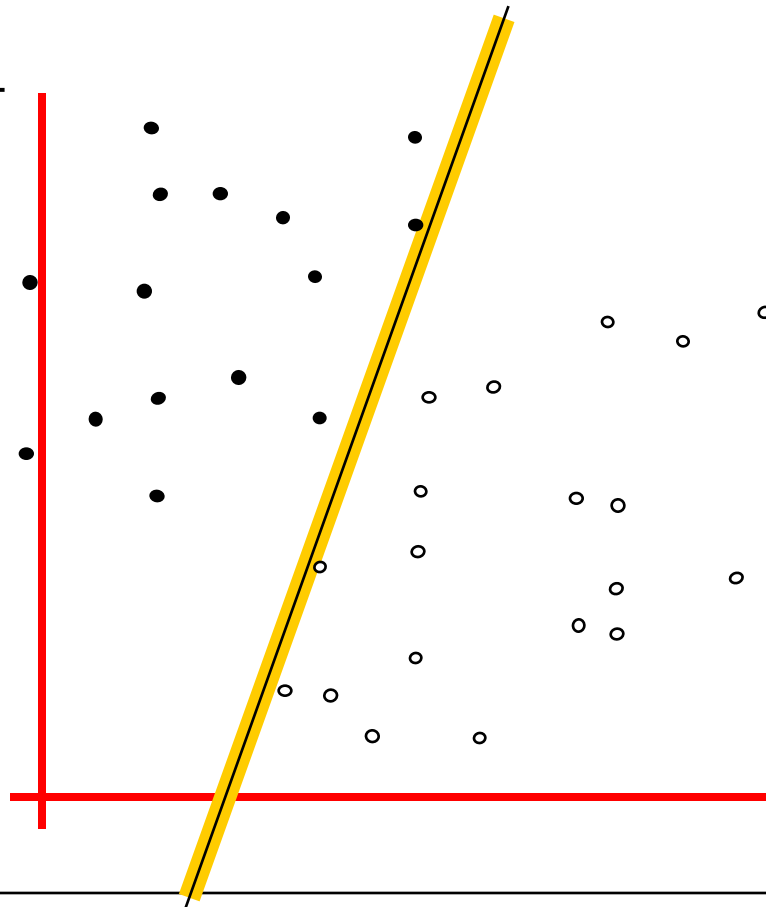
..but which is best?

Classifier Margin



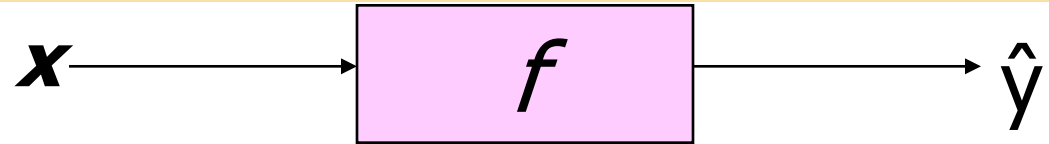
$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} - b)$$

- 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.

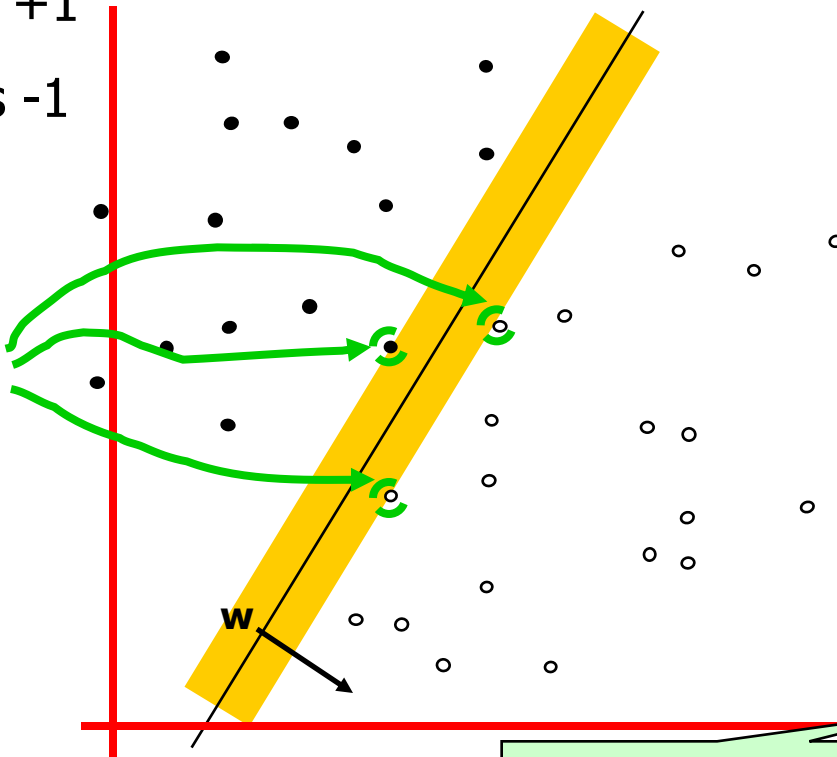
Maximum Margin



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

- denotes +1
- denotes -1

Support Vectors are those datapoints that the margin pushes up against



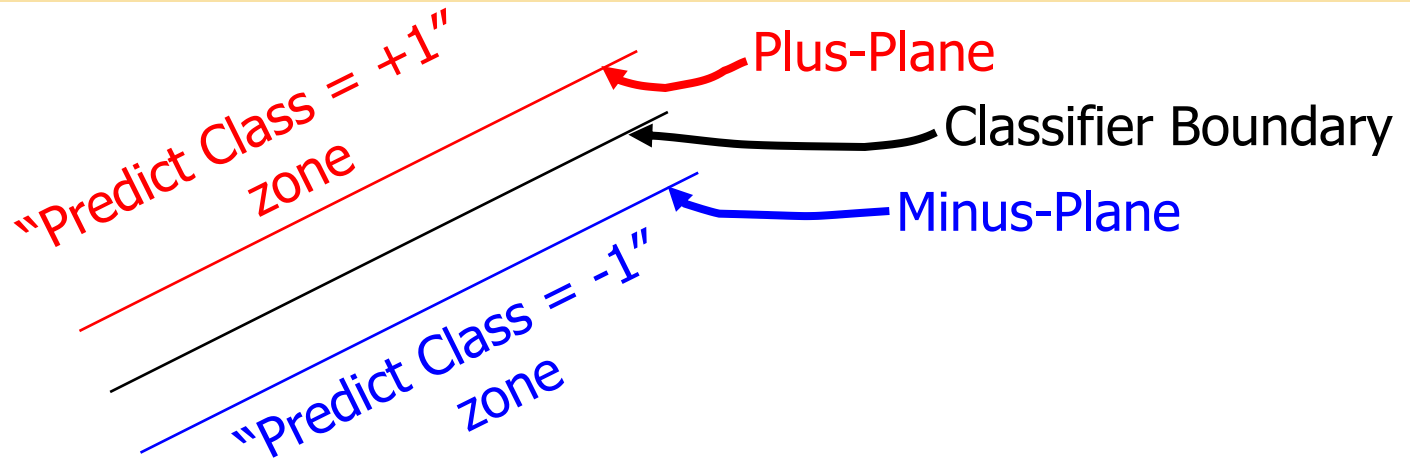
The maximum margin linear classifier is the linear classifier with the, maximum margin.

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

Why Maximum Margin?

- Intuitively this feels safest.
- 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.
- LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
- Empirically it works very very well.

Specifying a line and margin

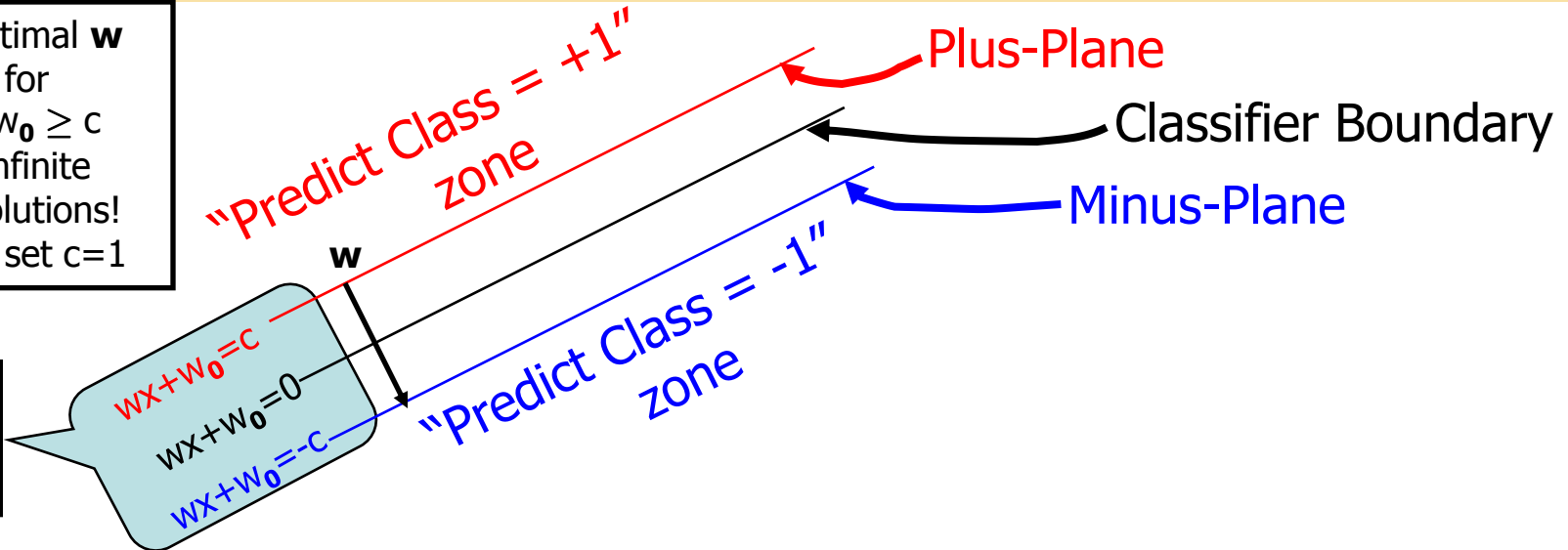


- How do we represent this mathematically?
- ...in m input dimensions?

Specifying a line and margin

Finding optimal \mathbf{w} and c for $\mathbf{w} \cdot \mathbf{x} + w_0 \geq c$ has an infinite number solutions!
Solution: set $c=1$

There is a problem using c



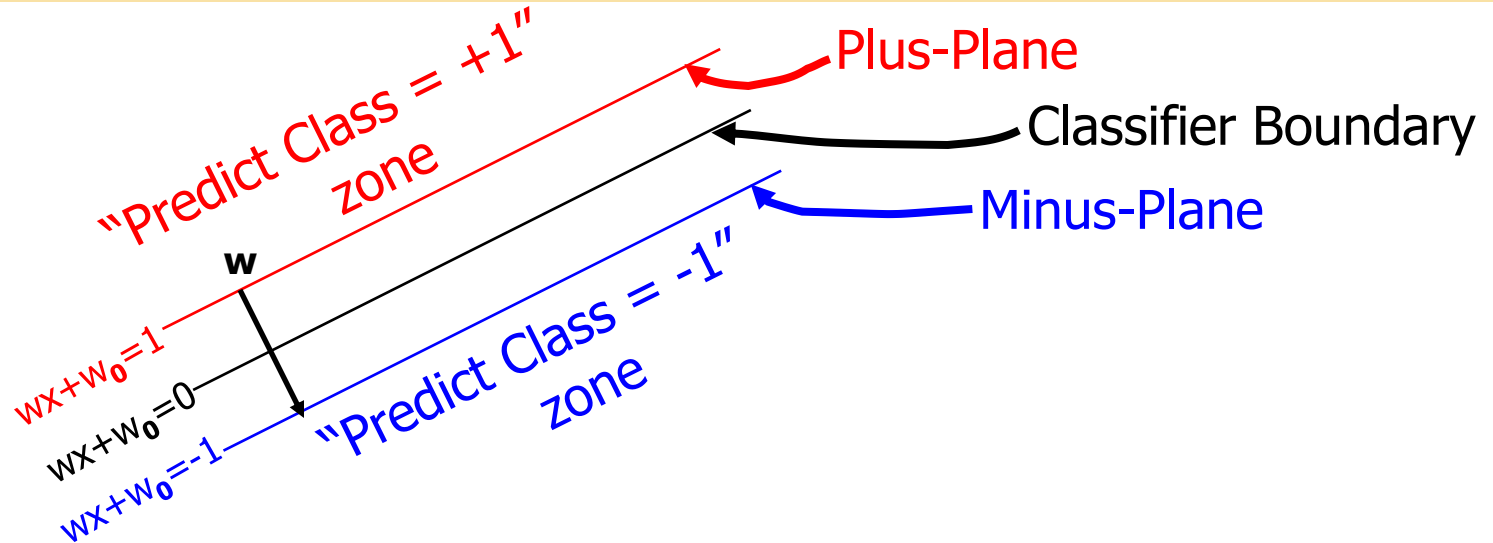
- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = +c \}$

- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = -c \}$

Classify $+1$ If $\mathbf{w} \cdot \mathbf{x} + w_0 \geq c$

as.. -1 If $\mathbf{w} \cdot \mathbf{x} + w_0 \leq -c$

Specifying a line and margin



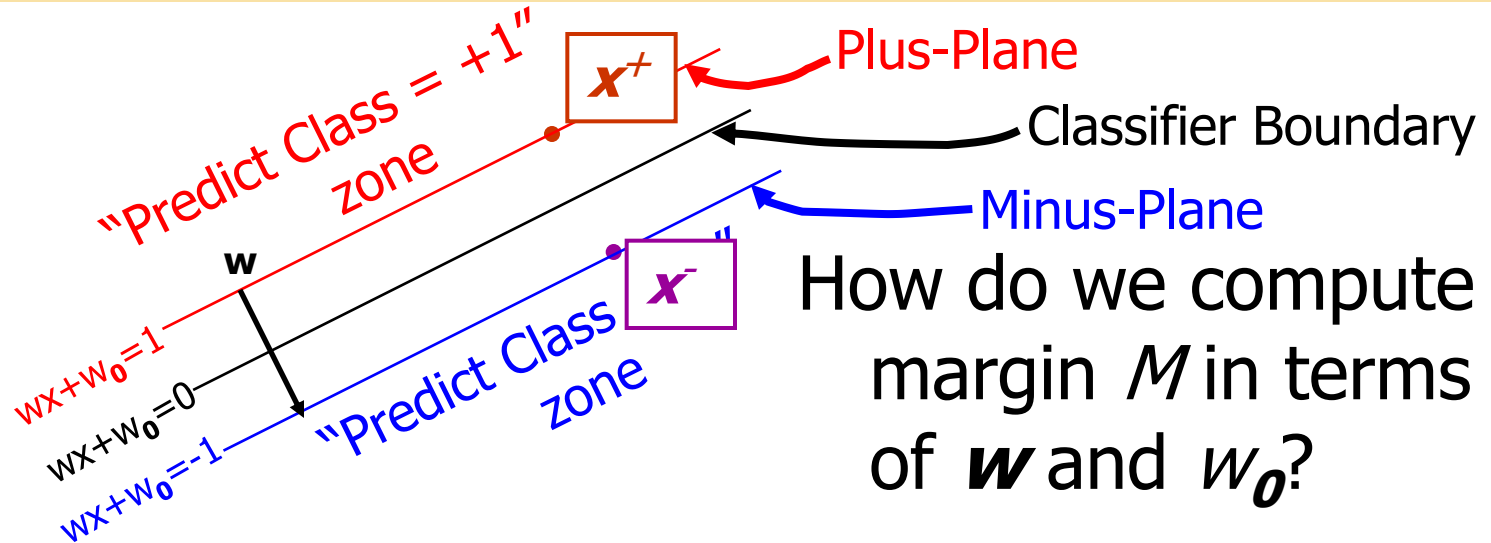
- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = -1 \}$

Classify **+1** If $\mathbf{w} \cdot \mathbf{x} + w_0 \geq 1$

as.. **-1** If $\mathbf{w} \cdot \mathbf{x} + w_0 \leq -1$

generally: $y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1$

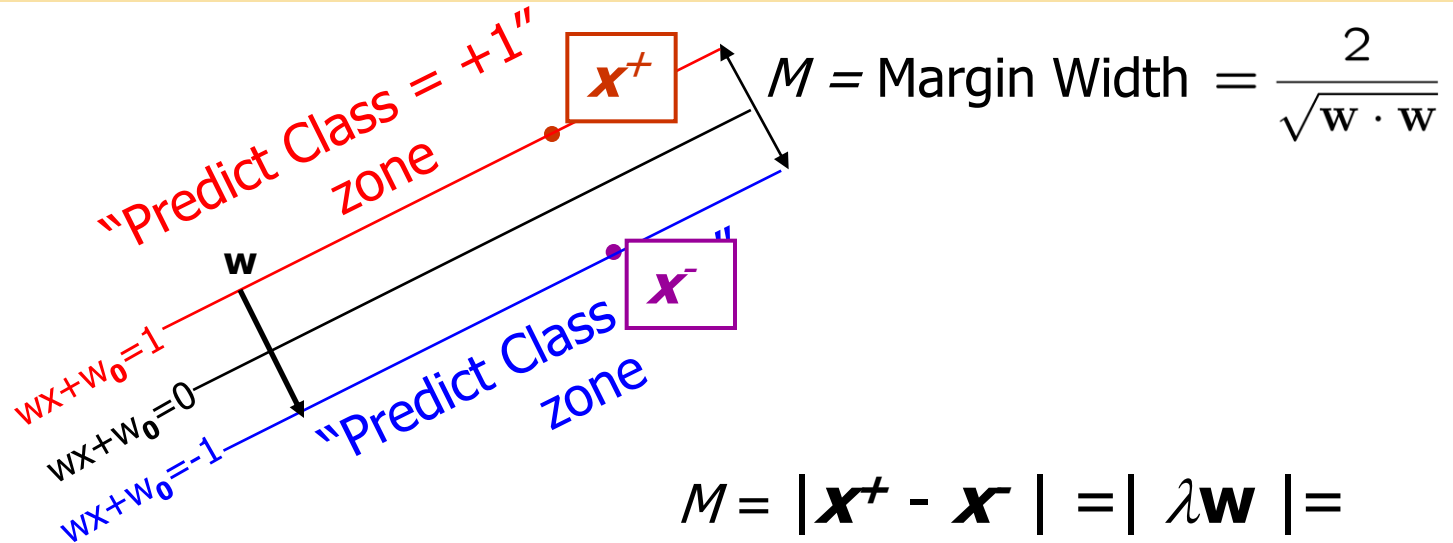
Computing the margin width



- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + w_0 = -1 \}$
- The vector \mathbf{w} is perpendicular to the Plus Plane
- Let \mathbf{x}^- be any point on the minus plane
- Let \mathbf{x}^+ be the closest plus-plane-point to \mathbf{x}^- .
- **Claim:** $\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$ for some value of λ .

Any location in \mathbb{R}^m : not necessarily a datapoint

Computing the margin width



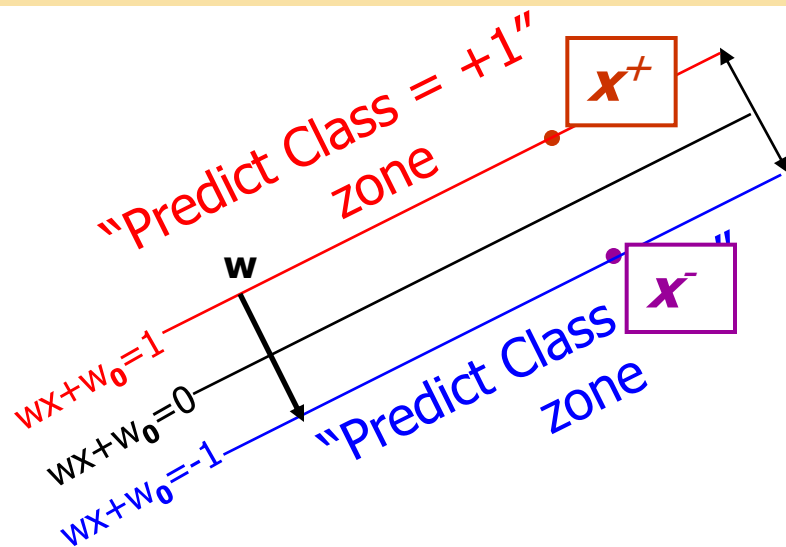
What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + w_0 = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + w_0 = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

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

$$\begin{aligned} M &= |\mathbf{x}^+ - \mathbf{x}^-| = |\lambda \mathbf{w}| = \\ &= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w} \cdot \mathbf{w}} \\ &= \frac{2\sqrt{\mathbf{w} \cdot \mathbf{w}}}{\mathbf{w} \cdot \mathbf{w}} = \frac{2}{\sqrt{\mathbf{w} \cdot \mathbf{w}}} \end{aligned}$$

Learning the Maximum Margin Classifier




- Given a guess of w and w_0 we can
 - Compute whether all data points are 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 w_0 '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 – General Formulation

Find $\arg \max_{\mathbf{w}} c + \mathbf{h}^T \mathbf{w} + \frac{\mathbf{w}^T \mathbf{Q} \mathbf{w}}{2}$  Quadratic criterion


Subject to

$$\begin{aligned} a_{11}w_1 + a_{12}w_2 + \dots + a_{1d}w_d &\leq b_1 \\ a_{21}w_1 + a_{22}w_2 + \dots + a_{2d}w_d &\leq b_2 \\ &\vdots \\ a_{N1}w_1 + a_{N2}w_2 + \dots + a_{Nd}w_d &\leq b_N \end{aligned}$$

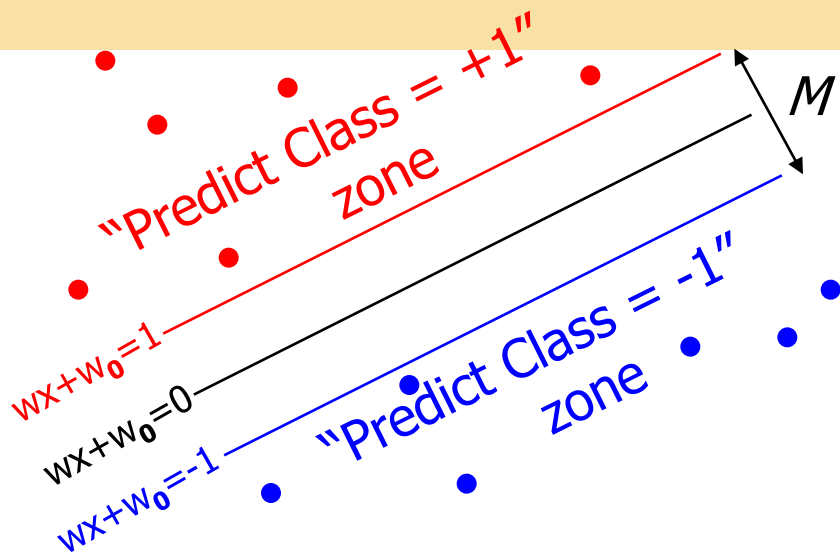
 n additional linear inequality constraints

And subject to

$$\begin{aligned} a_{(N+1)1}w_1 + a_{(N+1)2}w_2 + \dots + a_{(N+1)d}w_d &= b_{(N+1)} \\ a_{(N+2)1}w_1 + a_{(N+2)2}w_2 + \dots + a_{(N+2)d}w_d &= b_{(N+2)} \\ &\vdots \\ a_{(N+e)1}w_1 + a_{(N+e)2}w_2 + \dots + a_{(N+e)d}w_d &= b_{(N+e)} \end{aligned}$$

 e additional linear equality constraints

Learning the Maximum Margin Classifier



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

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

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

Assume N datapoints, each (\mathbf{x}_i, y_i) where $y_i = +/- 1$

How many constraints will we have? N

What should they be?

$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \geq 1 \text{ if } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \leq -1 \text{ if } y_i = -1$$

What should our quadratic optimization criterion be?

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

Or, more generally:

$$y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1$$

Formulating SVM Problem as a QP

- Formulation: We want to find \mathbf{w} such that:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1, \forall i$$

- We want to maximize the margin width, $M = \frac{2}{|\mathbf{w}|}$, so we want to minimize \mathbf{w}
- This can be written as a QP as:

$$\text{Minimize } (\mathbf{w} \cdot \mathbf{w})$$

$$\text{Subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1, \forall i$$

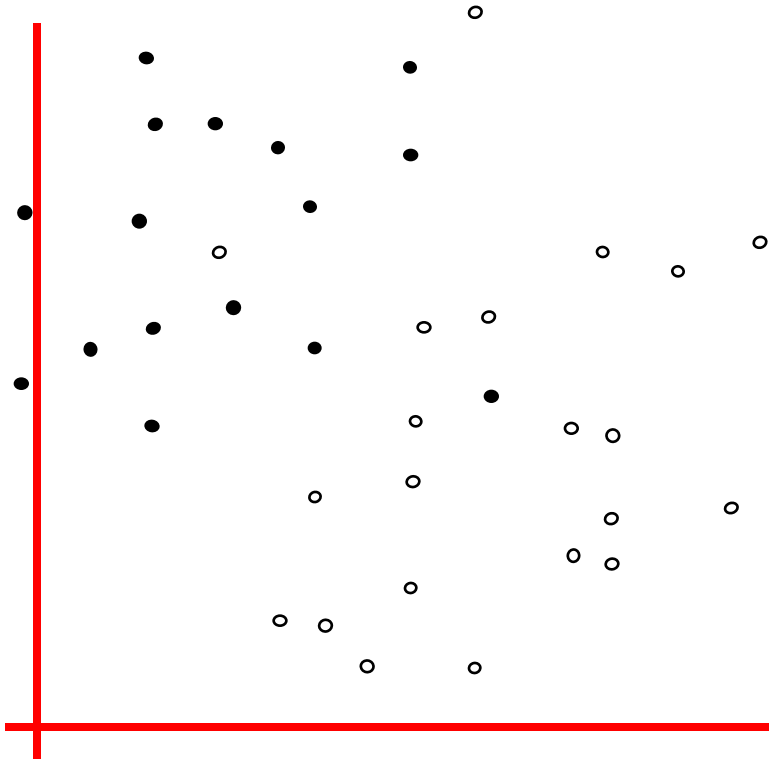
- This is a standard quadratic programming problem, whose complexity depends on d
 - It can be solved directly to find \mathbf{w} and w_0

What about noise?

This is going to be a problem!

What should we do?

- denotes +1
- denotes -1



Idea 1:

Find minimum $\|w\|^2$, while minimizing number of training set errors.

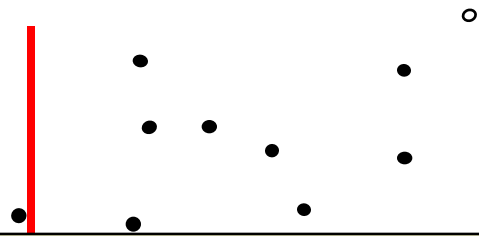
Problem: Two things to minimize makes for an ill-defined optimization

What about noise?

This is going to be a problem!

What should we do?

- denotes +1
- denotes -1



Idea 1.1:

Minimize

Tradeoff parameter

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

There's a serious practical problem which will make us reject this approach.

Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

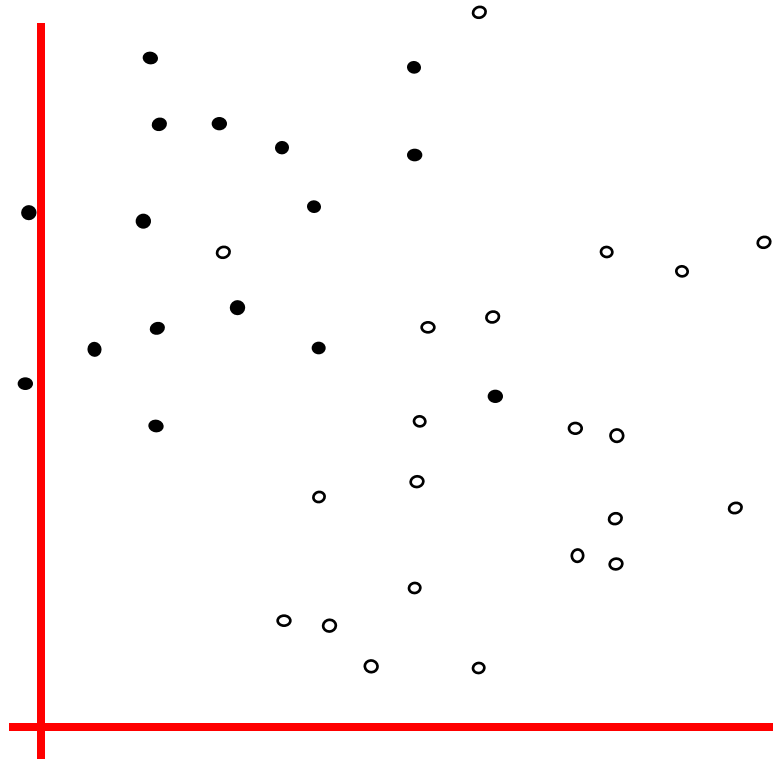
(Also, doesn't distinguish between disastrous errors and near misses)

What about noise?

This is going to be a problem!

What should we do?

- denotes +1
- denotes -1

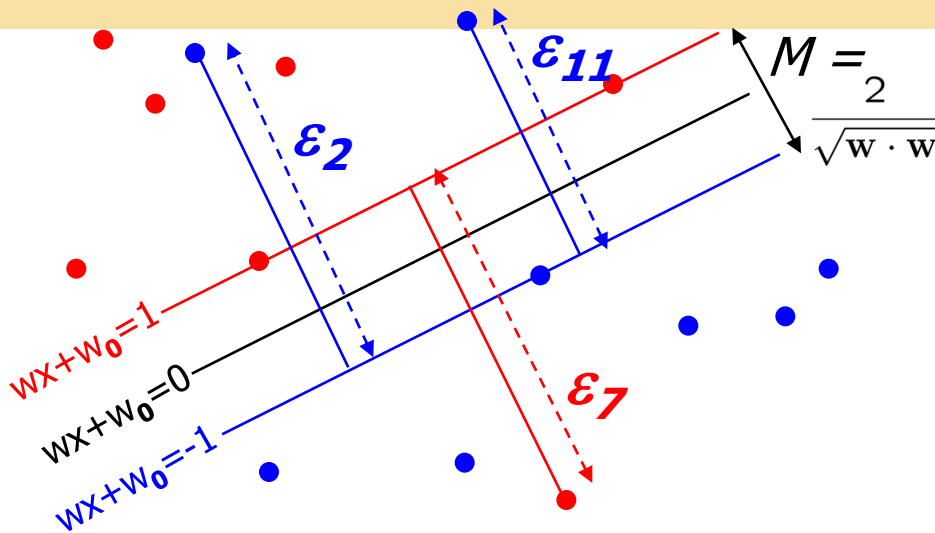


Idea 2.0:

Minimize

$W \cdot W + C$ (*distance of error points to their correct place*)

Learning Maximum Margin with Noise



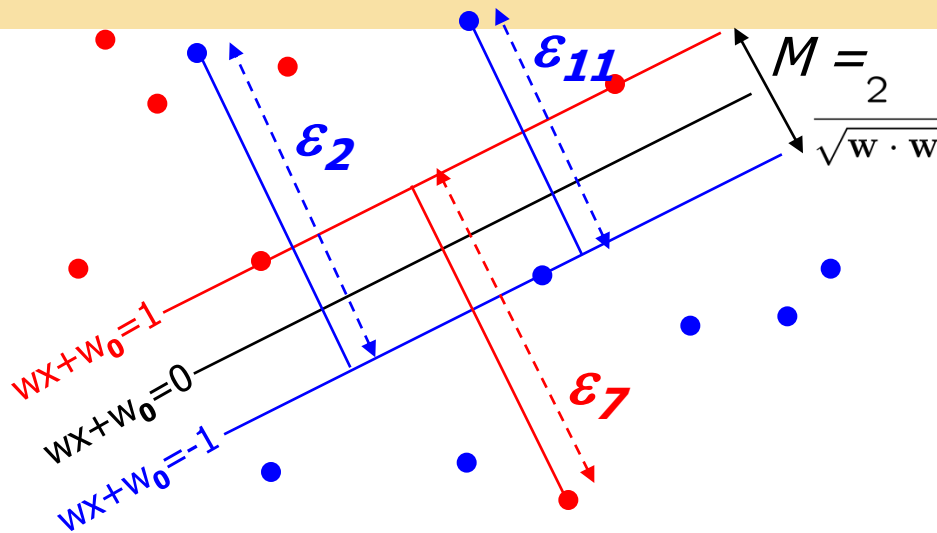
- Given guess of \mathbf{w} , b we can
- Compute sum of distances of points to their correct zones
 - Compute the margin width
- Assume N datapoints, each (\mathbf{x}_i, y_i) where $y_i = +/- 1$

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 \mathbf{w} , b we can

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

Assume N datapoints, each (\mathbf{x}_i, y_i) where $y_i = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_i \varepsilon_i$

How many constraints will we have? N

What should they be?

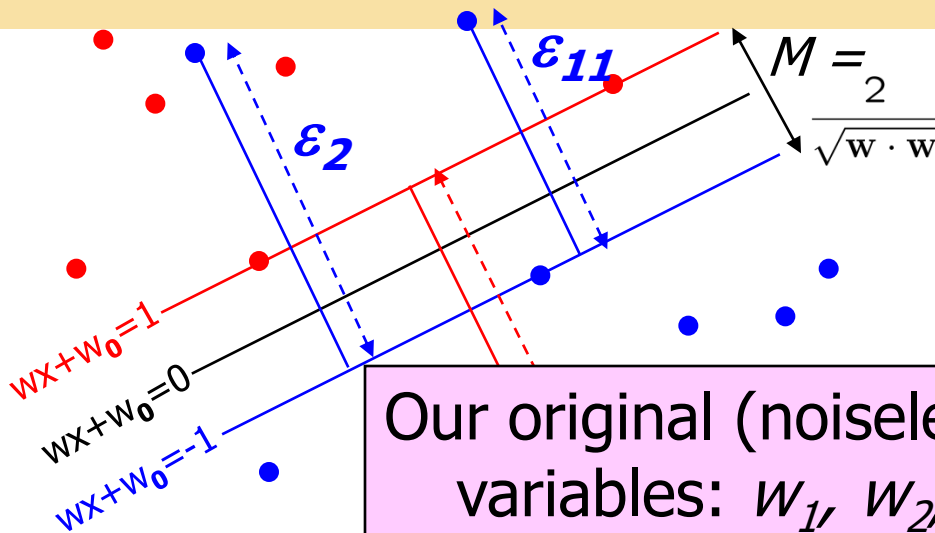
$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \geq 1 - \varepsilon_i \quad \text{if } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \leq -1 + \varepsilon_i \quad \text{if } y_i = -1$$

Or, more generally:

$$y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \varepsilon_i$$

Learning Maximum Margin with Noise



Given $d = \#$ input dimensions, b we can

- Control the distances of points to their correct zones

Our original (noiseless data) QP had $d+1$ variables: w_1, w_2, \dots, w_d and w_0 .

Our new (noisy data) QP has $d+1+N$ variables: $w_1, w_2, \dots, w_d, w_0, \epsilon_k, \epsilon_1, \dots, \epsilon_N$

What should the quadratic optimization criterion be?

Minimize $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_i \epsilon_i$

What should the constraints be?

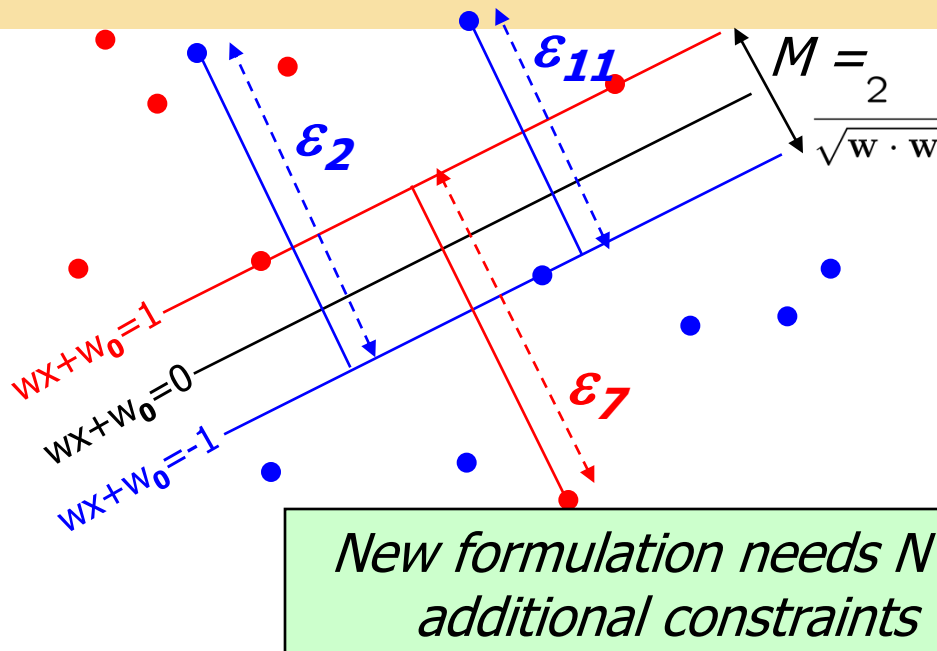
$\mathbf{w} \cdot \mathbf{x}_i + w_0 \geq 1 - \epsilon_i$ if $y_i = 1$

$\mathbf{w} \cdot \mathbf{x}_i + w_0 \leq -1 + \epsilon_i$ if $y_i = -1$

Or, more generally:

$y_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \epsilon_i$

Learning Maximum Margin with Noise



- Given guess of \mathbf{w} , b we can
- Compute sum of distances of points to their correct zones
 - Compute the margin width

Assume N datapoints, each (\mathbf{x}_i, y_i) where $y_i = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_i \varepsilon_i$

How many constraints will we have? $2N$

What should they be?

$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \geq 1 - \varepsilon_i \quad \text{if } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + w_0 \leq -1 + \varepsilon_i \quad \text{if } y_i = -1$$

$$\varepsilon_i \geq 0 \quad \forall i$$

Reformulating the QP problem

- Let's rewrite the QP as Lagrangian formulation.
- There are two reasons for this reformulation:
 - The constraints will be replaced by constraints on the Lagrange multipliers, which are easier to handle.
 - The training data will end up only appearing as dot products between vectors. This is crucial for the kernel trick to work.

Reformulating the QP problem (2)

- We reformulate into a Lagrangian formulation by introducing a positive Lagrange multiplier, α_i , $i=1, \dots, n$ –one for each of the inequality constraints.
- The rule is that for constraints of the form $c_i \geq 0$, the constraint equations are multiplied by *positive* Lagrange multipliers and subtracted from the object function, to form the Lagrangian.
- This gives the **primal** Lagrangian:

$$L_P = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + \sum_i \alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1\}$$

Karush-Kuhn-Tucker Conditions

- The solution which satisfies the KKT conditions is an **optimal solution**.
- For the primal problem, these are the KKT conditions:

$$\frac{\partial L_P}{\partial w_v} = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \dots, d$$

$$\frac{\partial L_P}{\partial w_0} = \sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 \geq 0 \quad i = 1, \dots, n$$

$$\alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1\} = 0 \quad i = 1, \dots, n$$

if $y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 > 0$
then α_i **must** be 0!

Therefore, only support vectors will have $\alpha_i > 0$.

Reformulating the QP problem (3)

$$L_P = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + \sum_i \alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1\}$$

- In the **primal**, we must minimize L_P with respect to \mathbf{w} and w_0 and require that the derivatives of L_P with respect to all the α_i vanish, all subject to the constraints $\alpha_i \geq 0$.
- We can equivalently solve the **dual** formulation: maximize L_P subject to the constraints that the gradient of L_P with respect to \mathbf{w} and w_0 vanish, and subject also to the constraints $\alpha_i \geq 0$.
 - This particular dual formulation is called the Wolfe dual.
 - It has the property that the maximum of L_P subject to the dual constraints occurs at the same values of \mathbf{w} , w_0 and α , as the minimum of L_P subject to the primal constraints.

Wolfe Dual Optimization Problem

$$L_P = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + \sum_i \alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1\}$$

- We want to *maximize* L_P with respect to α_i
 - Subject to the constraints that the gradient of L_P with respect to \mathbf{w} and w_0 are 0 and also that $\alpha_i \geq 0$:

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_P}{\partial w_0} = \sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0$$

- Plugging these equalities back into the primal gives us the dual formulation, L_D , which now only contains α and no \mathbf{w} :

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

Wolfe Dual Optimization Problem

- We want to solve the primal problem L_D
 - Since L_D is convex and L_D^* is concave, we can use the Wolfe dual optimization problem

Remember the Wolfe dual has the property that it will find the same optimal α , \mathbf{w} and w_0 as the primal...

...and that the primal KKT conditions enforced that only support vectors will have $\alpha_i > 0$

- Plugging these into the primal gives us the dual formulation L_D , which only contains α and no \mathbf{w} :

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

Original QP Formulation (with noise)

- Formulation: We want to find \mathbf{w} such that:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i, \forall i$$

- We want to maximize the margin width, $M = \frac{2}{|\mathbf{w}|}$, so we want to minimize \mathbf{w}
- This can be written as a QP as:

$$\text{Minimize } \frac{1}{2}(\mathbf{w} \cdot \mathbf{w})$$

$$\text{Subject to } \begin{aligned} y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) &\geq 1 - \xi_i, \forall i \\ \xi_i &\geq 0 \end{aligned}$$

- This is a standard quadratic programming problem, whose complexity depends on d
 - It can be solved directly to find \mathbf{w} and w_0

Langrangian formulation

- The **primal** Lagrangian:

$$L_P = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_i \xi_i - \sum_i \alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i\} - \sum_i \mu_i \xi_i$$

where μ_i are Lagrange multipliers introduced to enforce positivity on the ξ_i , and C is a error penalty term.

- A larger C corresponds to assigning a higher penalty to errors.

Karush-Kuhn-Tucker Conditions

- The solution which satisfies the KKT conditions is an **optimal solution**.
- For the noisy primal problem, these are the KKT conditions:

$$\frac{\partial L_P}{\partial w_v} = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \dots, d$$

$$\frac{\partial L_P}{\partial w_0} = \sum_i \alpha_i y_i = 0$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

$$\xi_i \geq 0 \quad \forall i$$

$$\mu_i \geq 0 \quad \forall i$$

$$\mu_i \xi_i = 0 \quad \forall i$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i \geq 0 \quad i = 1, \dots, n$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i) = 0 \quad i = 1, \dots, n$$

Karush-Kuhn-Tucker Conditions

- The solution which satisfies the KKT conditions is an **optimal solution**.
- For the noisy primal problem, these are the KKT conditions:

$$\frac{\partial L_P}{\partial w_v} = w_v - \sum_i \alpha_i y_i x_{iv} = 0 \quad v = 1, \dots, n$$

$$\frac{\partial L_P}{\partial w_0} = \sum_i \alpha_i y_i = 0$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

$$\xi_i \geq 0 \quad \forall i$$

$$\mu_i \geq 0 \quad \forall i$$

$$\mu_i \xi_i = 0 \quad \forall i$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i \geq 0 \quad i = 1, \dots, n$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i) = 0 \quad i = 1, \dots, n$$

if $\mu_i = 0$
then $\alpha_i = C$

if $y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i > 0$
then α_i **must** be 0!

Therefore, only support vectors (and errors) will have $\alpha_i > 0$.

if $\xi_i > 0$
then $\mu_i = 0$

So...

...if there is an error then $\alpha_i = C$
...only support vectors have $\alpha_i < C$

Wolfe Dual Optimization Problem

$$L_P = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w}) + C \sum_i \xi_i - \sum_i \alpha_i \{y_i(\mathbf{x}_i \cdot \mathbf{w} + w_0) - 1 + \xi_i\} - \sum_i \mu_i \xi_i$$

- We want to *maximize* L_P with respect to α_i
 - Subject to the constraints that the gradient of L_P with respect to \mathbf{w} and w_0 are 0 and also that $C \geq \alpha_i \geq 0$:

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_P}{\partial w_0} = \sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0$$

Only difference from no-noise scenario!

- The dual formulation, L_D , again only contains α and no \mathbf{w} (the ξ_i do not appear in the Wolfe Dual):

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

New QP Formulation

$$\text{Maximize } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j$$

Datapoints with $C > \alpha_i > 0$ are *guaranteed* to be the support vectors (due to KKT conditions.)

Subject to these constraints:

$$0 \leq \alpha_i \leq C \quad \forall i$$

.. this sum needs to be over all $\alpha_i > 0$ (i.e., support vectors and errors).

Then define:

$$D_\alpha = \{x_i \in D \mid \alpha_i > 0\}$$

$$w = \sum_{i \in D_\alpha} \alpha_i y_i x_i$$

The
 $w(x)$

We only consider support vectors, since we don't keep track of ξ_i values.

$$D_{\alpha'} = \{x_i \in D \mid 0 < \alpha_i < C\}$$

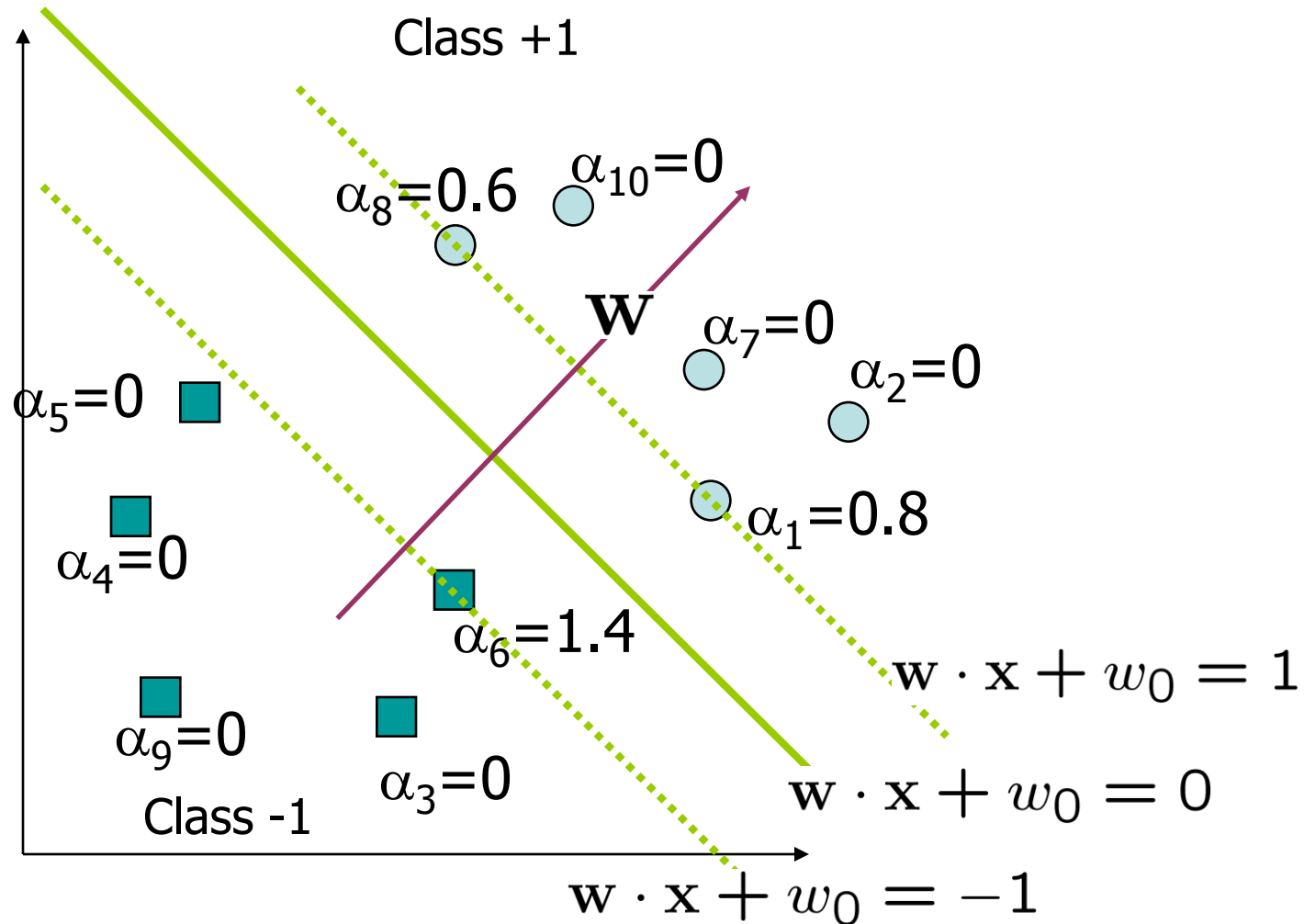
$$w_0 = \frac{1}{|D_{\alpha'}|} \sum_{i \in D_{\alpha'}} y_i - x_i * w$$

Average over all support vectors for numerical stability.

Characteristics of the Solution

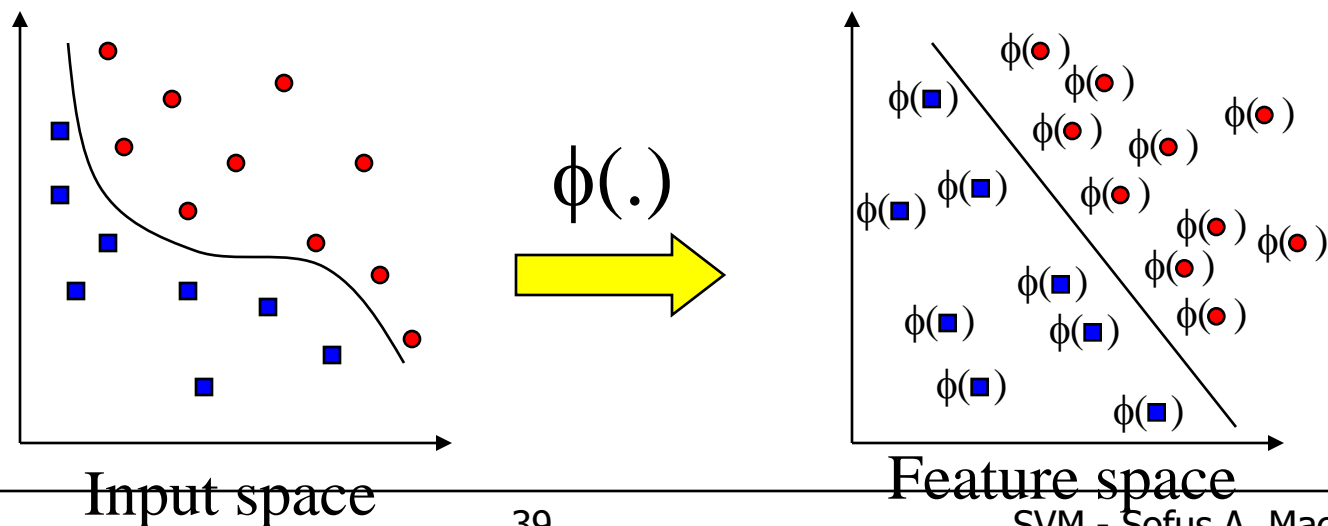
- Many of the α_i are zero
 - \mathbf{w} is a linear combination of a small number of examples
 - Sparse representation
- \mathbf{x}_i with non-zero α_i are called support vectors (SV)
 - The decision boundary is determined only by the SV
- For testing with a new data \mathbf{z}
 - Compute $\mathbf{w} \cdot \mathbf{z} + w_0 = \sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j} \cdot \mathbf{z}) + w_0$ and classify \mathbf{z} as +1 if the sum is positive, and -1 otherwise

A Geometric Interpretation



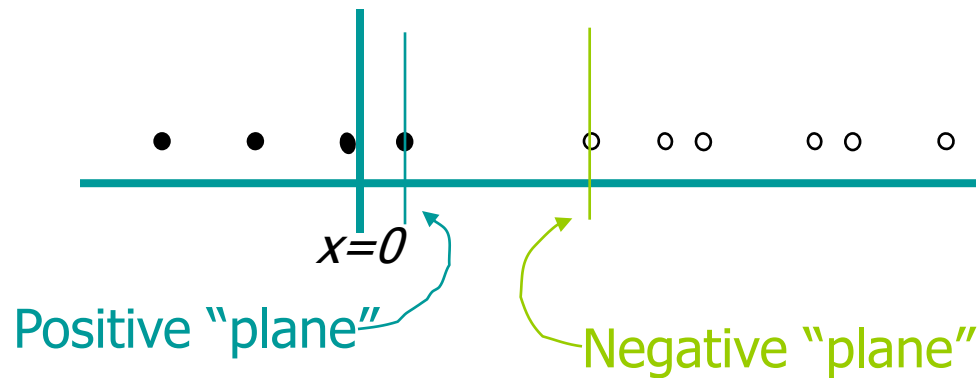
Extension to Non-linear

- Possible problem of the transformation
 - High computation burden and hard to get a good estimate
- SVM solves these issues simultaneously
 - Kernel tricks for efficient computation
 - Minimize $\|\mathbf{w}\|^2$ can lead to a “good” classifier



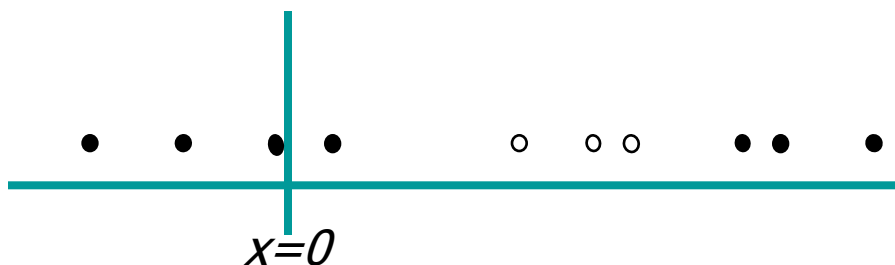
Suppose we're in 1-dimension

What would
SVMs do with
this data?

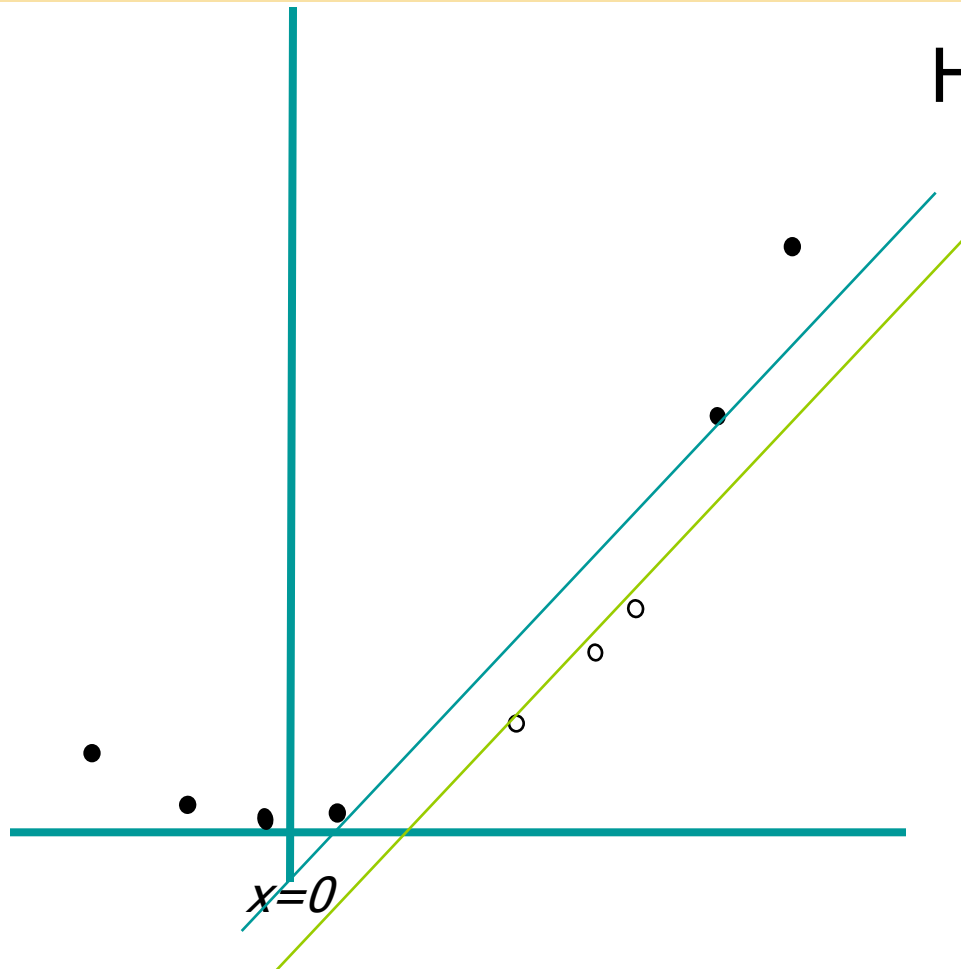


Harder 1-dimensional dataset

What can be
done about
this?



Harder 1-dimensional dataset



How about using
non-linear basis
functions?

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

Common SVM basis functions

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

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

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

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

This is sensible.

Is that the end of the story?

No...there's one more trick!

Quadratic Basis Functions

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_d \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_d^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_d \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_d \\ \vdots \\ \sqrt{2}x_{d-1}x_d \end{pmatrix}$$

Constant Term

Linear Terms

Pure Quadratic Terms

Quadratic Cross-Terms

Number of terms (assuming m input dimensions) = $\binom{d+2}{2}$
= $(d+2)(d+1)/2$
= (as near as makes no difference) $d^2/2$

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

$$\text{Maximize } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j Q_{ij} \quad \text{where } Q_{ij} = y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$$

Subject to these constraints:

$$0 \leq \alpha_i \leq C \quad \forall i$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Then define:

$$\mathbf{w} = \sum_{i \text{ s.t. } \alpha_i > 0} \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$w_0 = y_I (1 - \varepsilon_I) - \mathbf{x}_I \cdot \mathbf{w}$$

$$\text{where } I = \arg \max_i \alpha_i$$

Then classify with:

$$f(\mathbf{x}, \mathbf{w}, w_0) = \text{sign}(\mathbf{w} \cdot \phi(\mathbf{x}) - w_0)$$

QP with basis functions

$$\text{Maximize } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j Q_{ij} \text{ where } Q_{ij} = y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$$

Subject to these constraints:

$$0 \leq \alpha_i \leq C$$

Then define:

$$\mathbf{w} = \sum_{i \text{ s.t. } \alpha_i > 0} \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$w_0 = y_I (1 - \varepsilon_I) - \mathbf{x}_I \cdot \mathbf{w}$$

$$\text{where } I = \arg \max_i \alpha_i$$

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

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

The whole thing costs $N^2 d^2 / 4$.
Yeeks!

...or does it?

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

$$\begin{pmatrix} 1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ \vdots \\ \sqrt{2}a_d \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_d^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \vdots \\ \sqrt{2}a_1a_d \\ \sqrt{2}a_2a_3 \\ \vdots \\ \sqrt{2}a_1a_d \\ \vdots \\ \sqrt{2}a_{d-1}a_d \end{pmatrix} \bullet \begin{pmatrix} 1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ \vdots \\ \sqrt{2}b_d \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_d^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \vdots \\ \sqrt{2}b_1b_d \\ \sqrt{2}b_2b_3 \\ \vdots \\ \sqrt{2}b_1b_d \\ \vdots \\ \sqrt{2}b_{d-1}b_d \end{pmatrix}$$

$$\begin{aligned} & 1 \\ & + \\ & \sum_{i=1}^d 2a_i b_i \\ & + \\ & \sum_{i=1}^d a_i^2 b_i^2 \\ & + \\ & \sum_{i=1}^d \sum_{j=i+1}^d 2a_i a_j b_i b_j \end{aligned}$$

Quadratic Dot Products

Let's look at another function of \mathbf{a} and \mathbf{b} :

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) = 1 + 2 \sum_{i=1}^d a_i b_i + \sum_{i=1}^d a_i^2 b_i^2 + \sum_{i=1}^d \sum_{j=i+1}^d 2a_i a_j b_i b_j$$

$$\begin{aligned} & (\mathbf{a} \cdot \mathbf{b} + 1)^2 \\ &= (\mathbf{a} \cdot \mathbf{b})^2 + 2\mathbf{a} \cdot \mathbf{b} + 1 \\ &= \left(\sum_{i=1}^d a_i b_i \right)^2 + 2 \sum_{i=1}^d a_i b_i + 1 \\ &= \sum_{i=1}^d \sum_{j=1}^d a_i b_i a_j b_j + 2 \sum_{i=1}^d a_i b_i + 1 \\ &= \sum_{i=1}^d (a_i b_i)^2 + 2 \sum_{i=1}^d \sum_{j=i+1}^d a_i b_i a_j b_j + 2 \sum_{i=1}^d a_i b_i + 1 \end{aligned}$$

They're the same!

And this is only $O(d)$ to compute!

QP with basis functions

$$\text{Maximize } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j Q_{ij} \text{ where } Q_{ij} = y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))$$

Subject to these constraints:

$$0 \leq \alpha_i \leq C$$

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

Each dot product now only requires d additions and multiplications

Then define:

$$\mathbf{w} = \sum_{i \text{ s.t. } \alpha_i > 0} \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$w_0 = y_I (1 - \varepsilon_I) - \mathbf{x}_I \cdot \mathbf{w}$$

$$\text{where } I = \arg \max_i \alpha_i$$

Then classify with:

$$f(\mathbf{x}, \mathbf{w}, w_0) = \text{sign}(\mathbf{w} \cdot \phi(\mathbf{x}) - w_0)$$

Higher Order Polynomials

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

Higher Dimensional Spaces

- Theorem: For any data set, there exists a mapping Φ to a higher-dimensional space such that the data is linearly separable
- $\Phi(\mathbf{X}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_D(\mathbf{x}))$
- Example: Map to quadratic space
 - $\mathbf{x} = (x_1, x_2)$ (just two features)
 - $\Phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2, \sqrt{2} x_1, \sqrt{2} x_2, 1)$
 - compute linear separator in this space

SVM Standard Kernel Functions

- Dot product
$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$
- Polynomial of degree d
$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$$
- Gaussian with scale σ
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / \sigma^2)$$
- Polynomials often give strange boundaries. Gaussians generally work well.

More 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:

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

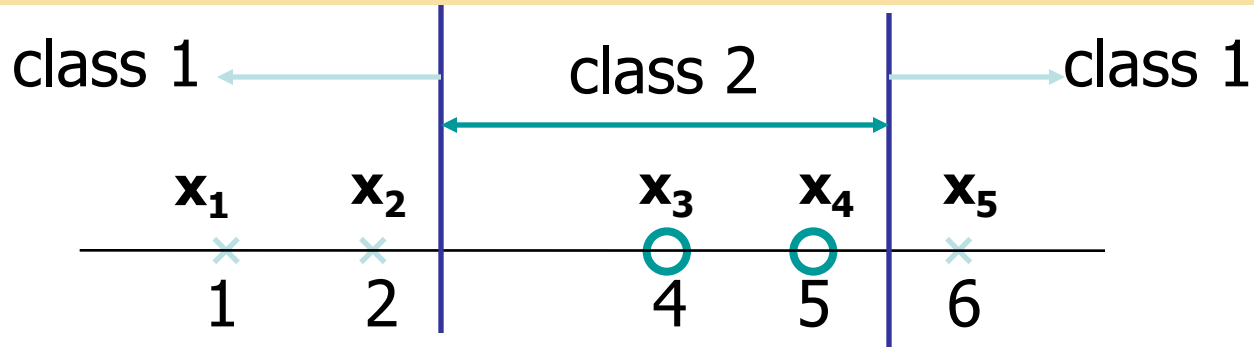
- Neural-net-style Kernel Function:

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

What are the kernel functions?

- Consider the linear kernel that just computes the dot product of two vectors, $K(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z} = |\mathbf{x}| \cdot |\mathbf{z}| \cdot \cos u$, where u is the angle between \mathbf{x} and \mathbf{z}
- Hence, if the two vectors are orthogonal, their dot products is 0
- If they lie in the same direction, their dot product is maximal
- By analogy, people tend to think of kernels as similarity functions between input vectors
- E.g., the Gaussian kernel is 1 if the two vectors are identical and smaller otherwise (close to 0 if the difference of the two vectors is very large)

1D Example



- We use the polynomial kernel of degree 2
 - $K(x,y) = (x \cdot y + 1)^2$
 - C is set to 100
- We first find α_i ($i=1, \dots, 5$) by

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^5 \alpha_i y_i = 0$$

$$\text{max. } \sum_{i=1}^5 \alpha_i - \frac{1}{2} \sum_{i=1}^5 \sum_{j=1}^5 \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

1D Example

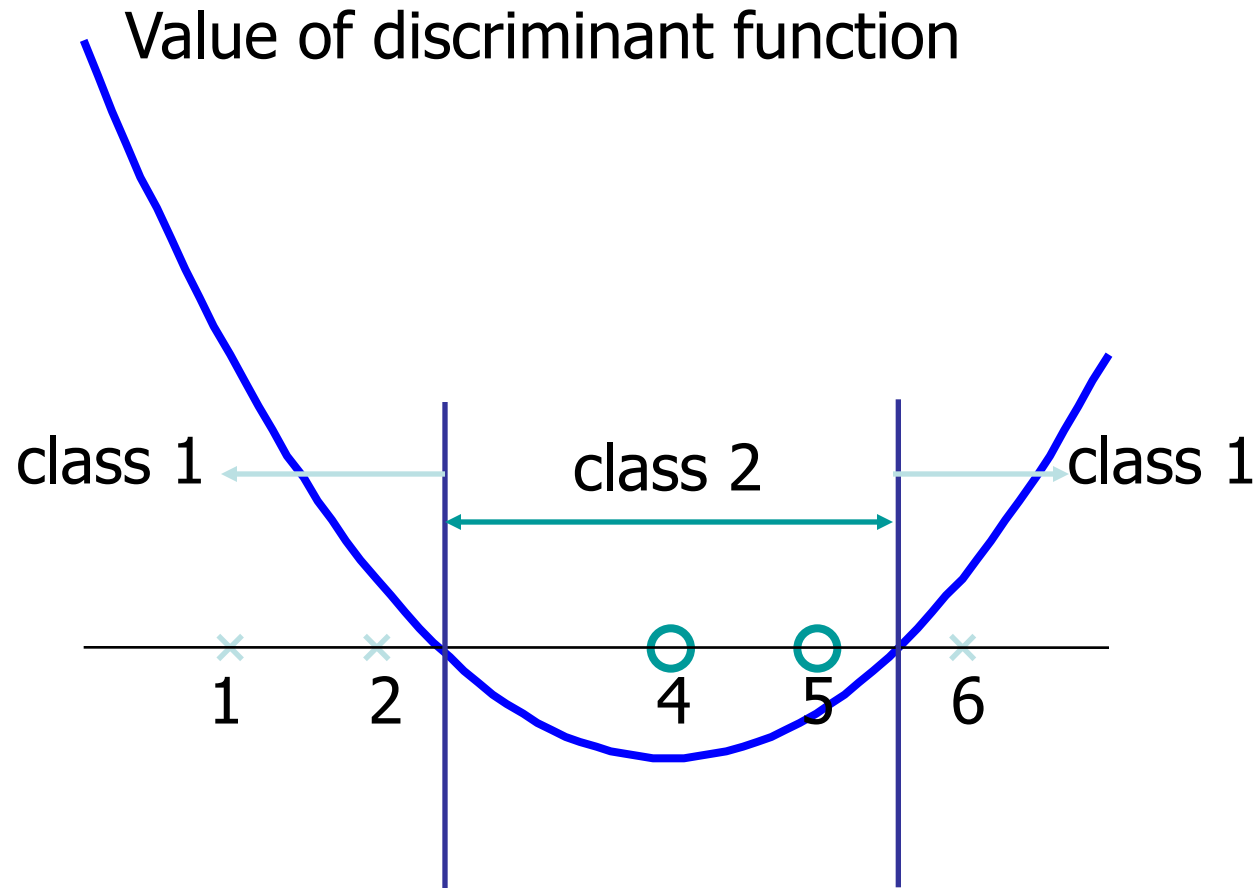
- By using a QP solver, we get
 - $\alpha_1=0, \alpha_2=2.5, \alpha_3=0, \alpha_4=7.333, \alpha_5=4.833$
 - Note that the constraints are indeed satisfied
 - The support vectors are $\{x_2=2, x_4=5, x_5=6\}$
- The discriminant function is

$$\begin{aligned} f(y) &= 2.5(1)(2x + 1)^2 + 7.333(-1)(5x + 1)^2 + 4.833(1)(6x + 1)^2 + b \\ &= 0.6667x^2 - 5.333x + b \end{aligned}$$

- w_0 is recovered by solving $f(2)=1$ or by $f(5)=-1$ or by $f(6)=1$, as x_2, x_4, x_5 lie on $y_i(w \cdot \phi(z) + w_0) = 1$ and all give $b=9$

 $f(y) = 0.6667x^2 - 5.333x + 9$

1D Example



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.

Higher Order Polynomials Revisited

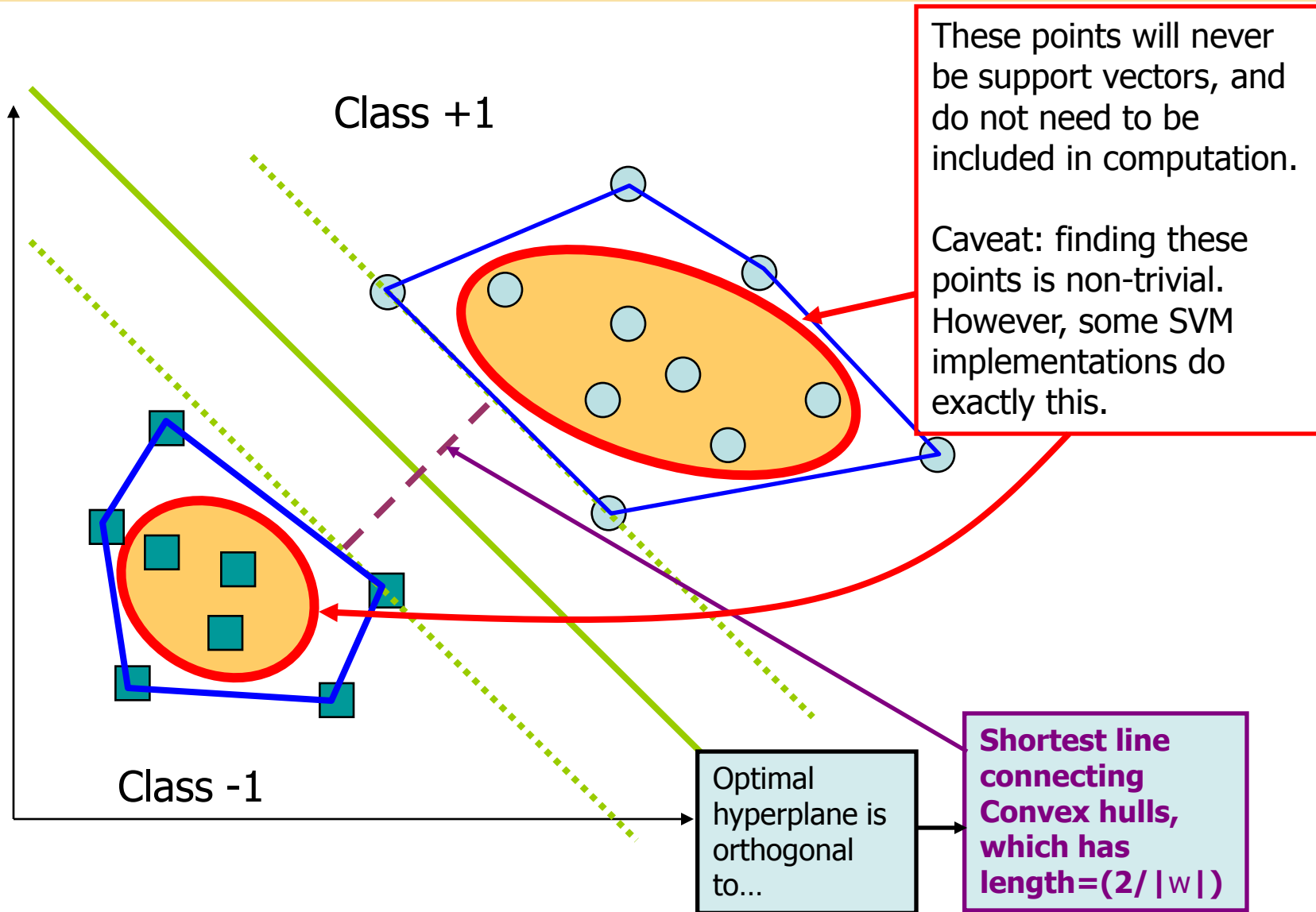
Poly-nomial	$\phi(\mathbf{x})$	Cost to build Q_{ij} matrix traditionally	Cost if 100 inputs	$\phi(\mathbf{a}) \cdot \phi(\mathbf{b})$	Cost to build Q_{ij} matrix with kernels	Cost if 100 inputs
Quadratic	All $d^2/2$ terms up to degree 2	$d^2 N^2 / 4$	2,500 N^2	$(\mathbf{a} \cdot \mathbf{b} + 1)^2$	$d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

Higher Order Polynomials Revisited

Poly-nomial	$\phi(\mathbf{x})$	Cost to build Q_{ij} matrix traditionally	Cost if 100 inputs	$\phi(\mathbf{a}) \cdot \phi(\mathbf{b})$	Cost to build Q_{ij} matrix with kernels	Cost if 100 inputs
Quadratic	All $d^2/2$ terms up to degree 2	$d^2 N^2 / 4$	2,500 N^2	$(\mathbf{a} \cdot \mathbf{b} + 1)^2$	$d N^2 / 2$	50 N^2
Cubic	All $d^3/6$ terms up to degree 3	$d^3 N^3 / 6$		$(\mathbf{a} \cdot \mathbf{b} + 1)^3$	$d N^2 / 2$	50 N^2
Quartic	All $d^4/24$ terms up to degree 4	$d^4 N^4 / 48$	1,960,000 N^2	$(\mathbf{a} \cdot \mathbf{b} + 1)^4$	$d N^2 / 2$	50 N^2

Costly in N .
Can anything be done?

SVM Margin geometry



Support Vector Machines Summary

- Advantages of SVMs
 - variable-sized hypothesis space
 - polynomial-time exact optimization rather than approximate methods
 - unlike decision trees and neural networks
 - Training is relatively easy
 - No local optimal, unlike in neural networks
 - It scales relatively well to high dimensional data
 - Tradeoff between classifier complexity and error can be controlled explicitly
 - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Disadvantages of SVMs
 - Must choose kernel and kernel parameters: Gaussian, σ
 - Very large problems are computationally intractable
 - quadratic in number of examples
 - problems with more than 20,000 examples are very difficult to solve exactly
 - Batch algorithm

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

Evaluation of SVMs

Criterion	Perc	Logistic	LDA	Trees	Nets	NNbr	SVM
Mixed data	no	no	no	yes	no	no	no
Missing values	no	no	yes	yes	no	somewhat	no
Outliers	no	yes	no	yes	yes	yes	yes
Monotone transformations	no	no	no	yes	somewhat	no	no
Scalability	yes	yes	yes	yes	yes	no	no
Irrelevant inputs	no	no	no	somewhat	no	no	yes*
Linear combinations	yes	yes	yes	no	yes	somewhat	yes
Interpretable	yes	yes	yes	yes	no	no	yes**
Accurate	yes	yes	yes	no	yes	no	yes

* = dot product kernel with absolute value penalty

** = dot product kernel