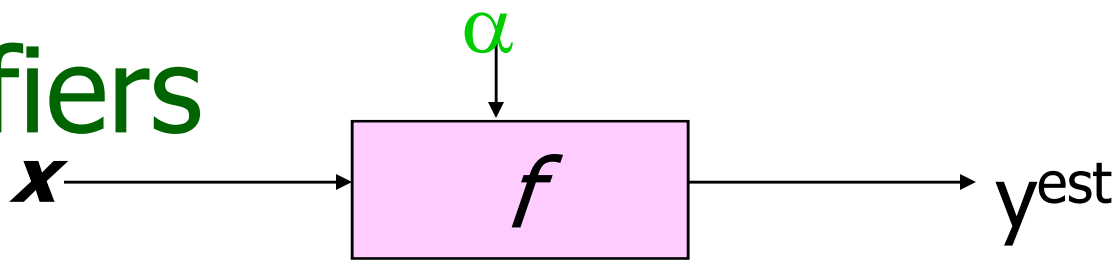
Support Vector Machines

Modified from the slides by Dr. Andrew W. Moore

http://www.cs.cmu.edu/~awm/tutorials

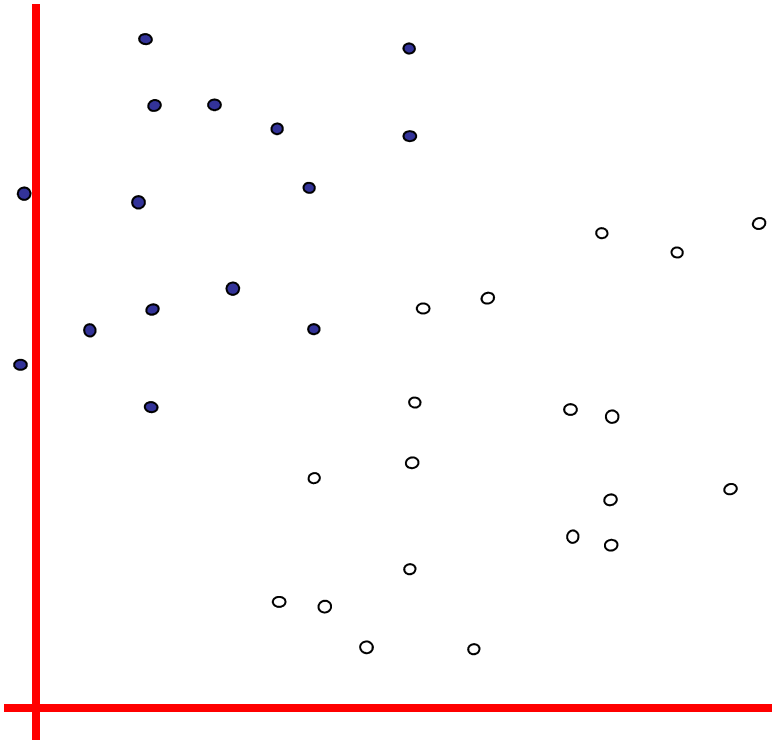
Nov 23rd, 2001

Linear Classifiers



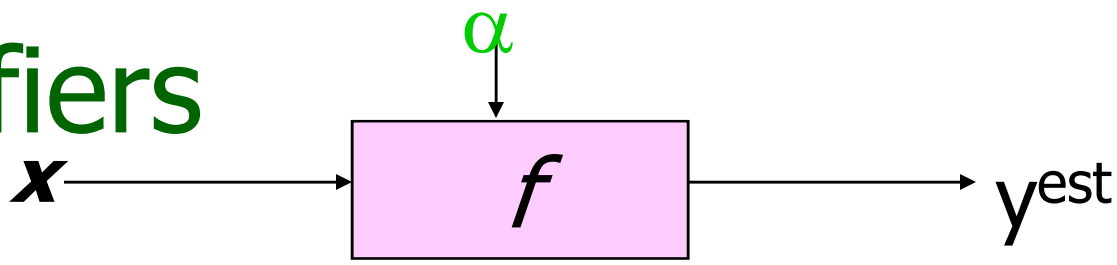
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



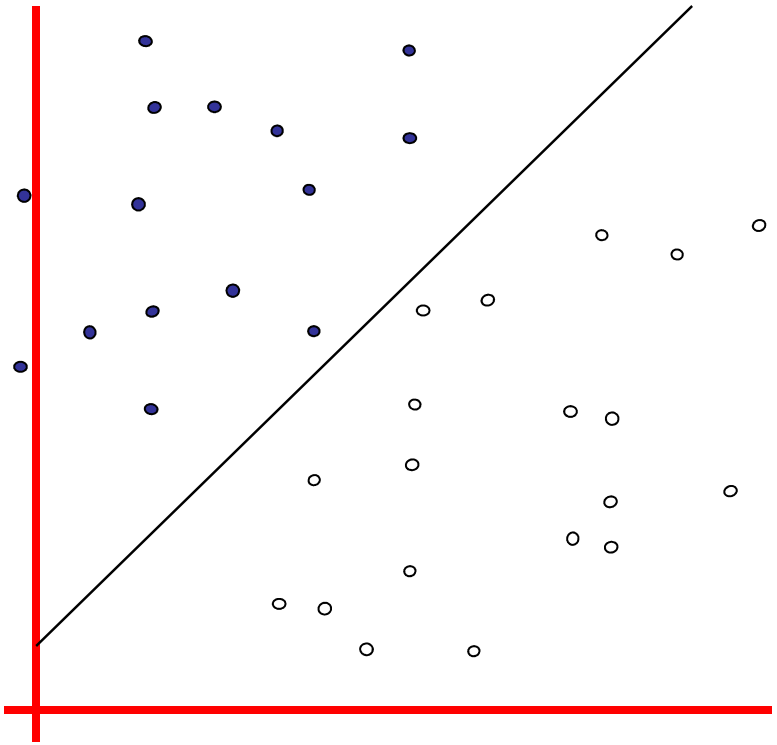
How would you classify this data?

Linear Classifiers



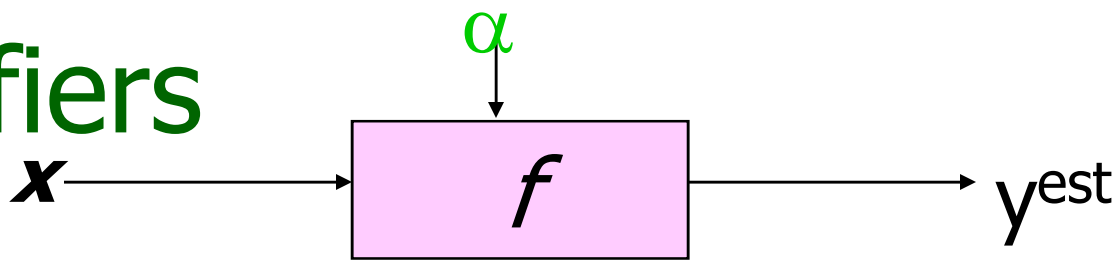
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



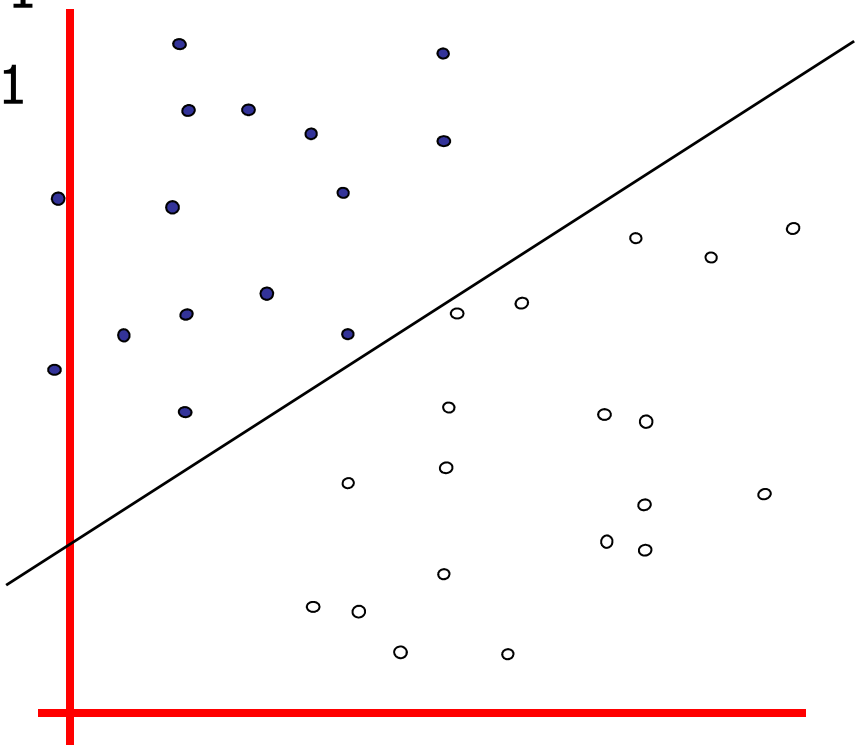
How would you classify this data?

Linear Classifiers



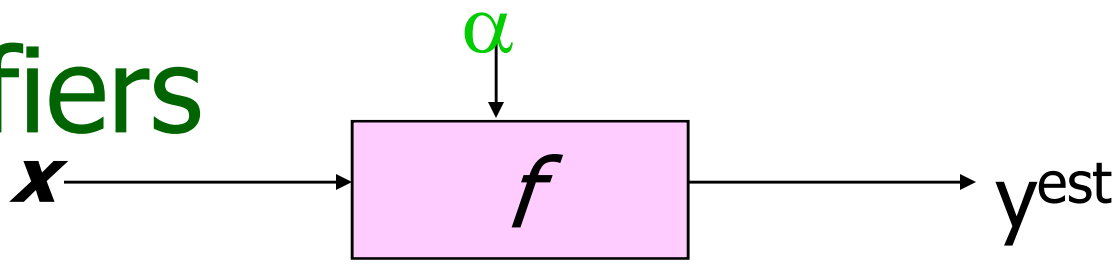
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



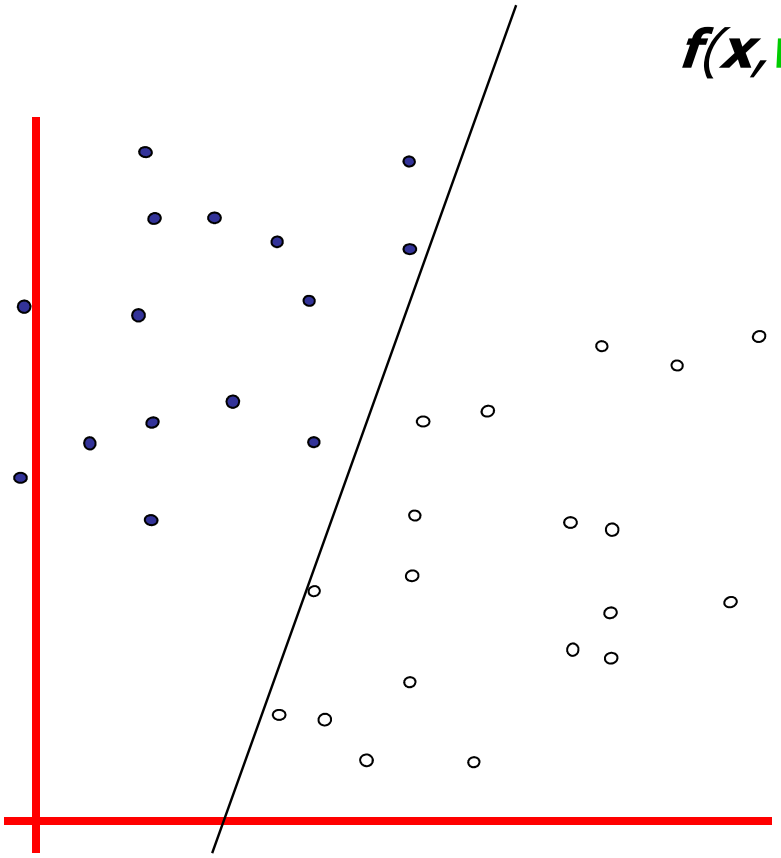
How would you classify this data?

Linear Classifiers



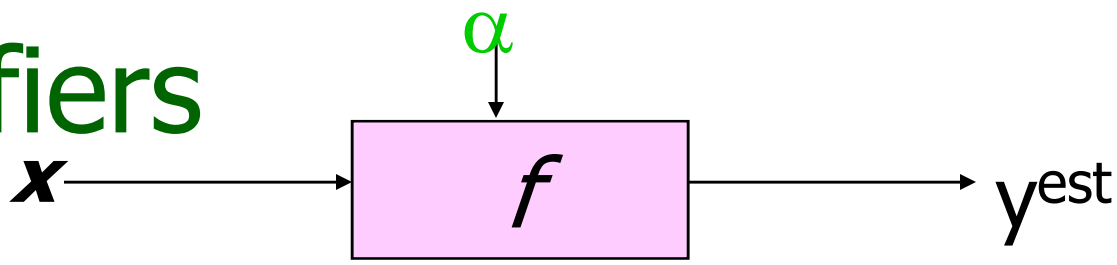
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



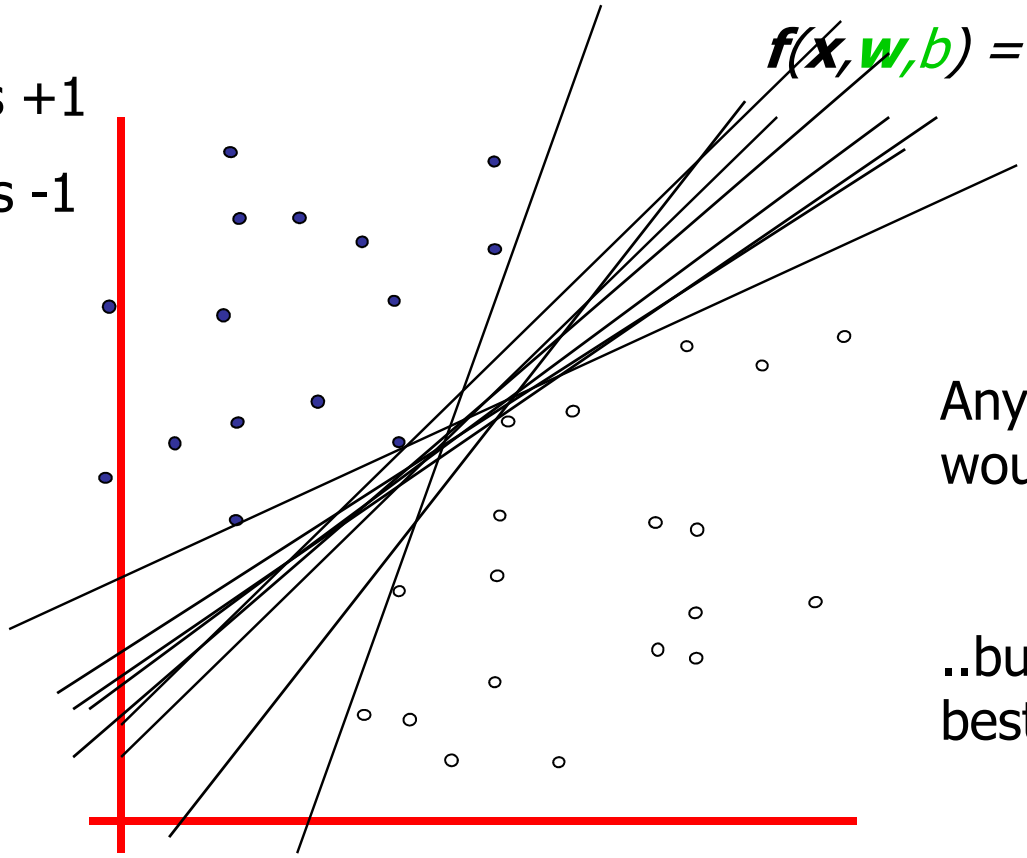
How would you classify this data?

Linear Classifiers



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

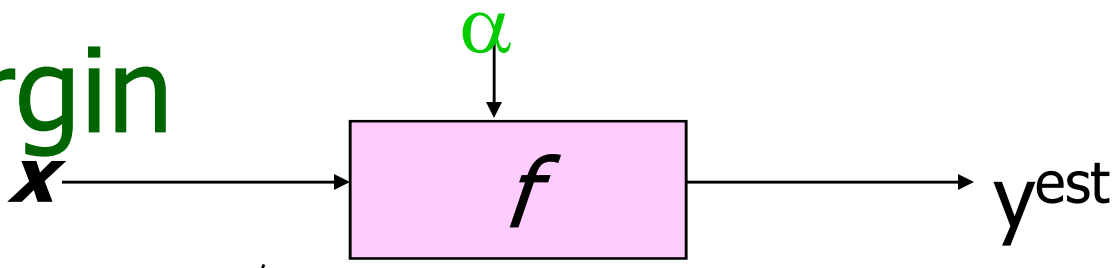
- denotes +1
- denotes -1



Any of these would be fine..

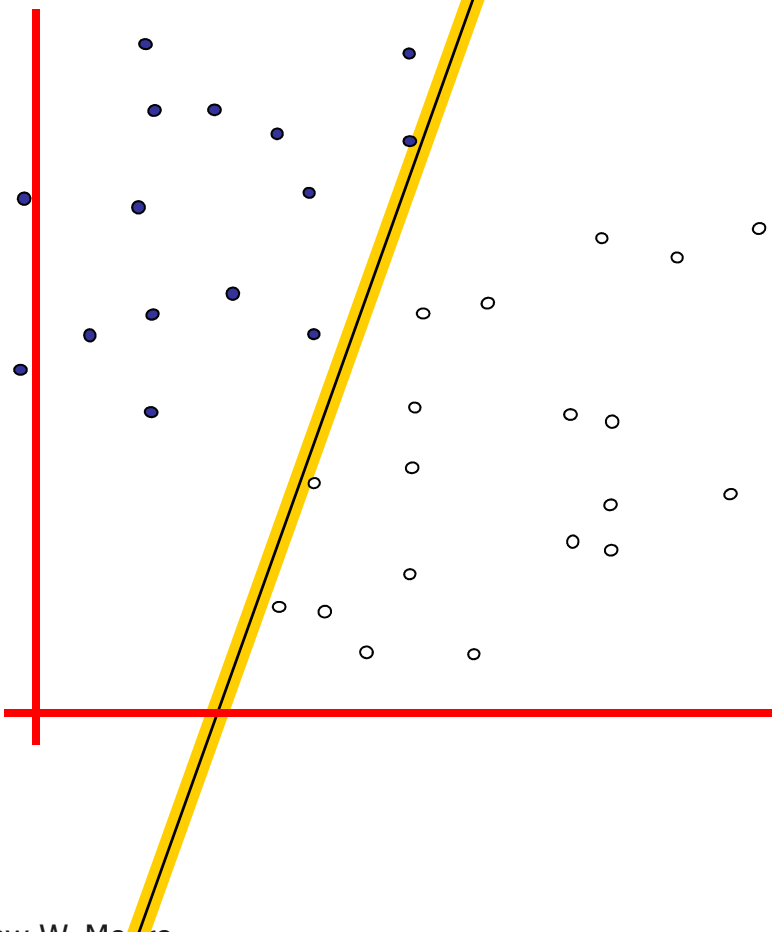
..but which is best?

Classifier Margin



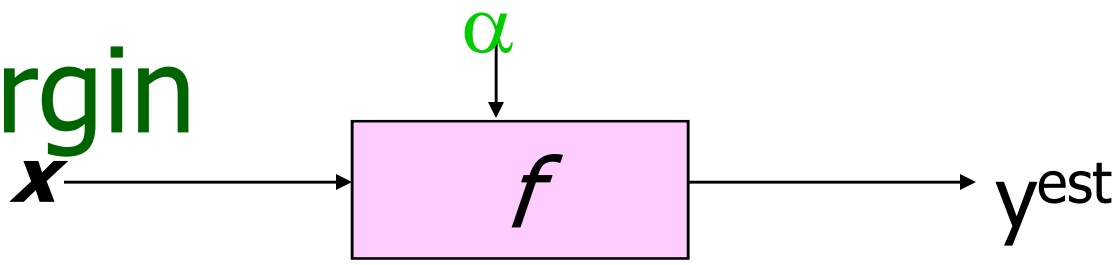
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



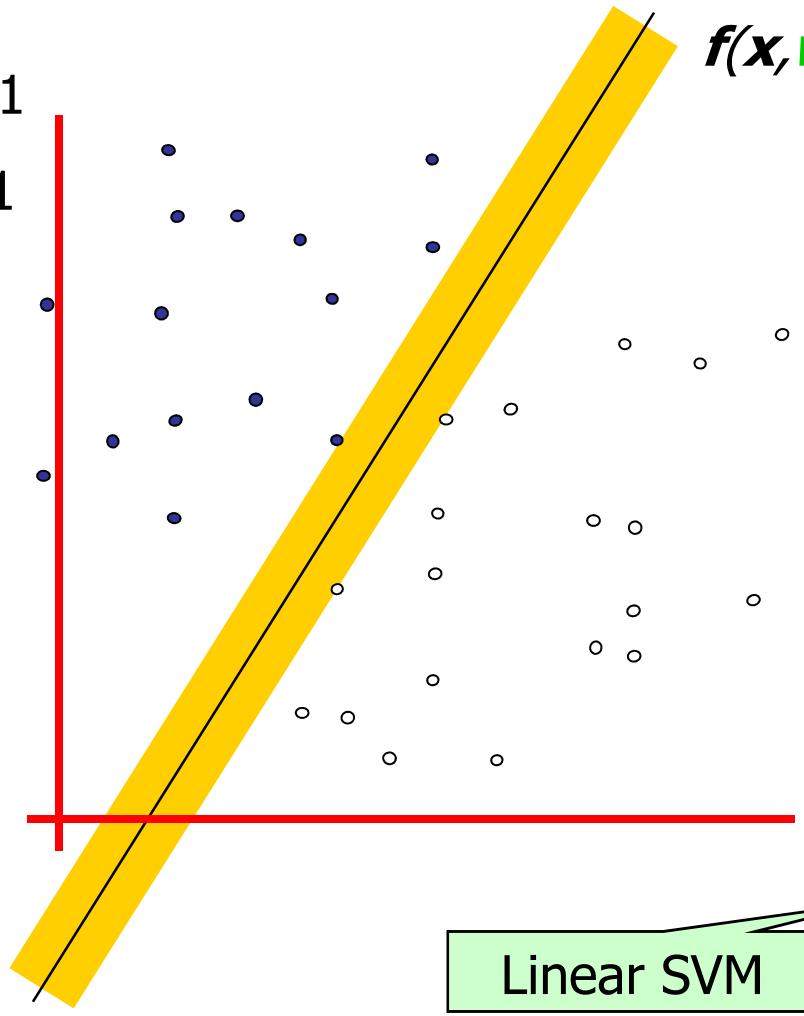
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



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

- denotes +1
- denotes -1

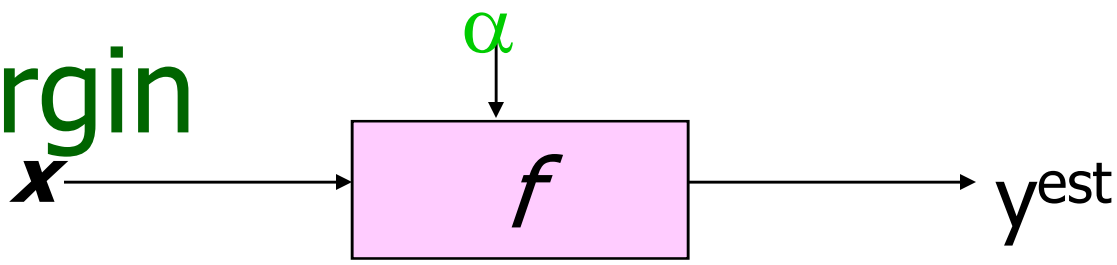


The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

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

Linear SVM

Maximum Margin



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

- denotes +1
- denotes -1

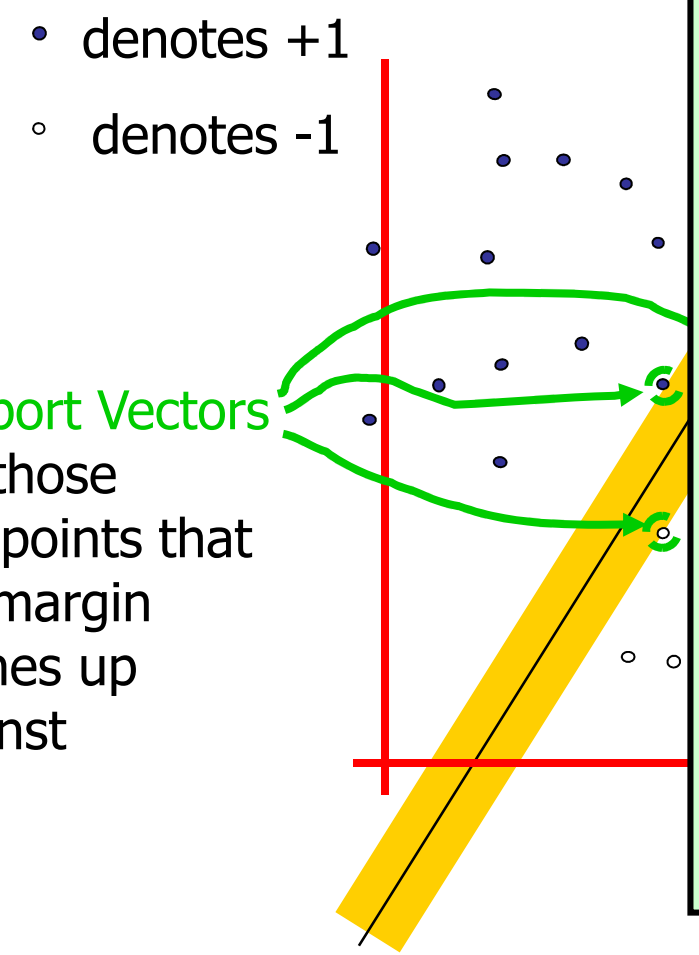
Support Vectors are those datapoints that the margin pushes up against

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

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

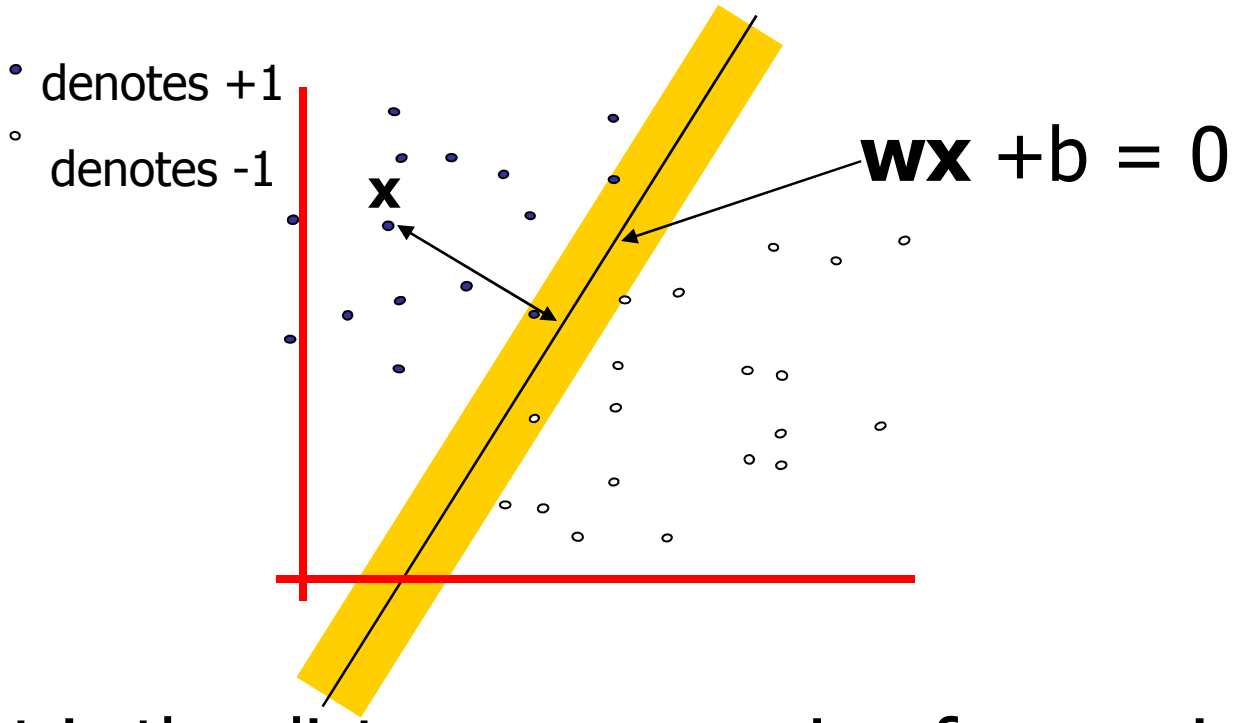
Linear SVM

Why Maximum Margin?



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.

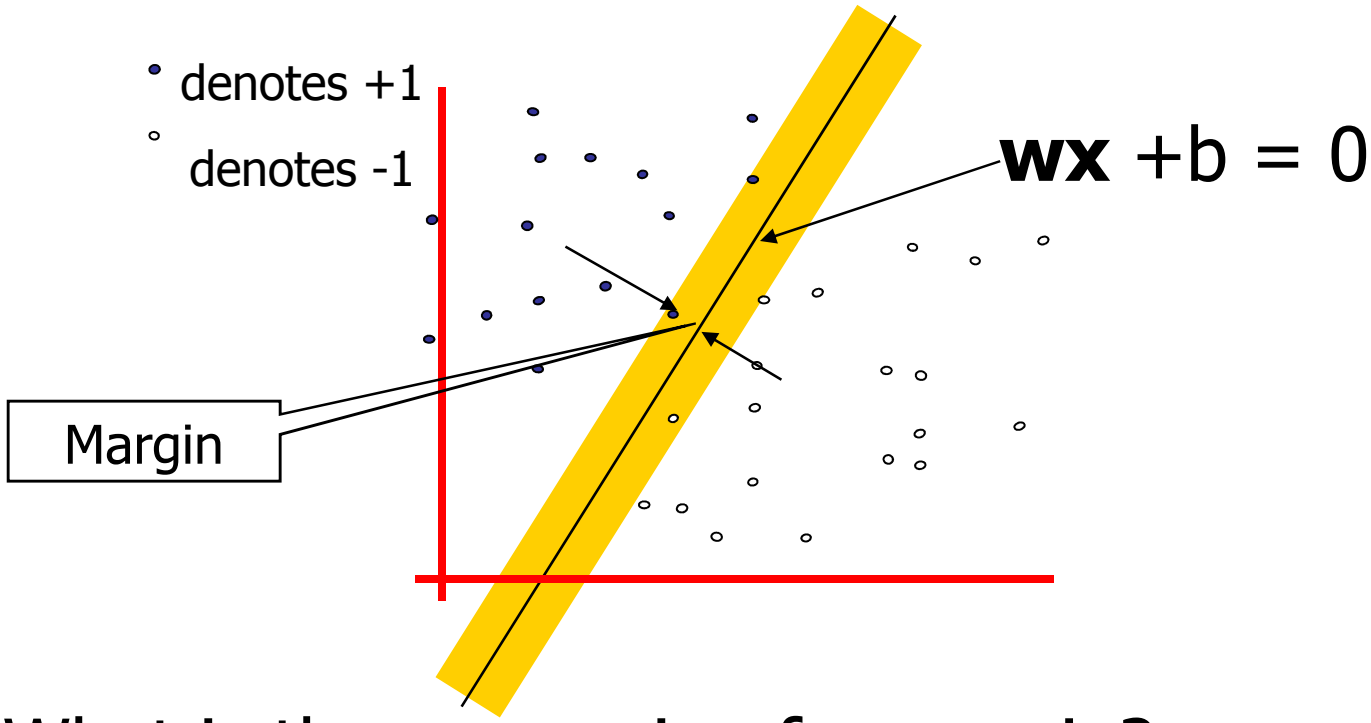
Estimate the Margin



- What is the distance expression for a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\left\|\mathbf{w}\right\|_2^2}} = \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

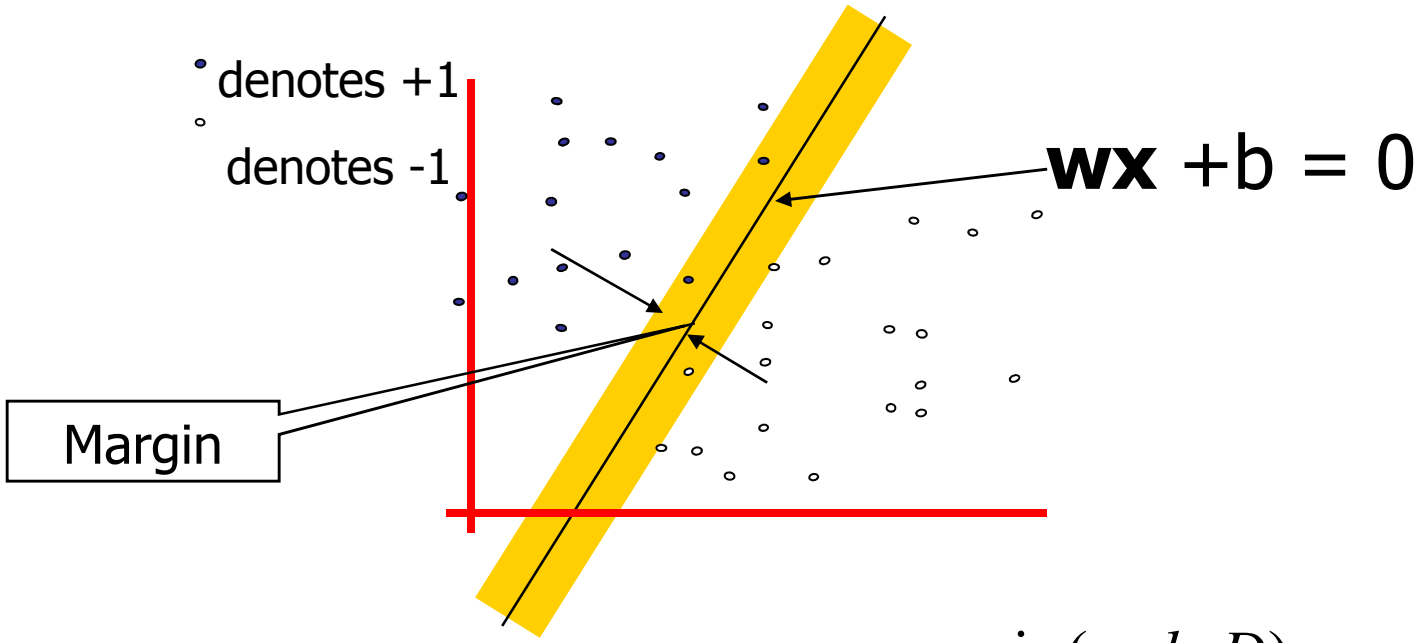
Estimate the Margin



- What is the expression for margin?

$$\text{margin} \equiv \min_{\mathbf{x}\in D} d(\mathbf{x}) = \min_{\mathbf{x}\in D} \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Maximize Margin

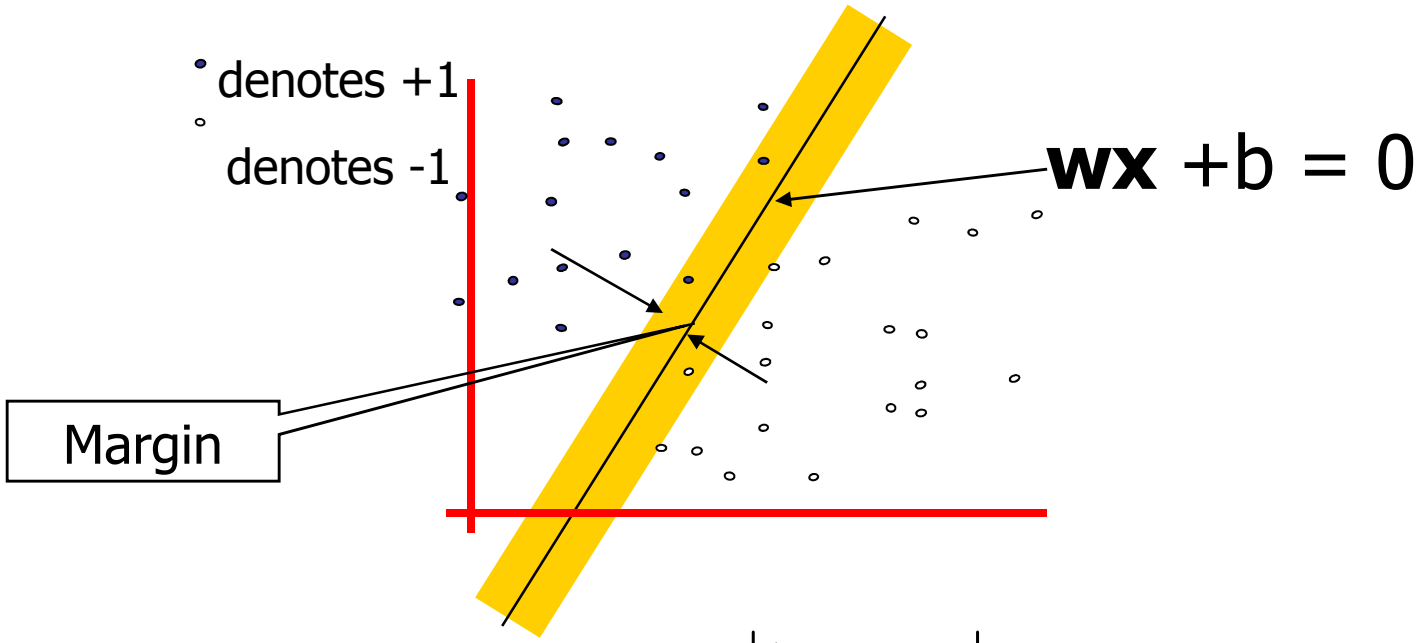


$$\underset{\mathbf{w},b}{\text{argmax}}\ \text{margin}(\mathbf{w},b,D)$$

$$= \underset{\mathbf{w},b}{\text{argmax}} \min_{\mathbf{x}_i \in D} d(\mathbf{x}_i)$$

$$= \underset{\mathbf{w},b}{\text{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left|b + \mathbf{x}_i \cdot \mathbf{w}\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

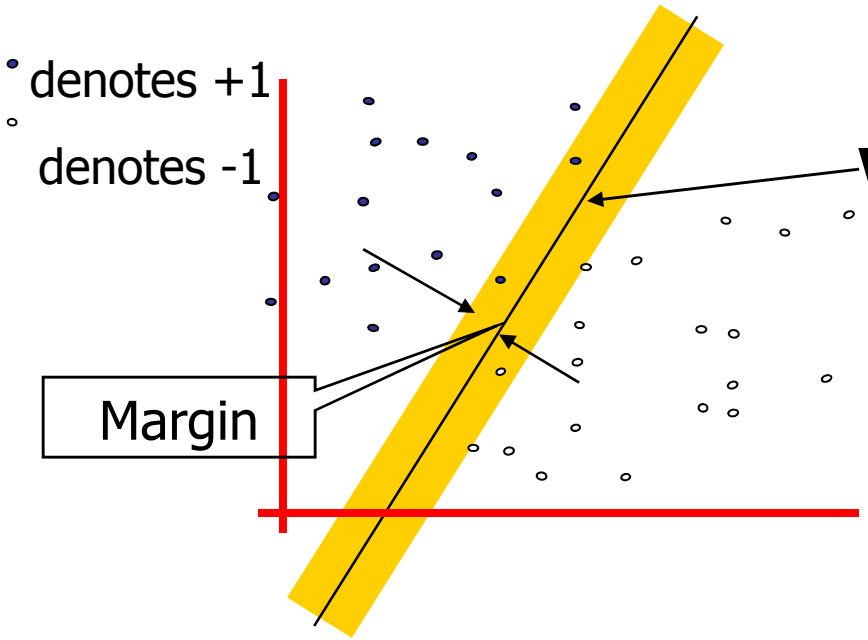
Maximize Margin



$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left| b + \mathbf{x}_i \cdot \mathbf{w} \right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{subject to } \forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) > 0$$

Maximize Margin



wx +b = 0

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left|b + \mathbf{x}_i \cdot \mathbf{w}\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{subject to } \forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 0$$

Strategy:

$$\forall \mathbf{x}_i \in D : \left|b + \mathbf{x}_i \cdot \mathbf{w}\right| \geq 1$$

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{d} w_i^2$$

$$\text{subject to } \forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) \geq 1$$

Maximum Margin Linear Classifier

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\operatorname{argmin}} \sum_{k=1}^{d} w_k^2$$

subject to

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1$$

....

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1$$

- How to solve it?

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

Subject to

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

n additional linear inequality constraints

And subject to

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

e additional linear equality constraints

Quadratic Programming

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

Subject to

And subj

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)

addi…nal linear
equality
…nstraints

$a_{(n+1)2}u_2 + \ldots = b_{(n+2)}$

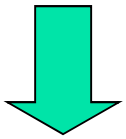
$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}$

e additional linear equality constraints

Quadratic Programming

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b} \sum_i w_i^2$$

subject to $y_i \left(\vec{w} \cdot \vec{x}_i + b\right) \geq 1$ for all training data $(\vec{x}_i, y_i)$



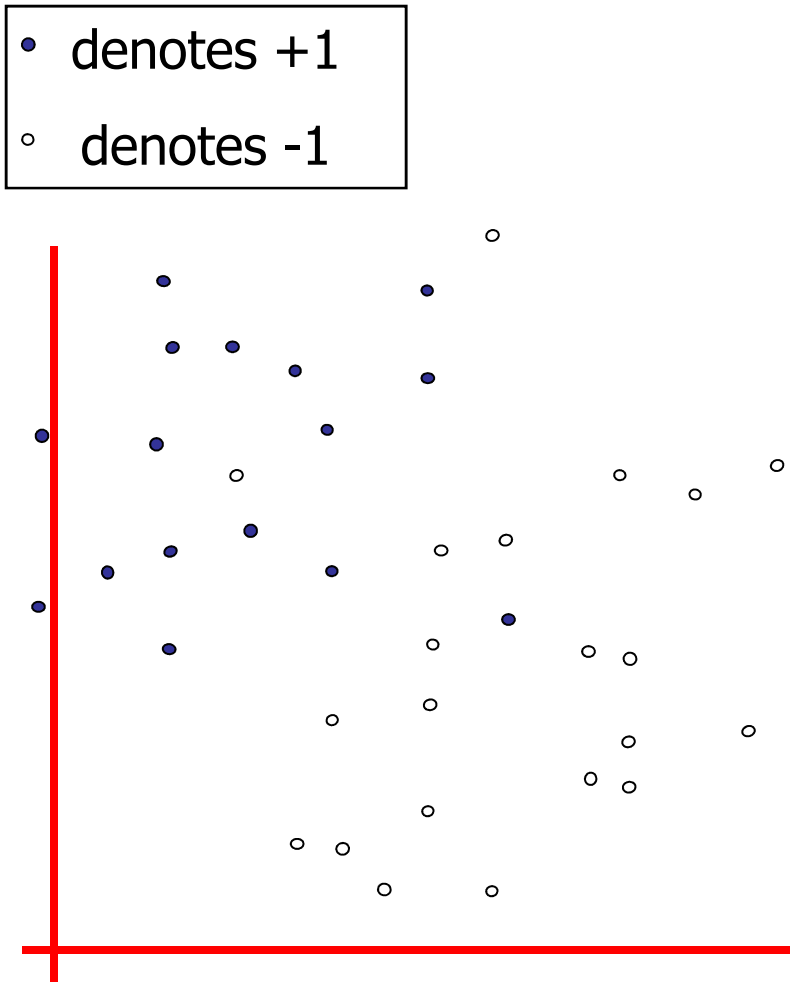
$$\{\vec{w}^*, b^*\} = \operatorname*{argmax}_{\vec{w},b} \left\{0 + \vec{0} \cdot \vec{w} - \vec{w}^T \mathbf{I_n} \vec{w}\right\}$$

$$\left.\begin{array}{l} y_1 \left(\vec{w} \cdot \vec{x}_1 + b\right) \geq 1 \\ y_2 \left(\vec{w} \cdot \vec{x}_2 + b\right) \geq 1 \\ .... \\ y_N \left(\vec{w} \cdot \vec{x}_N + b\right) \geq 1 \end{array}\right\} \text{inequality constraints}$$

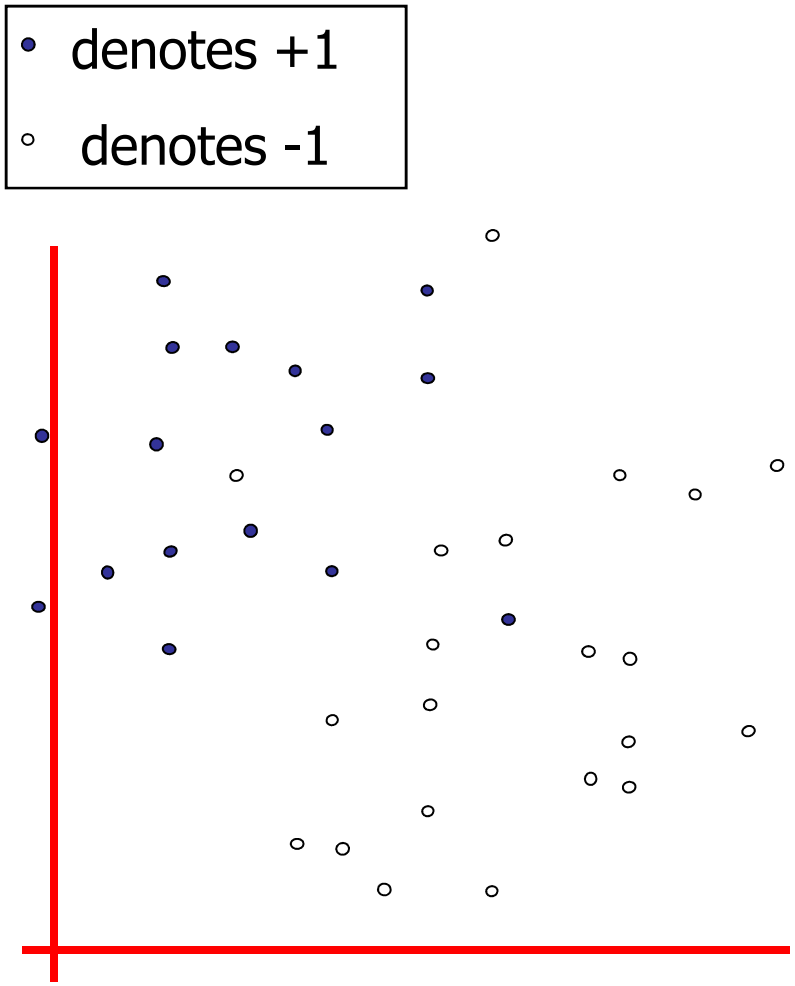
Uh-oh!

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!

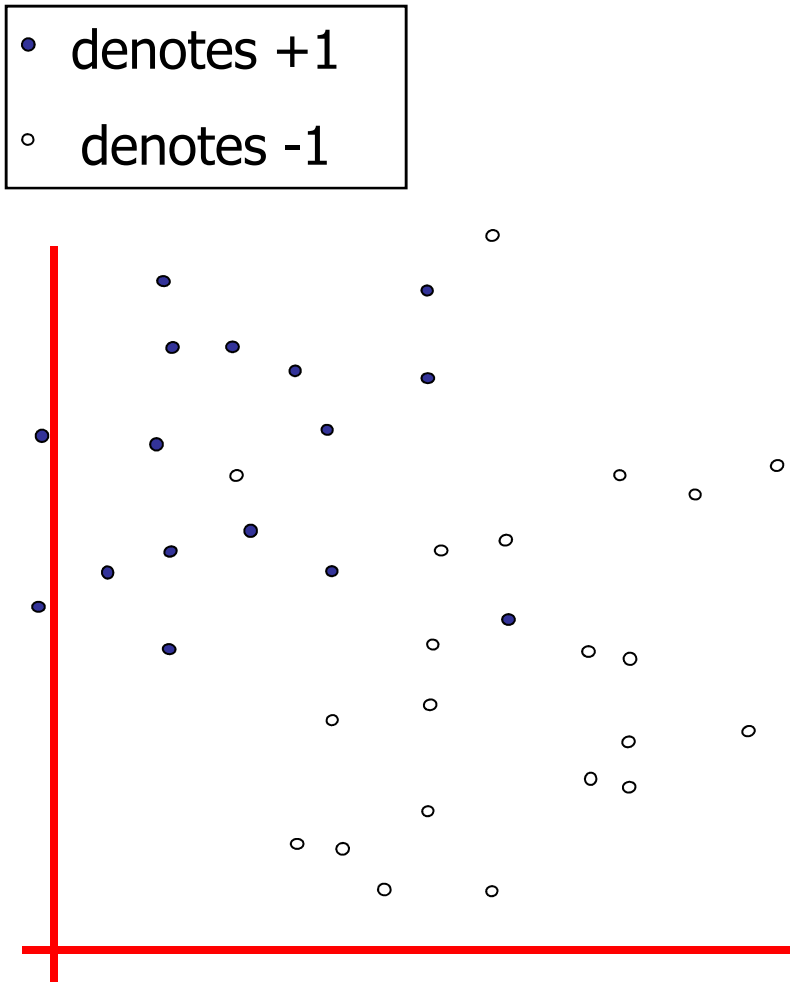
What should we do?

Idea 1.1:

Minimize

w.w *+ C (#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!

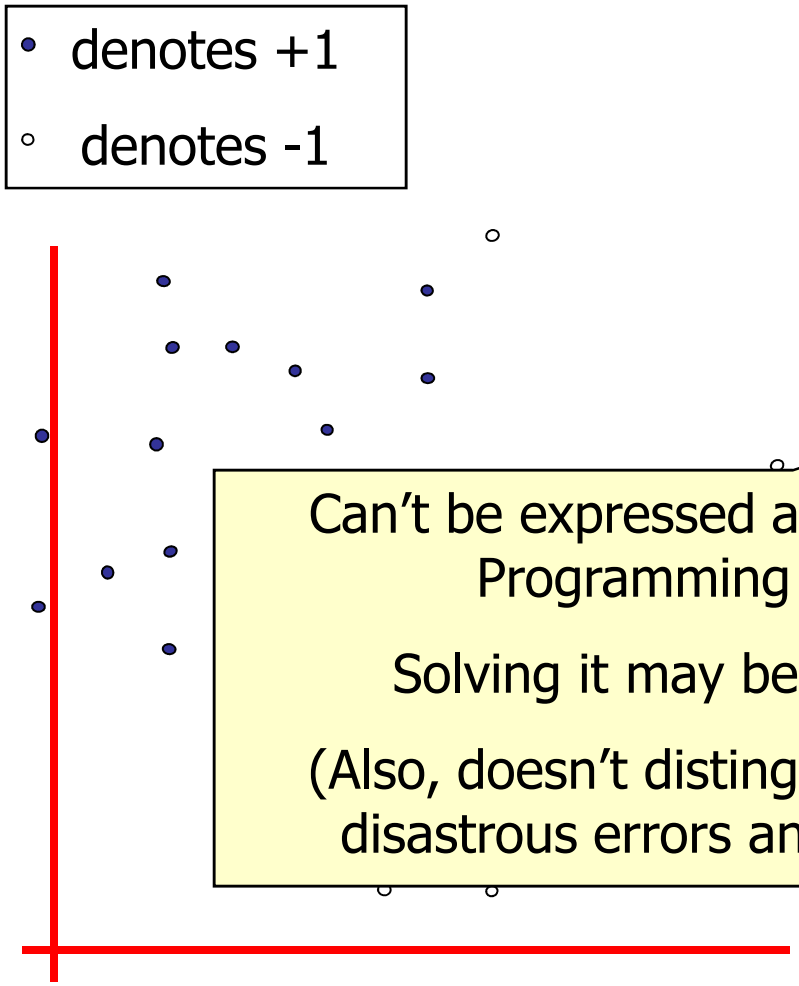
What should we do?

Idea 1.1:

Minimize

w.w *+ C (#train errors)*

Tradeoff parameter



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)

So… any other ideas?

Uh-oh!

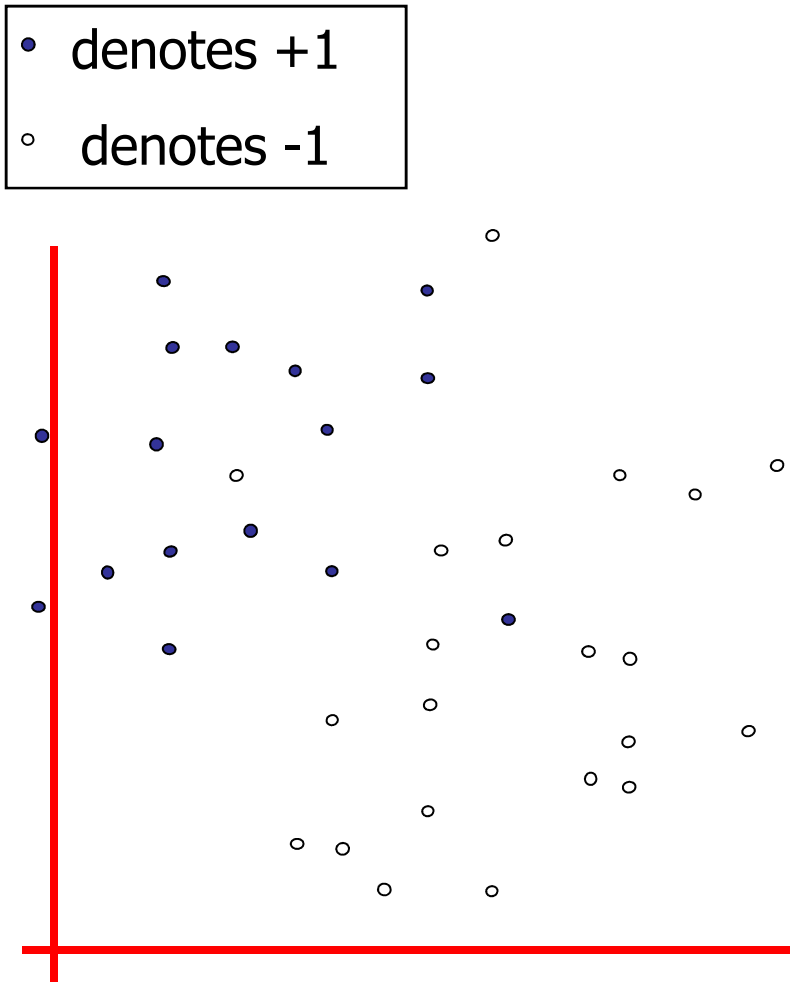
This is going to be a problem!

What should we do?

Idea 2.0:

Minimize

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



Support Vector Machine (SVM) for Noisy Data

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b,\vec{\varepsilon}} \sum_{i=1}^{\mathrm{d}} w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

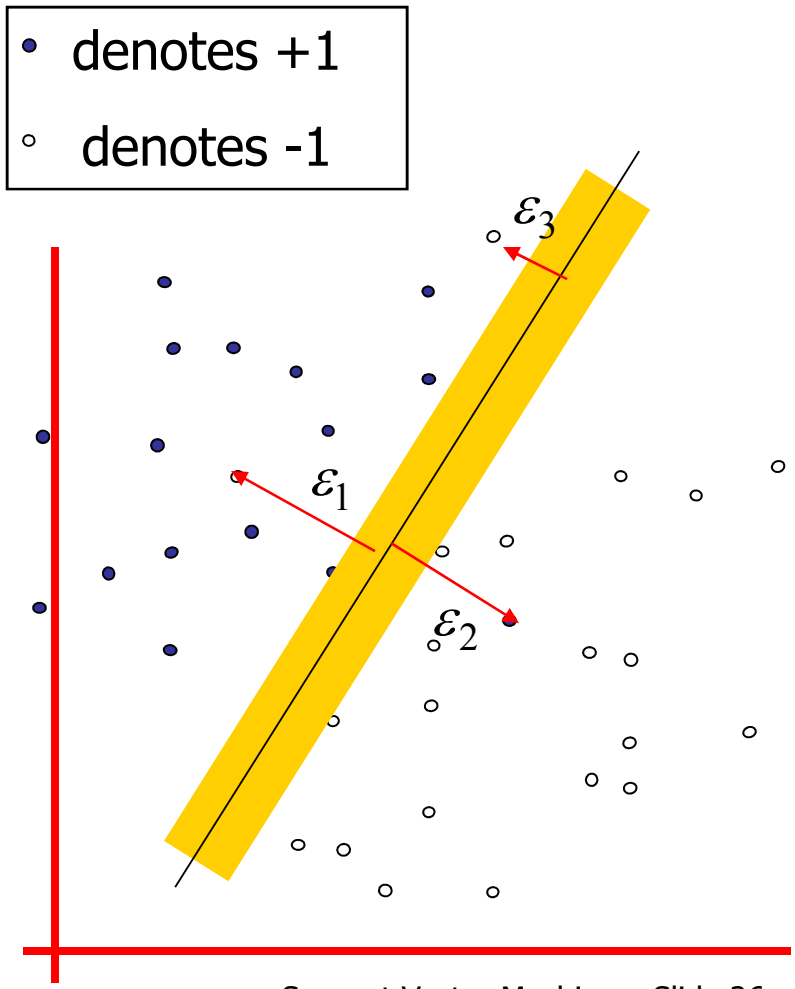
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1 - \varepsilon_1$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1 - \varepsilon_2$$

...

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1 - \varepsilon_N$$

- Any problem with the above formalism?



Support Vector Machine (SVM) for Noisy Data

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b,\vec{\varepsilon}} \sum_{i=1}^{\mathrm{d}} w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

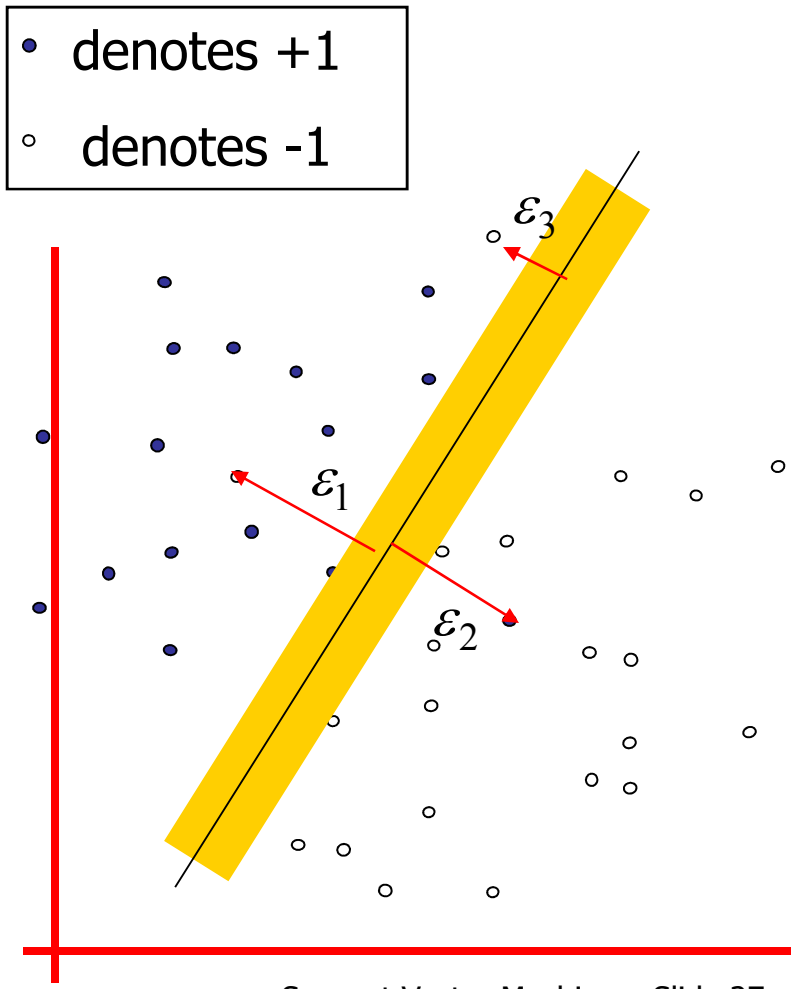
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0$$

...

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0$$

- Balance the trade off between margin and classification errors



Support Vector Machine for Noisy Data

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b,\vec{\varepsilon}}{\operatorname{argmin}} \sum_i w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

$$\left.\begin{array}{l} y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0 \\ y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0 \\ .... \\ y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0 \end{array}\right\} \text{inequality constraints}$$

How do we determine the appropriate value for *c* ?

The Dual Form of QP

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

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

Then define:

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

The Dual Form of QP

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

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

Then define:

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

Then classify with:

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

The Dual Form of QP

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

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

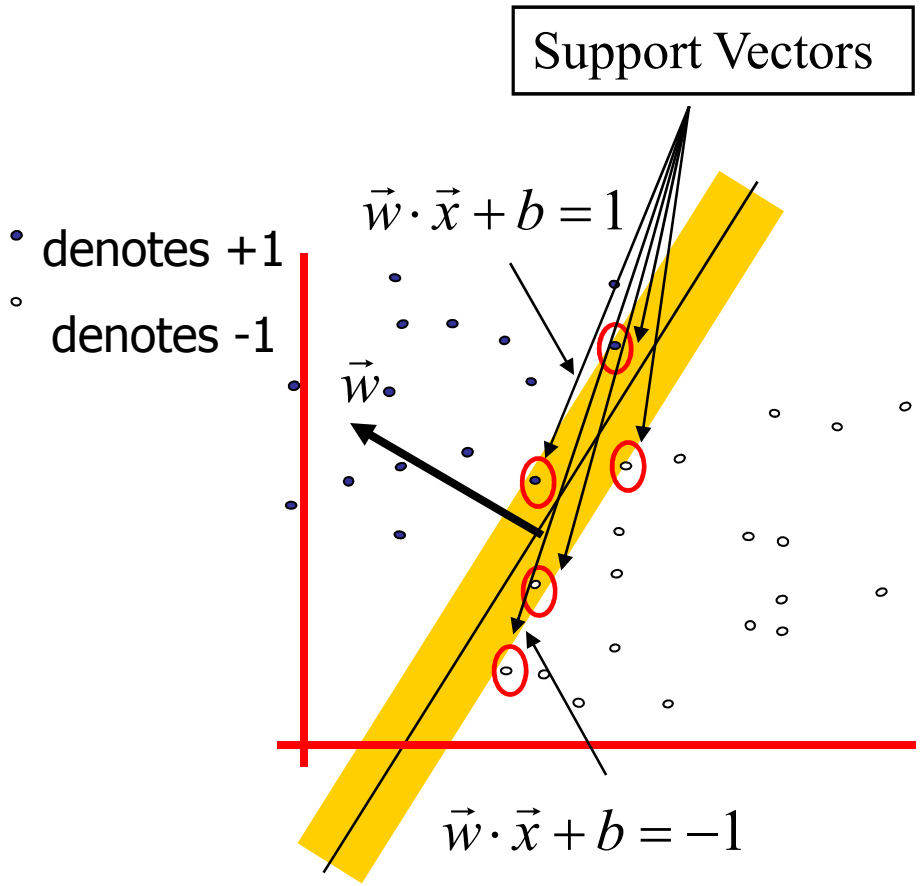
Then define:

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

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

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

Support Vectors



$$\forall i : \alpha_i \left(y_i \left(\vec{w} \cdot \vec{x}_i + b \right) - \left(1 - \varepsilon_i \right) \right) = 0$$

$\alpha_i = 0$ for non-support vectors

$\alpha_i \neq 0$ for support vectors

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

Decision boundary is determined only by those support vectors !

The Dual Form of QP

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

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

Then define:

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

Then classify with:

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

How to determine *b* ?

An Equivalent QP: Determine *b*

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\operatorname{argmin}} \sum_i w_i^2 + c\sum_{j=1}^{N} \varepsilon_j$$

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0$$

....

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0$$

Fix **w**

$$b^* = \underset{b,\{\varepsilon_i\}_{i=1}^{N}}{\operatorname{argmin}} \sum_{j=1}^{N} \varepsilon_j$$

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1-\varepsilon_2, \varepsilon_2 \geq 0$$

....

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1-\varepsilon_N, \varepsilon_N \geq 0$$

A linear programming problem !

An Equivalent QP

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

Subject to these constraints: $\alpha_K \geq 0$

Then define:

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

$$b = y_K (1 - \alpha_K \mathbf{x}_K . \mathbf{x}_K) - \sum_{k=1}^{R} \alpha_k y_k (\mathbf{x}_k . \mathbf{x}_K)$$

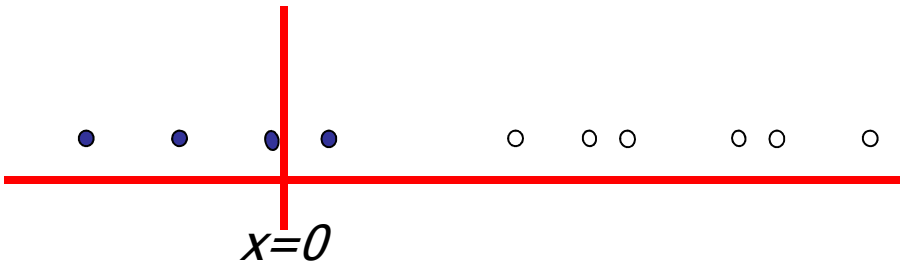
where $K = \arg \max_k \alpha_k$

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.

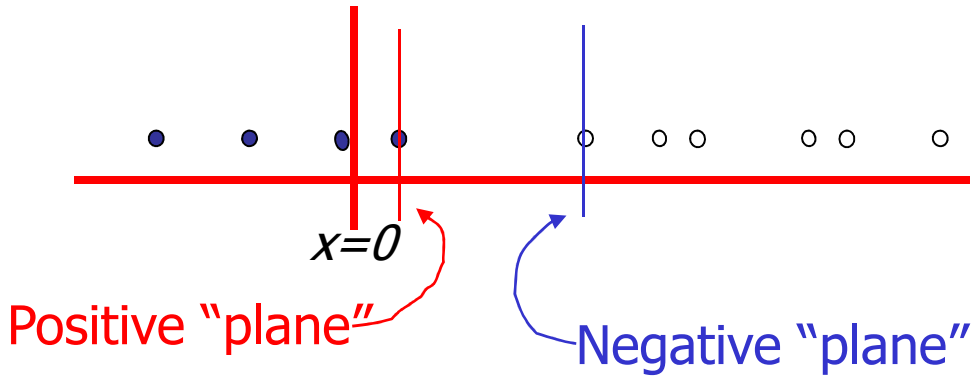
Suppose we're in 1-dimension

What would SVMs do with this data?

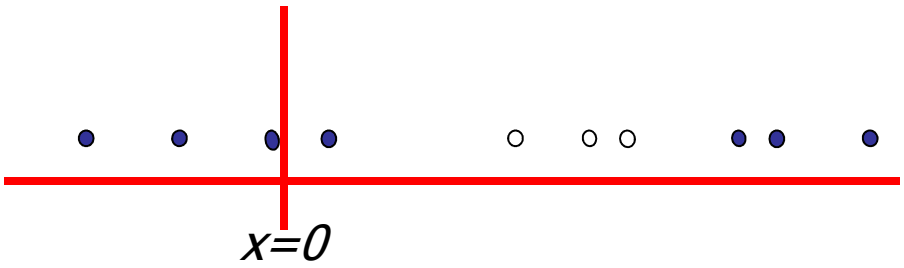


Suppose we're in 1-dimension

Not a big surprise



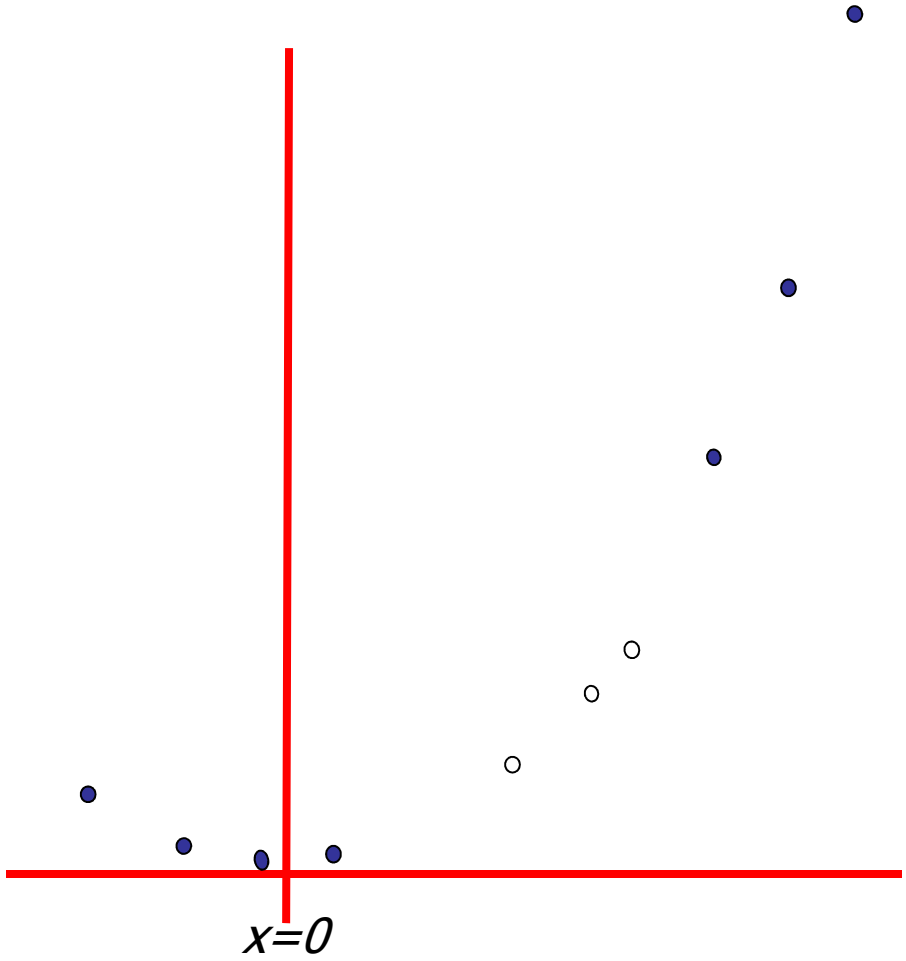
Harder 1-dimensional dataset



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

What can be done about this?

Harder 1-dimensional dataset

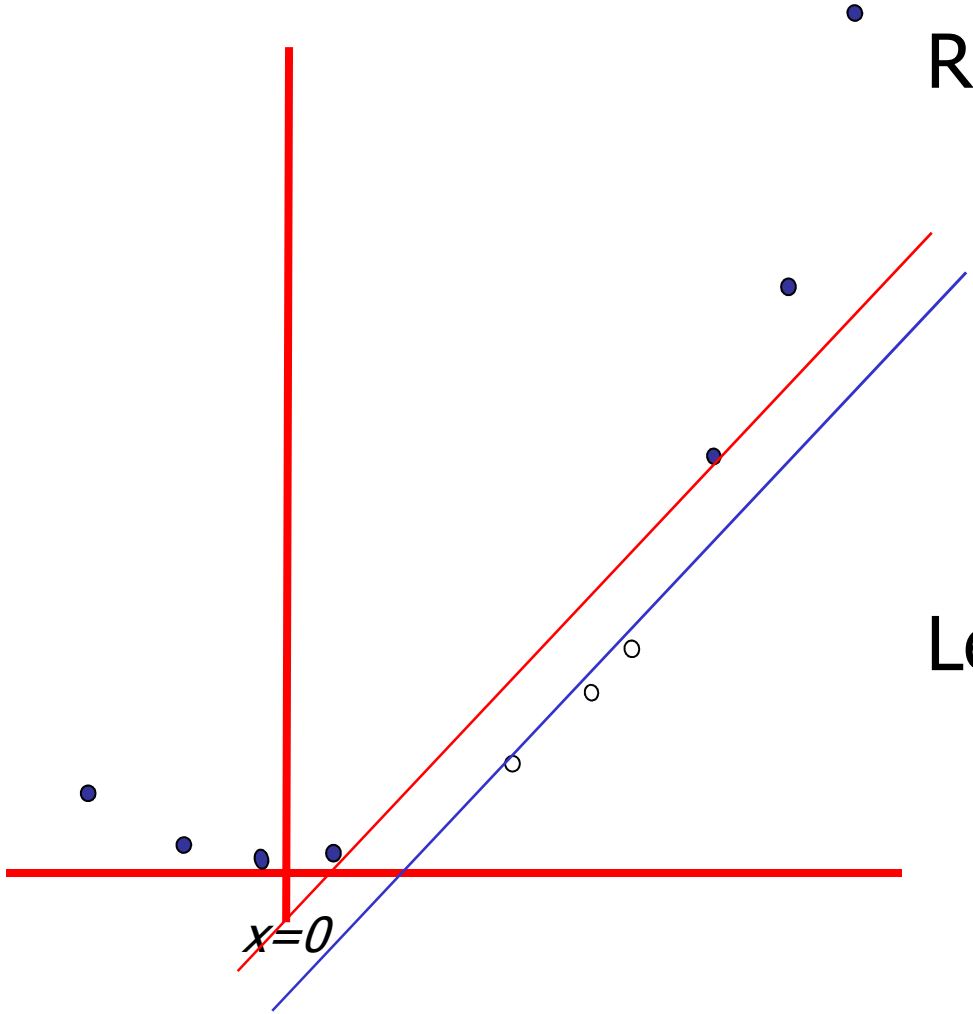


Remember how permitting non-linear basis functions made linear regression so much nicer?

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 $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \exp\left(-\frac{|\mathbf{x}_k - \mathbf{c}_j|^2}{\sigma^2}\right)$$

$\boldsymbol{z}_k =$ (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

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

Constant Term

Linear Terms

Pure Quadratic Terms

Quadratic Cross-Terms

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 (old)

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

Then define:

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

Then classify with:

f(x,w,b) = sign(w. 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: $0 \le \alpha_k \le C \quad \forall k$ $\sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

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

Then classify with:

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

Most important changes:

$\mathbf{x} \rightarrow \phi(\mathbf{x})$

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

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

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?

Then define:

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

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

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

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

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

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***:

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

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

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

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

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

They're the same!

And this is only O(m) to compute!

QP with Quadratic basis functions

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

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

Then define:

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

Then classify with:

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

QP with Quadratic 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 $Q_{kl} = y_k y_l K(\mathbf{x}_k, \mathbf{x}_l)$

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

Then define:

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

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

where $K = \arg\max_k \alpha_k$

Then classify with:

f(x,w,b) = sign(K(w, x) - b)

Most important change:

$$\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l) \rightarrow K(\mathbf{x}_k, \mathbf{x}_l)$$

Higher Order Polynomials

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

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:

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

Kernel Tricks

- Replacing dot product with a kernel function
- Not all functions are kernel functions
  - Need to be decomposable
    - $K(a,b) = \overrightarrow{\phi(a)} \cdot \overrightarrow{\phi(b)}$
  - Could $K(a,b) = (a-b)^3$ be a kernel function ?
  - Could $K(a,b) = (a-b)^4 - (a+b)^2$ be a kernel function?

Kernel Tricks

- Mercer's condition

  To expand Kernel function K(x,y) into a dot product, i.e. $K(x,y)=\Phi(x)\cdot\Phi(y)$, K(x, y) has to be positive semi-definite function, i.e., for any function f(x) whose $\int f^2(x)dx$ is finite, the following inequality holds

$$\int dxdyf(x)K(x,y)f(y) \geq 0$$

- Could $K(\vec{x},\vec{y}) = \left(\sum_i x_i y_i\right)^p$ be a kernel function?

Kernel Tricks

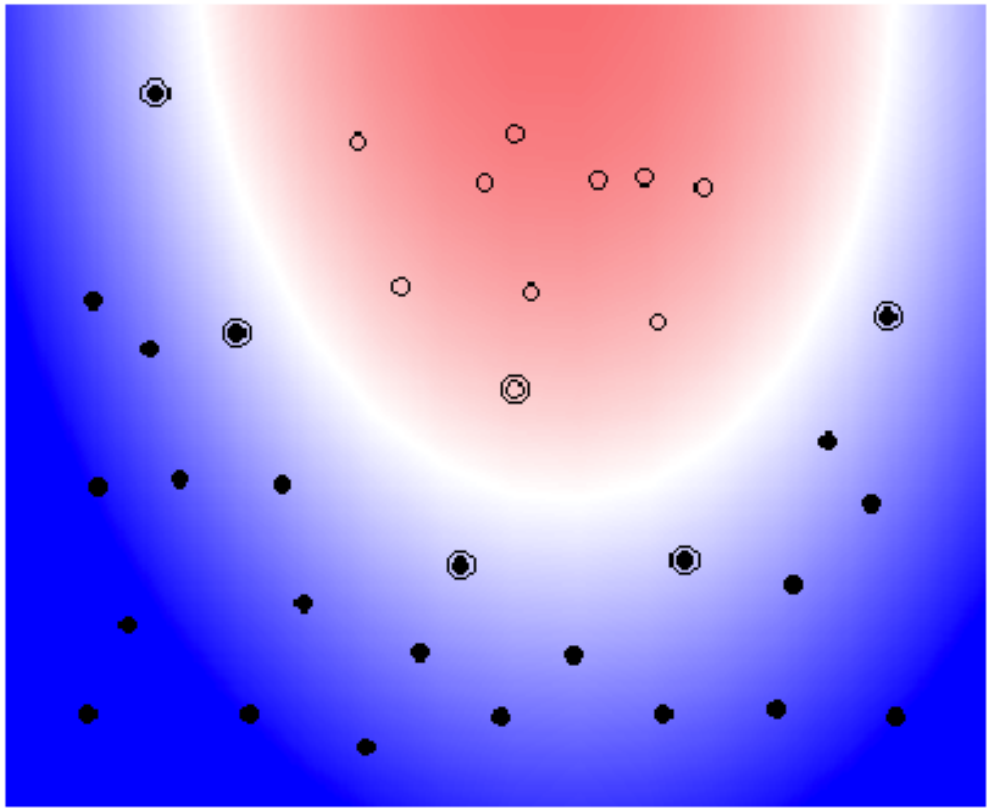
- Pro
  - Introducing nonlinearity into the model
  - Computational cheap
- Con
  - Still have potential overfitting problems

Nonlinear Kernel (I)

Example: SVM with Polynomial of Degree 2

Kernel: $K(\vec{x}_i, \vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$

plot by Bell SVM applet

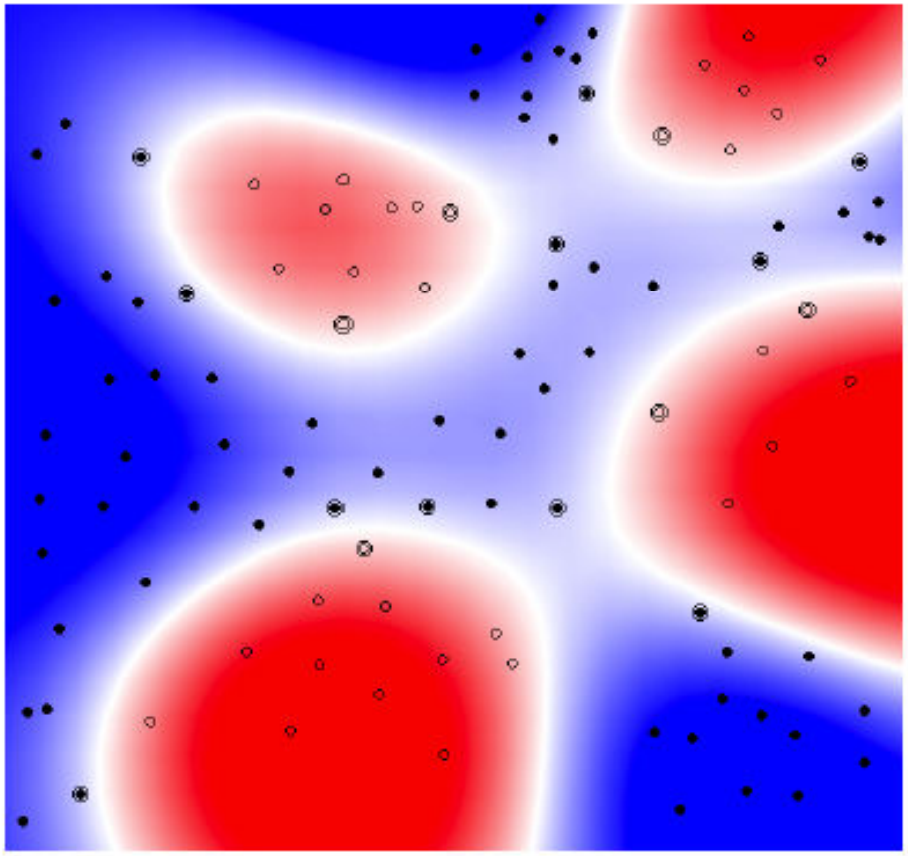


Nonlinear Kernel (II)

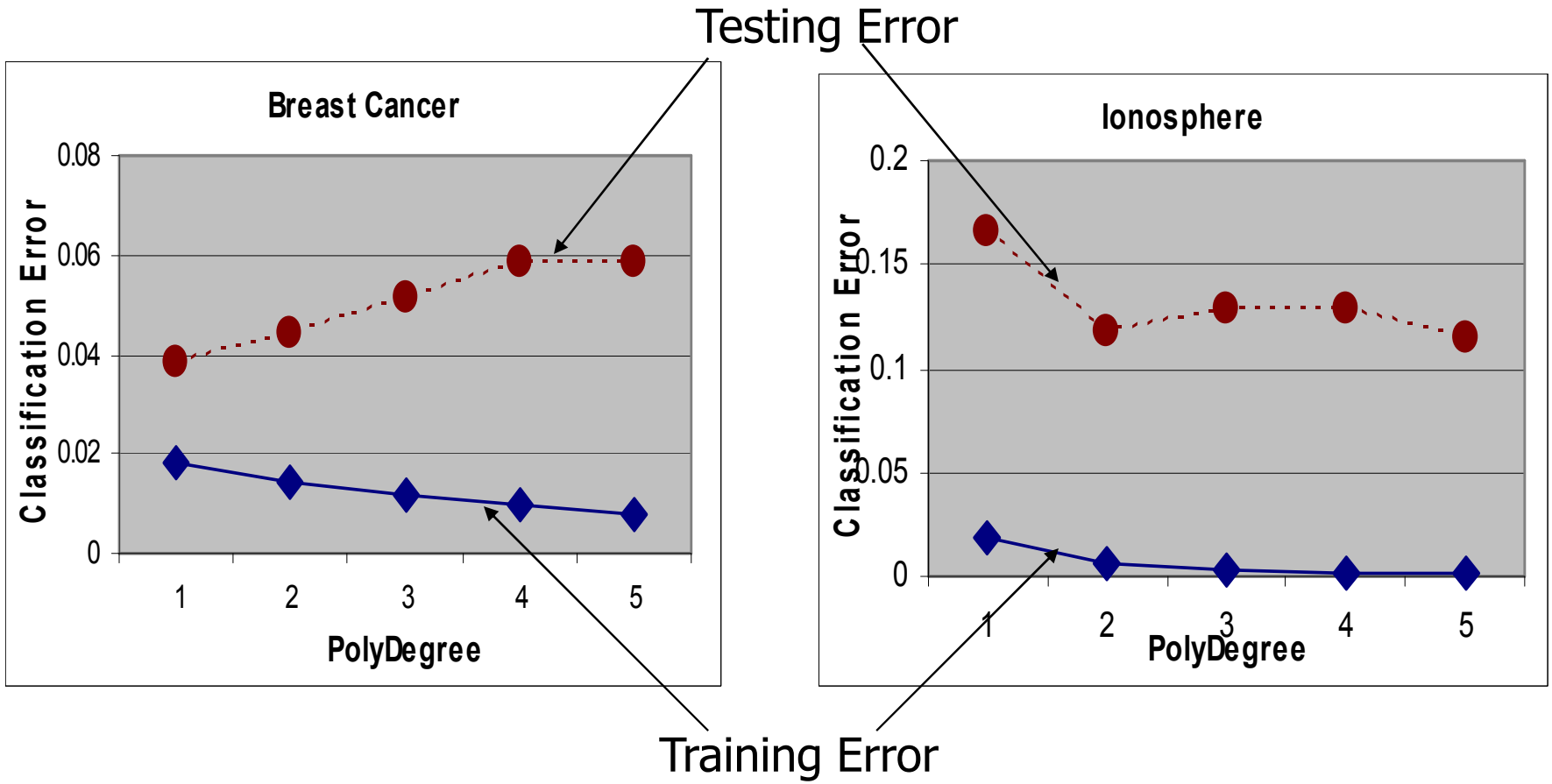
Example: SVM with RBF-Kernel

Kernel: $K(\vec{x}_i,\vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2/\sigma^2)$

plot by Bell SVM applet



Overfitting in SVM



SVM Performance

- Anecdotally they work very very well indeed.
- Example: They are currently (in 2003) the best-known classifier on a well-studied hand-written-character recognition benchmark
- There is a lot of excitement and religious fervor about SVMs as of 2001.
- Despite this, some practitioners are a little skeptical.

Doing multi-class classification

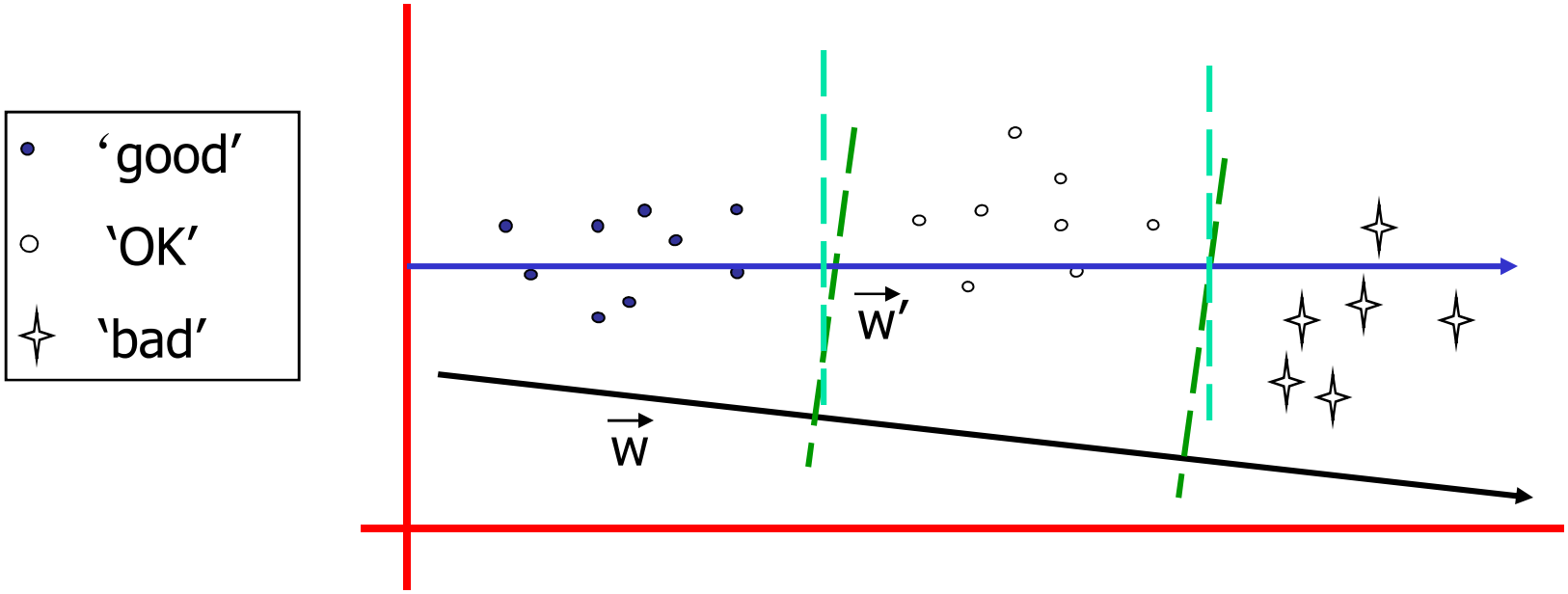
- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?
- Answer: with output arity N, learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - :
  - SVM N learns "Output==N" vs "Output != N"
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Ranking Problem

- Consider a problem of ranking essays
  - Three ranking categories: good, ok, bad
  - Given a input document, predict its ranking category
- How should we formulate this problem?
- A simple multiple class solution
  - Each ranking category is a independent class
- But, there is something missing here …

- We miss the ordinal relationship between classes !

Ordinal Regression



- Which choice is better?
- How could we formulate this problem?

Ordinal Regression

- What are the two decision boundaries?

Ordinal Regression

- What are the two decision boundaries?

$$\mathbf{w} \cdot \mathbf{x} + b_1 = 0 \text{ and } \mathbf{w} \cdot \mathbf{x} + b_2 = 0$$

- What is the margin for ordinal regression?

Ordinal Regression

- What are the two decision boundaries?

$$\mathbf{w}\cdot\mathbf{x}+b_1=0 \text{ and } \mathbf{w}\cdot\mathbf{x}+b_2=0$$

- What is the margin for ordinal regression?

$$\text{margin}_1(\mathbf{w},b_1) \equiv \min_{\mathbf{x}\in D_g\cup D_o} d(\mathbf{x}) = \min_{\mathbf{x}\in D_g\cup D_o} \frac{\left|\mathbf{x}\cdot\mathbf{w}+b_1\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{margin}_2(\mathbf{w},b_2) \equiv \min_{\mathbf{x}\in D_o\cup D_b} d(\mathbf{x}) = \min_{\mathbf{x}\in D_o\cup D_b} \frac{\left|\mathbf{x}\cdot\mathbf{w}+b_2\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{margin}(\mathbf{w},b_1,b_2)=\min(\text{margin}_1(\mathbf{w},b_1),\text{margin}_2(\mathbf{w},b_2))$$

- Maximize margin $\{\mathbf{w}^*,b_1^*,b_2^*\} = \arg\max_{\mathbf{w},b_1,b_2} \text{margin}(\mathbf{w},b_1,b_2)$

Ordinal Regression

$$\{\mathbf{w}^*, b_1^*, b_2^*\} = \arg\max_{\mathbf{w}, b_1, b_2} \text{margin}(\mathbf{w}, b_1, b_2)$$

$$= \arg\max_{\mathbf{w}, b_1, b_2} \min(\text{margin}_1(\mathbf{w}, b_1), \text{margin}_2(\mathbf{w}, b_2))$$

$$= \arg\max_{\mathbf{w}, b_1, b_2} \min\left(\min_{\mathbf{x} \in D_g \cup D_o} \frac{|\mathbf{x} \cdot \mathbf{w} + b_1|}{\sqrt{\sum_{i=1}^{d} w_i^2}}, \min_{\mathbf{x} \in D_o \cup D_b} \frac{|\mathbf{x} \cdot \mathbf{w} + b_2|}{\sqrt{\sum_{i=1}^{d} w_i^2}} \right)$$

subject to

$$\forall \mathbf{x}_i \in D_g : \mathbf{x}_i \cdot \mathbf{w} + b_1 > 0$$

$$\forall \mathbf{x}_i \in D_o : \mathbf{x}_i \cdot \mathbf{w} + b_1 < 0, \mathbf{x}_i \cdot \mathbf{w} + b_2 > 0$$

$$\forall \mathbf{x}_i \in D_b : \mathbf{x}_i \cdot \mathbf{w} + b_2 < 0$$

- How do we solve this monster ?

Ordinal Regression

- The same old trick
- To remove the scaling invariance, set

$$\forall \mathbf{x}_i \in D_g \cup D_o : \left| b_1 + \mathbf{x}_i \cdot \mathbf{w} \right| \geq 1$$
$$\forall \mathbf{x}_i \in D_o \cup D_b : \left| b_2 + \mathbf{x}_i \cdot \mathbf{w} \right| \geq 1$$

- Now the problem is simplified as:

$$\{\mathbf{w}^*, b_1^*, b_2^*\} = \arg\min_{\mathbf{w}, b_1, b_2} \sum_{i=1}^{d} w_i^2$$

subject to

$$\forall \mathbf{x}_i \in D_g : \mathbf{x}_i \cdot \mathbf{w} + b_1 \geq 1$$
$$\forall \mathbf{x}_i \in D_o : \mathbf{x}_i \cdot \mathbf{w} + b_1 \leq -1, \mathbf{x}_i \cdot \mathbf{w} + b_2 \geq 1$$
$$\forall \mathbf{x}_i \in D_b : \mathbf{x}_i \cdot \mathbf{w} + b_2 \leq -1$$

Ordinal Regression

- Noisy case

$$\{\mathbf{w}^*, b_1^*, b_2^*\} = \arg\min_{\mathbf{w}, b_1, b_2} \sum_{i=1}^{d} w_i^2 + c \sum_{x_i \in D_g} \varepsilon_i^+ + c \sum_{x_i \in D_o} \left(\varepsilon_i^+ + \varepsilon_i^-\right) + c \sum_{x_i \in D_b} \varepsilon_i^-$$

subject to

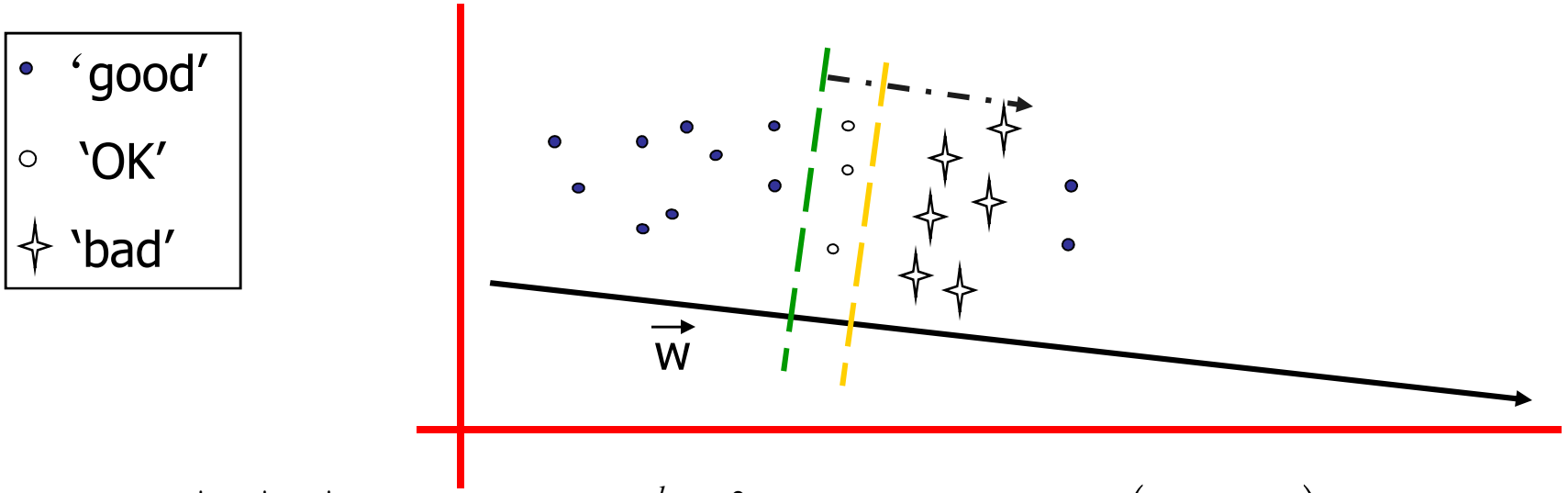
$$\forall \mathbf{x}_i \in D_g : \mathbf{x}_i \cdot \mathbf{w} + b_1 \geq 1 - \varepsilon_i^+, \varepsilon_i^+ \geq 0$$

$$\forall \mathbf{x}_i \in D_o : \mathbf{x}_i \cdot \mathbf{w} + b_1 \leq -1 + \varepsilon_i^-, \mathbf{x}_i \cdot \mathbf{w} + b_2 \geq 1 - \varepsilon_i^+, \varepsilon_i^+ \geq 0, \varepsilon_i^- \geq 0$$

$$\forall \mathbf{x}_i \in D_b : \mathbf{x}_i \cdot \mathbf{w} + b_2 \leq -1 + \varepsilon_i^-, \varepsilon_i^- \geq 0$$

- Is this sufficient enough?

Ordinal Regression



$$\{\mathbf{w}^*, b_1^*, b_2^*\} = \arg\min_{\mathbf{w}, b_1, b_2} \sum_{i=1}^{d} w_i^2 + c \sum_{x_i \in D_g} \varepsilon_i^+ + c \sum_{x_i \in D_o} \left(\varepsilon_i^+ + \varepsilon_i^-\right) + c \sum_{x_i \in D_b} \varepsilon_i^-$$

subject to

$$\forall \mathbf{x}_i \in D_g : \mathbf{x}_i \cdot \mathbf{w} + b_1 \geq 1 - \varepsilon_i^+, \varepsilon_i^+ \geq 0$$

$$\forall \mathbf{x}_i \in D_o : \mathbf{x}_i \cdot \mathbf{w} + b_1 \leq -1 + \varepsilon_i^-, \mathbf{x}_i \cdot \mathbf{w} + b_2 \geq 1 - \varepsilon_i^+, \varepsilon_i^+ \geq 0, \varepsilon_i^- \geq 0$$

$$\forall \mathbf{x}_i \in D_b : \mathbf{x}_i \cdot \mathbf{w} + b_2 \leq -1 + \varepsilon_i^-, \varepsilon_i^- \geq 0$$

$$b_1 \leq b_2$$

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: (Not for beginners including myself)

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

- Software (C and matlab wrappers): http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Software (fast): SVM-light, http://svmlight.joachims.org/