SSL – Course 6 Support Vector Machines

Adaptation of the presentation by

Andrew W. Moore

Professor

School of Computer Science

Carnegie Mellon University

www.cs.cmu.edu/~awm

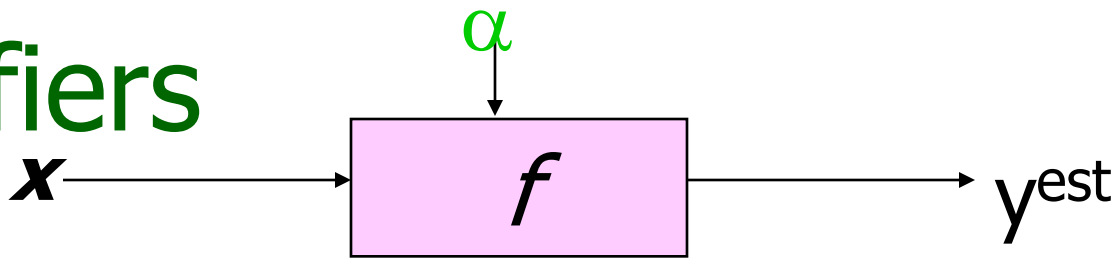
awm@cs.cmu.edu

412-268-7599

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew's tutorials: http://www.cs.cmu.edu/~awm/tutorials . Comments and corrections gratefully received.

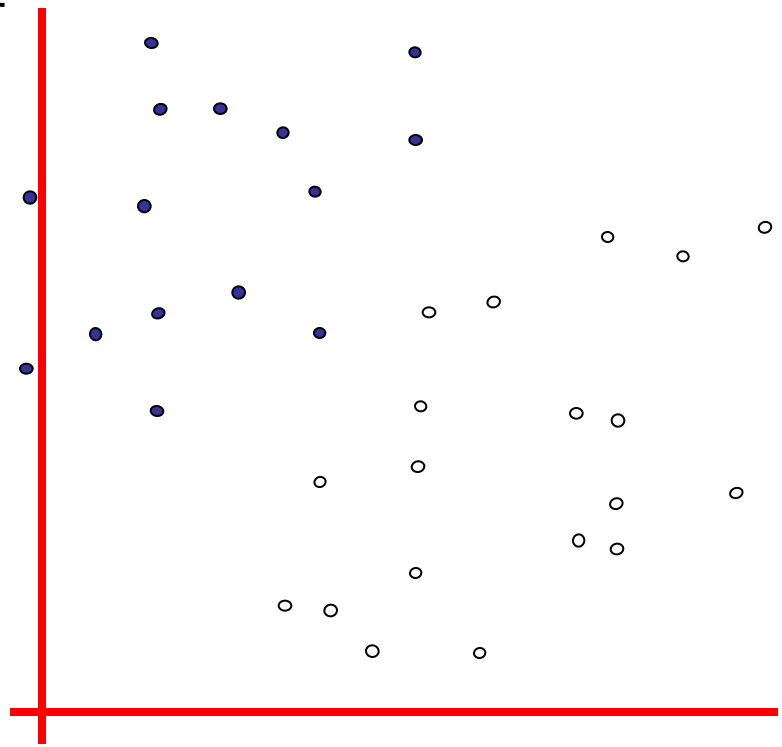
Nov 23rd, 2001

Linear Classifiers



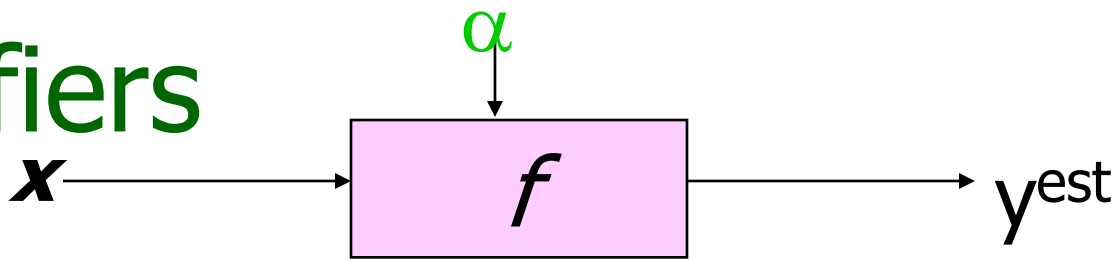
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} + b)$

- denotes +1
- denotes -1



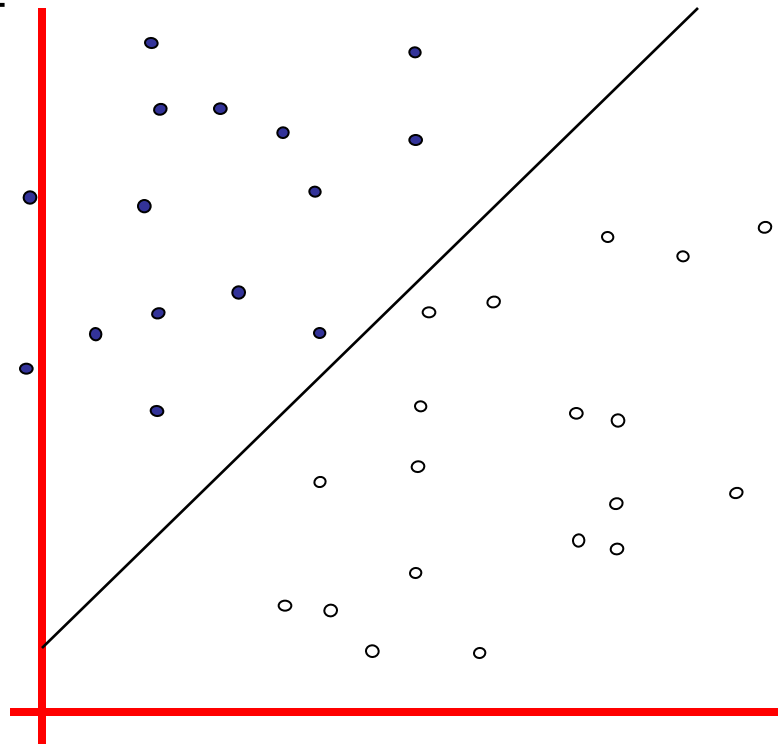
How would you classify this data?

Linear Classifiers



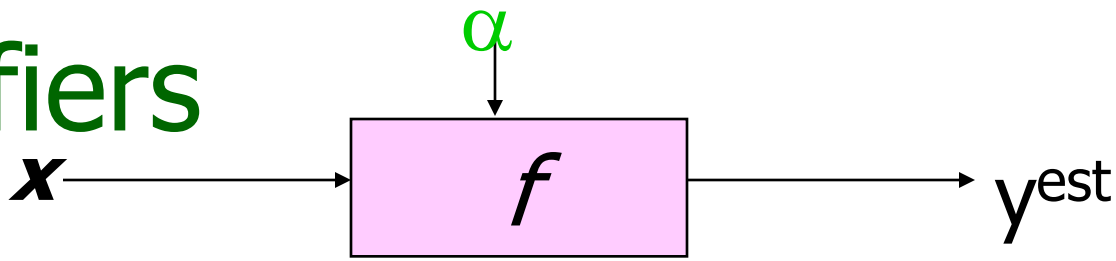
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} + b)$

- denotes +1
- denotes -1



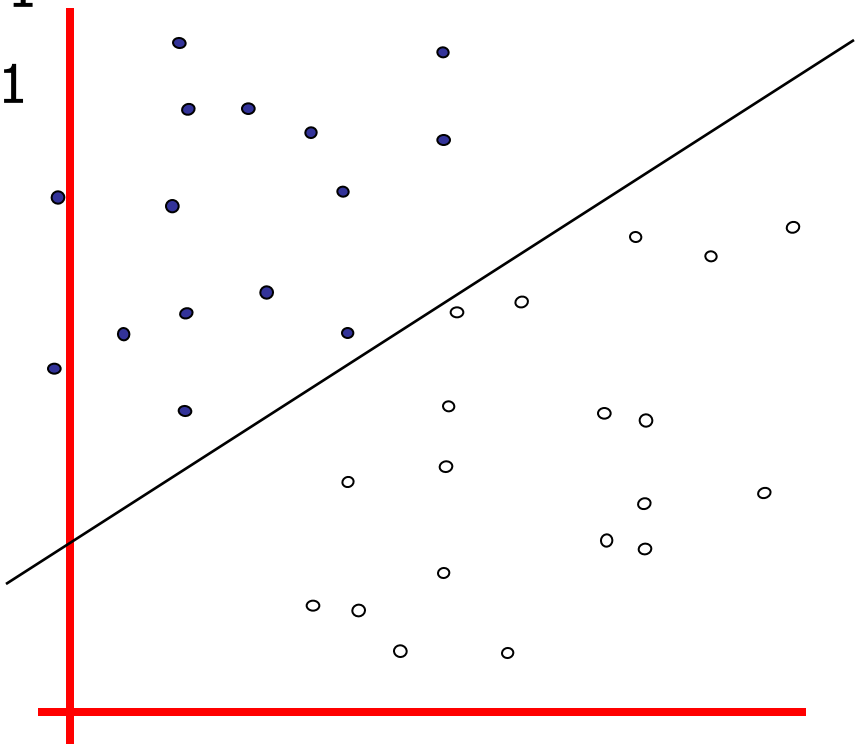
How would you classify this data?

Linear Classifiers



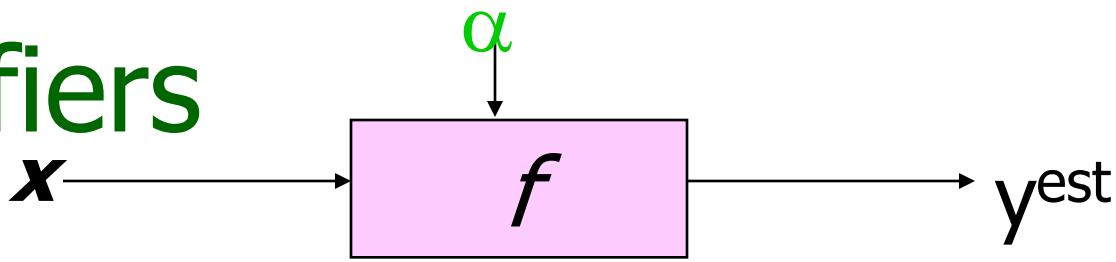
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} + b)$

- denotes +1
- denotes -1



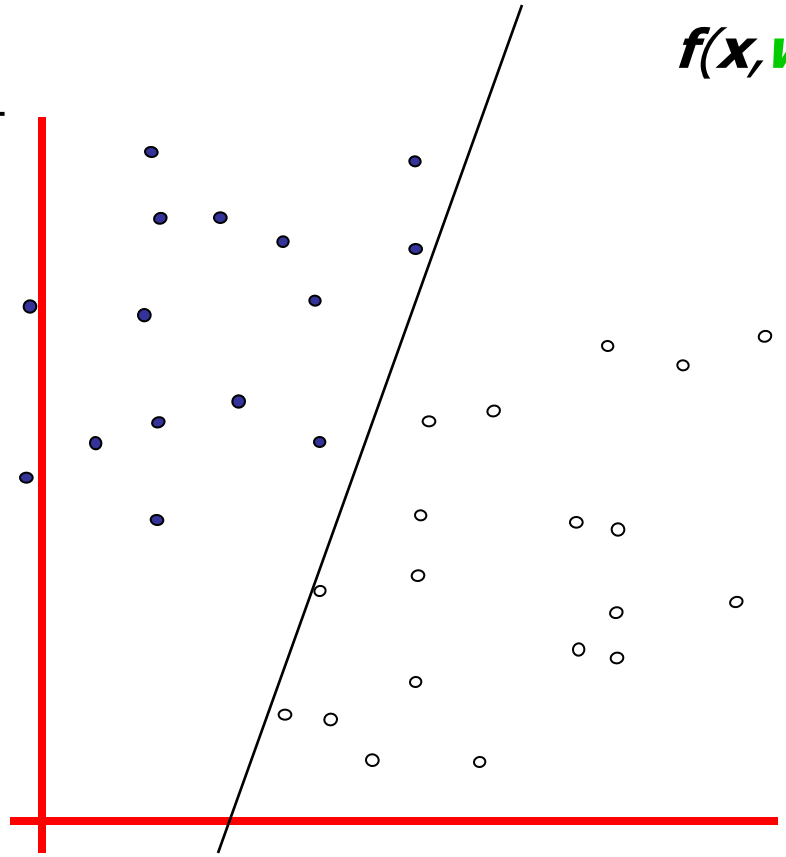
How would you classify this data?

Linear Classifiers



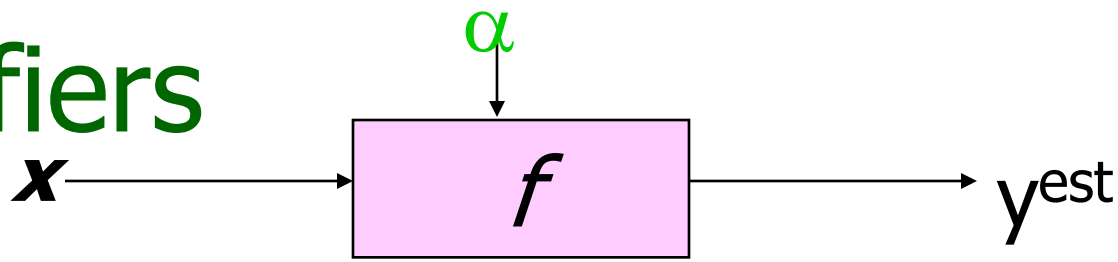
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} + b)$

- denotes +1
- denotes -1



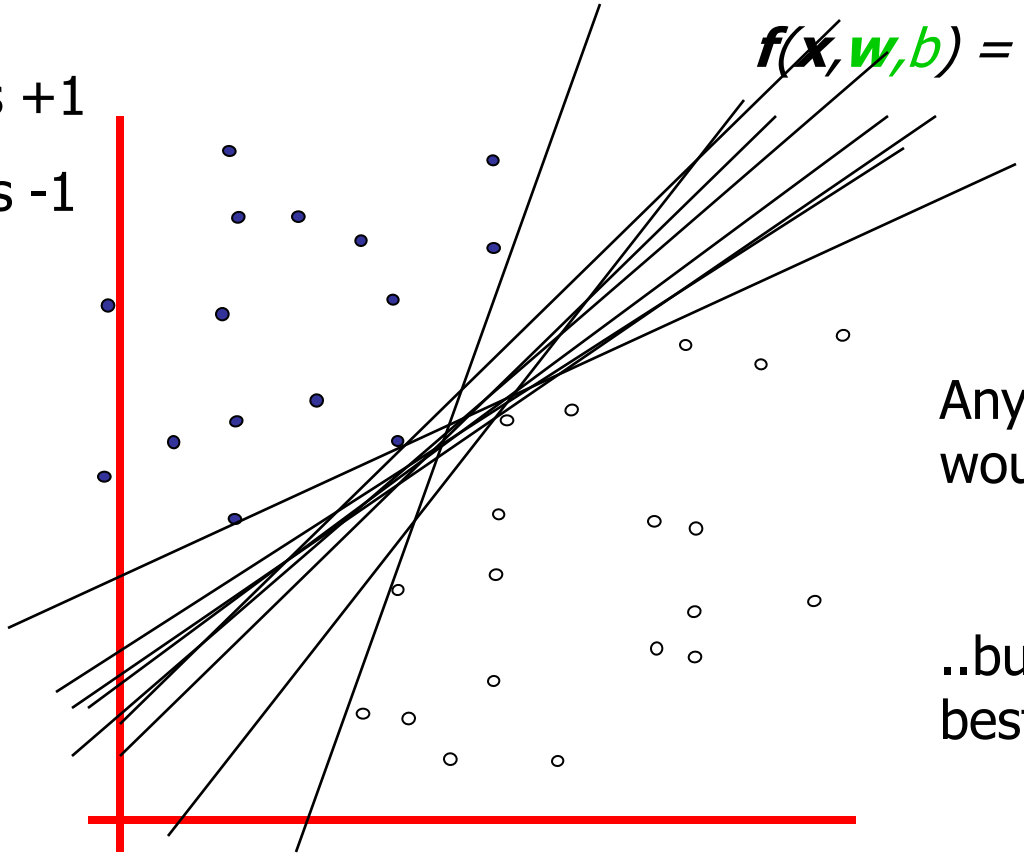
How would you classify this data?

Linear Classifiers



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

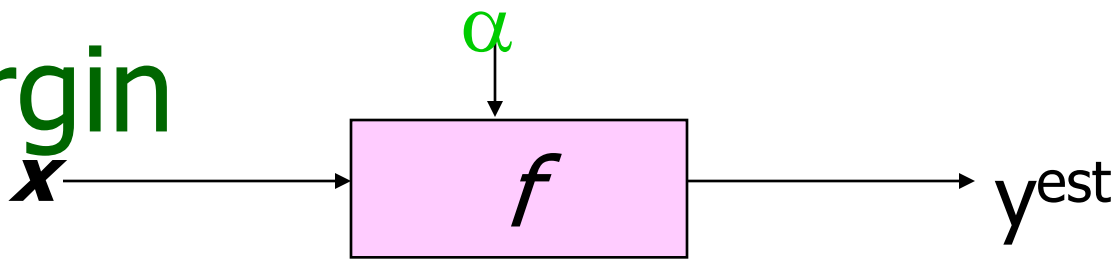
- denotes +1
- denotes -1



Any of these would be fine..

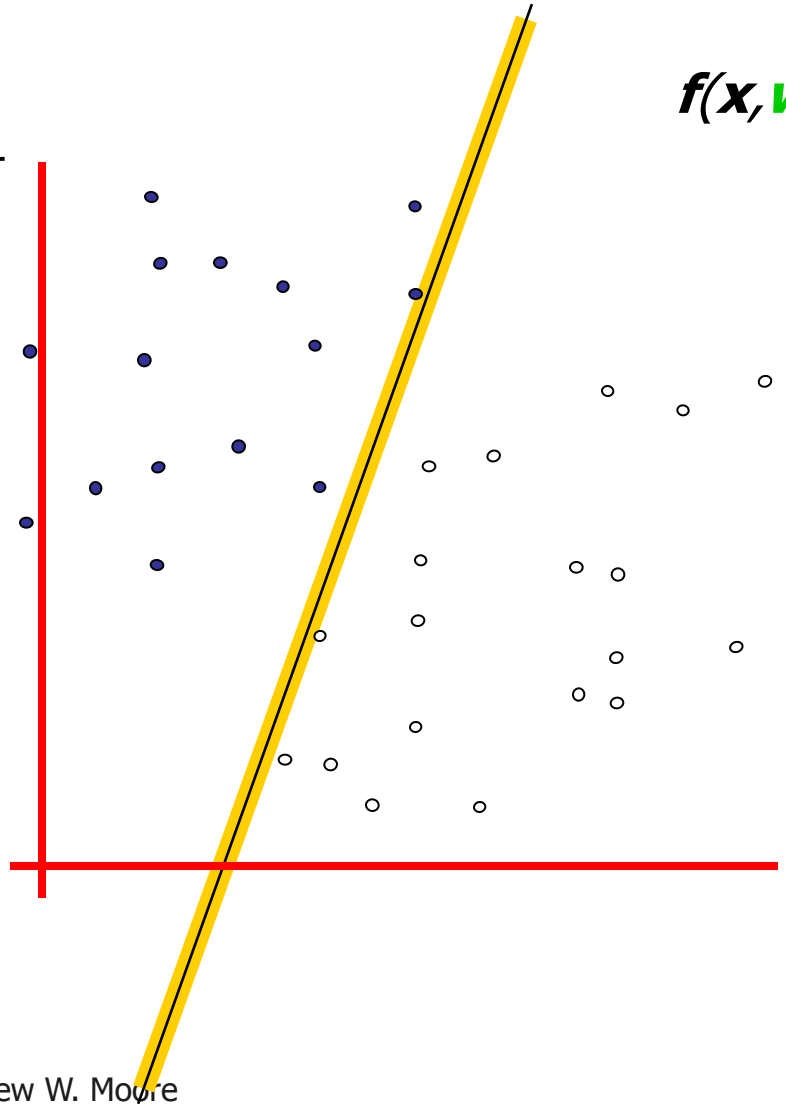
..but which is best?

Classifier Margin



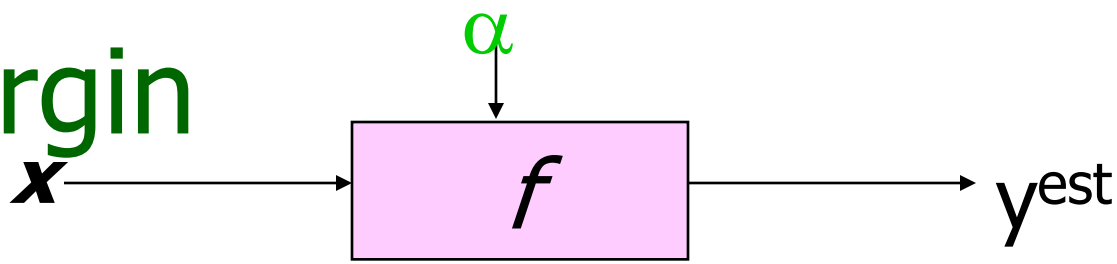
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} + b)$

- denotes +1
- denotes -1



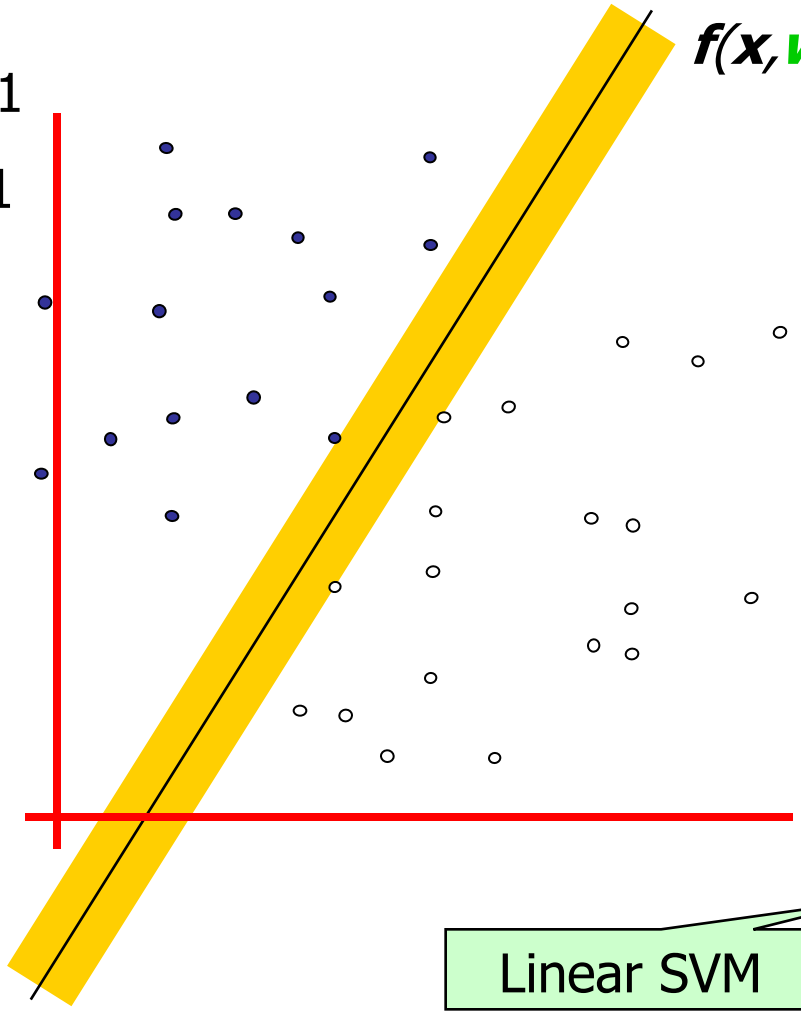
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

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 maximum margin.

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

Linear SVM

Training a linear SVM

- To find the maximum margin separator, we have to solve the following optimization problem:

$$\mathbf{w} \bullet \mathbf{x} + b > +1 \quad \textit{for positive cases}$$

$$\mathbf{w} \bullet \mathbf{x} + b < -1 \quad \textit{for negative cases}$$

$$\textit{and} \quad \| \mathbf{w} \|^2 \ \textit{is as small as possible}$$

- This is tricky but it's a convex problem. There is only one optimum and we can find it without fiddling with learning rates or weight decay or early stopping.
  - Don't worry about the optimization problem. It has been solved. It's called quadratic programming.
  - It takes time proportional to N^2 which is really bad for very big datasets
    - so for big datasets we end up doing approximate optimization!

Testing a linear SVM

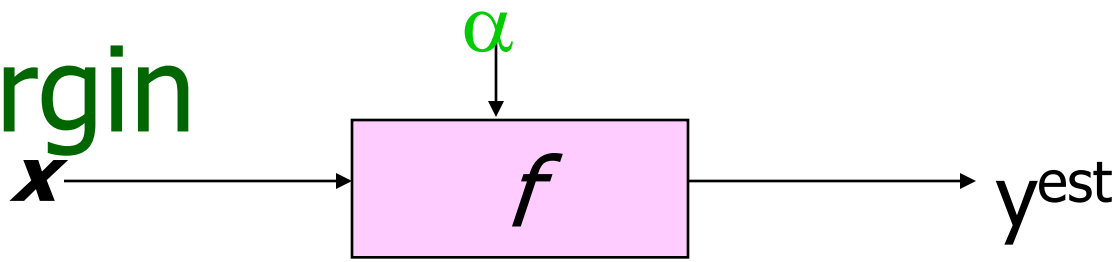
- The separator is defined as the set of points for which:

$$\mathbf{w} \bullet \mathbf{x} + b = 0$$

$$so\ if\ \ \mathbf{w} \bullet \mathbf{x} + b > 0\ \ say\ it's\ a\ positive\ case$$

$$and\ if\ \ \mathbf{w} \bullet \mathbf{x} + b < 0\ \ say\ it's\ a\ negative\ case$$

Maximum Margin



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

- denotes +1
- denotes -1

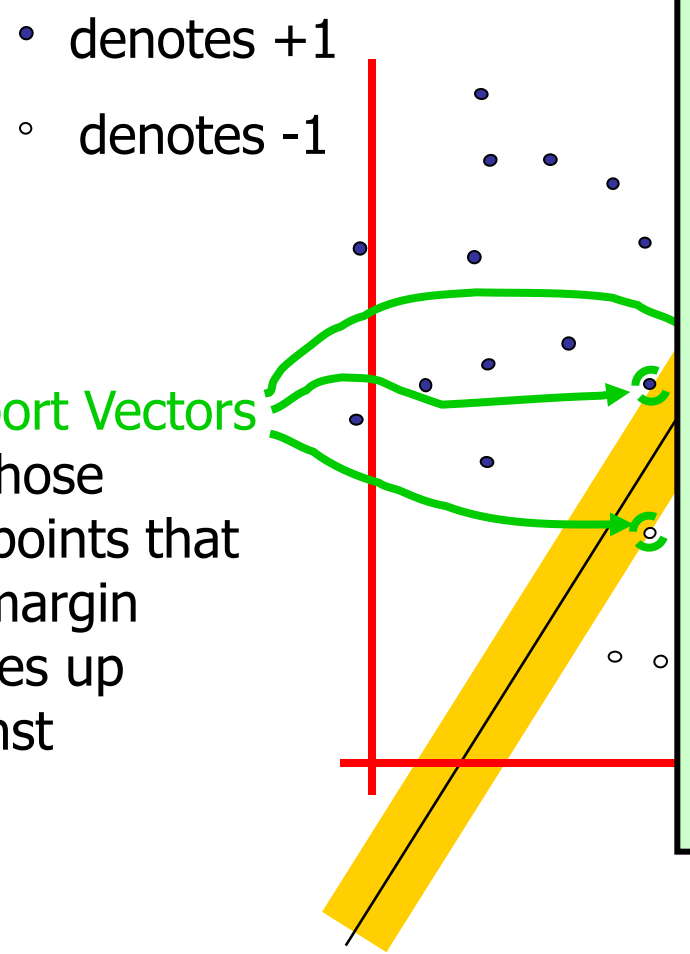
Support Vectors are those datapoints that the margin pushes up against

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

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

Linear SVM

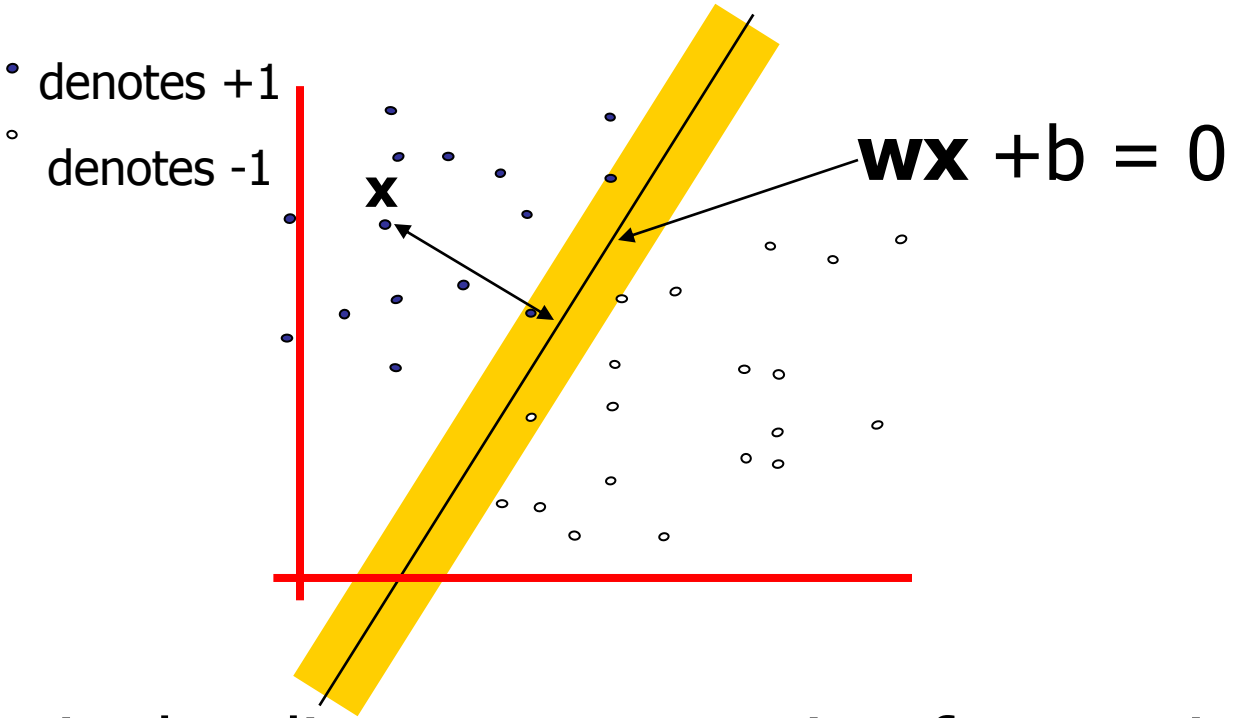
Why Maximum Margin?



1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary, this gives us least chance of causing a misclassification.
3. Robust 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.

LSVM)

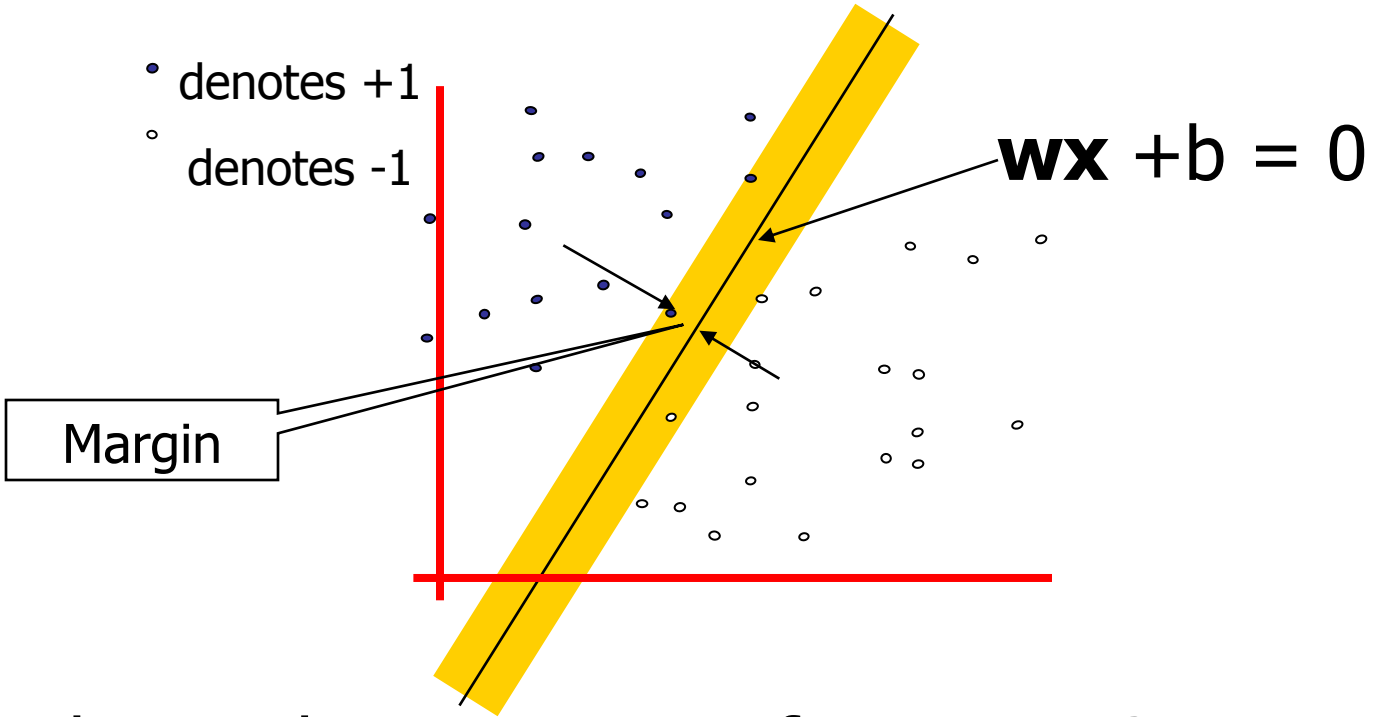
Estimate the Margin



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

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

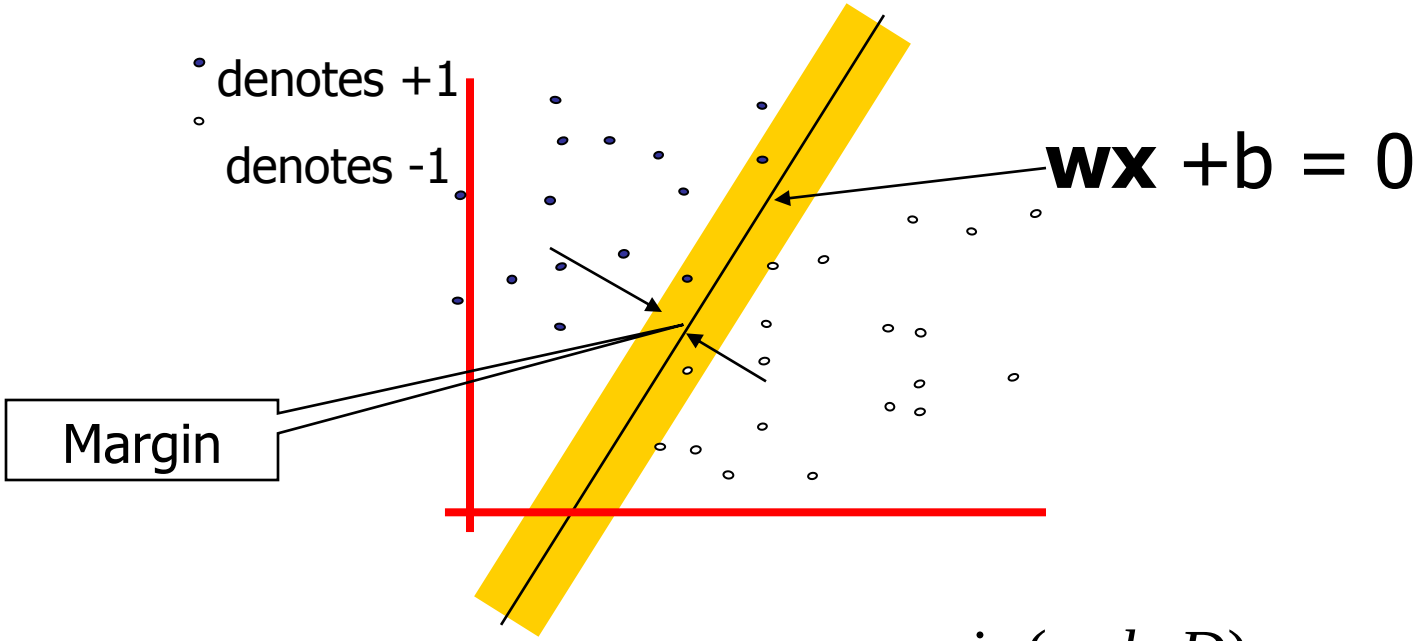
Estimate the Margin



- What is the expression for margin?

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

Maximize Margin

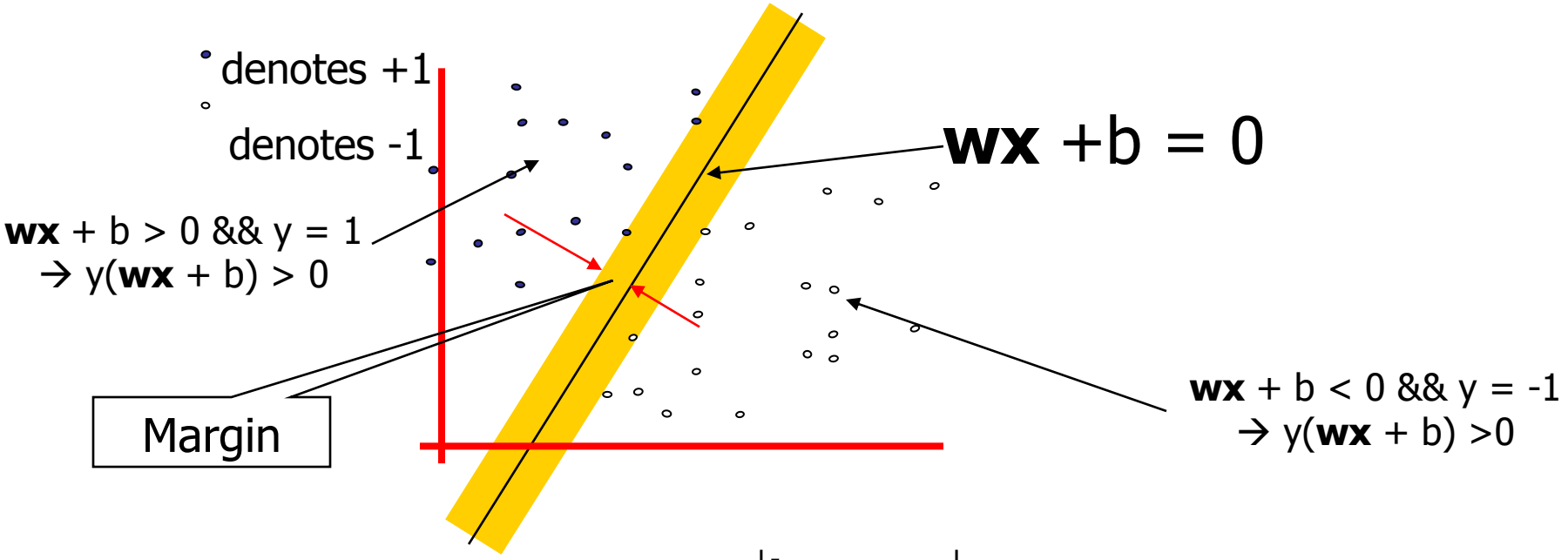


$$\underset{\mathbf{w},b}{\text{argmax}}\ \text{margin}(\mathbf{w},b,D)$$

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

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

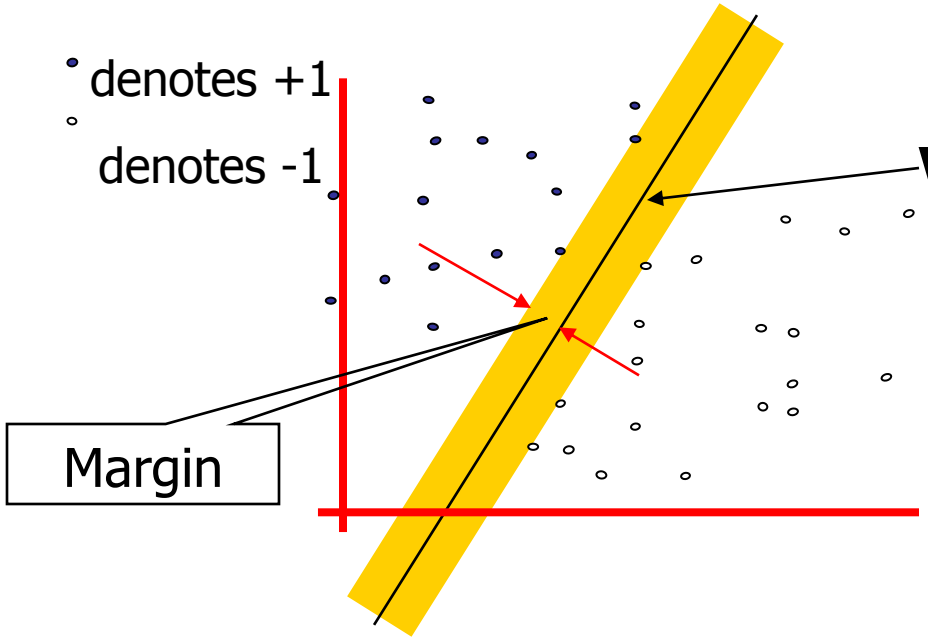
Maximize Margin



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

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

Maximize Margin



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

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

Strategy:

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

$$\underset{w,b}{\arg\max} \frac{1}{\sum_{i=1}^{d} w_i^2}$$

$$\text{Subject to} : \forall \mathbf{x}_i \in D : y_i(\mathbf{w} * \mathbf{x}_i + b) \geq 1$$

Maximum Margin Linear Classifier

$$\{w^*, b^*\} = \arg\min_{w,b} \quad \sum_{i=1}^{d} w_i^2$$

subject to:

$$y_1(w * x_1 + b) \geq 1$$
$$y_2(w * x_2 + b) \geq 1$$
$$\ldots$$
$$y_N(w * x_N + b) \geq 1$$

- How to solve it?

Using Langrange multipliers $a_n \geq 0$ and Quadratic Programming

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

Subject to

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

n additional linear inequality constraints

And subject to

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

e additional linear equality constraints

Quadratic Programming

Find $\arg\min_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ← 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)

…nal linear …equality …nstraints

$a_{(n+1)2}$ … $b_{(n+2)}$

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

e additional linear equality constraints

How to Solve Maximum Margin Linear Classifier – the Dual Form

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\sum_{i=1}^{d} w_i^2 - \sum_{n=1}^{N} a_n[y_n(w * x_n + b) - 1]$$

$$\mathbf{w} = \sum_{n=1}^{N} a_n y_n x_n$$

$$\sum_{n=1}^{N} a_n y_n = 0$$

$$\tilde{L}(\mathbf{a}) = \sum_{i=1}^{d} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m Q_{nm} \text{ , where } Q_{nm} = y_n y_m (\mathbf{x}_n \bullet \mathbf{x}_m)$$

$$a_n \geq 0$$

$$\sum_{n=1}^{N} a_n y_n = 0$$

Dual representation of the maximum margin problem

Maximum Margin Linear Classifier

$$\{w^*, b^*\} = \arg\min_{w,b} \quad \sum_{i=1}^{d} w_i^2$$

subject to:

$$y_i(w * x_i + b) \geq 1$$

$$a_i \geq 0$$

$$a_i[y_i(w * x_i + b) - 1] = 0$$

QP: obtain a_i and then w and b and in the end classify new y(x) according to its sign

Linear, Hard-Margin SVM Formulation

- Find *w, b* that solves

$$\min \frac{1}{2}\|w\|^2$$

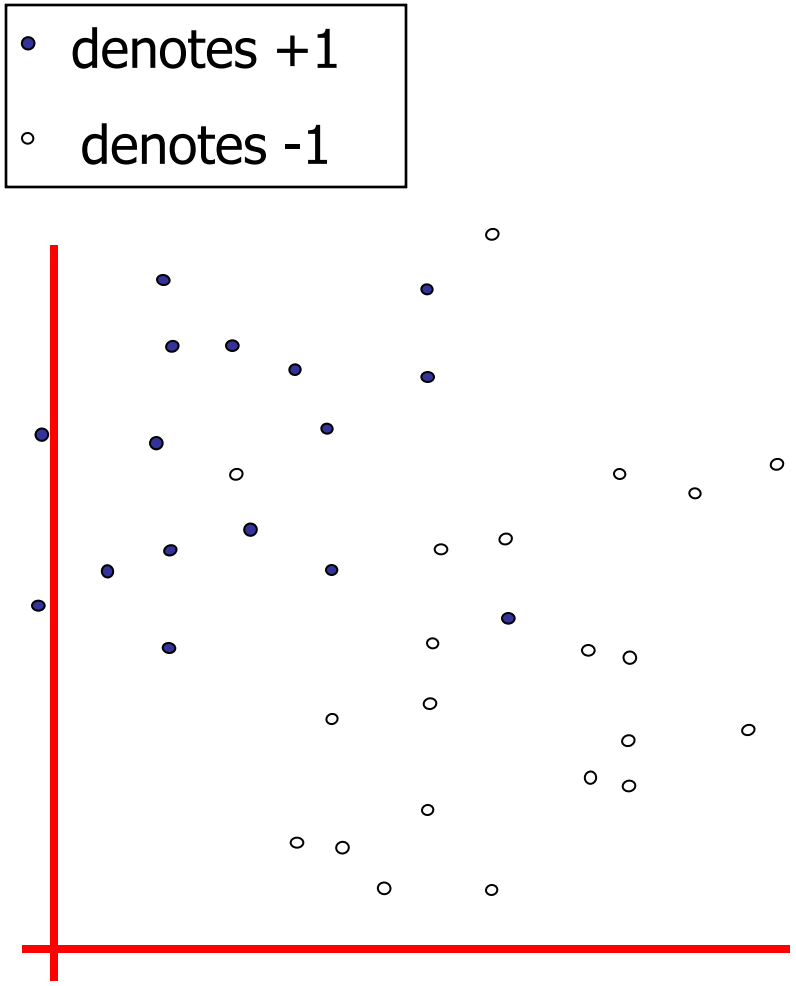
$$s.t.\ y_i(w \cdot x_i + b) \geq 1,\ \forall x_i$$

- Problem is convex, so there is a unique global minimum value (when feasible)
- There is also a unique minimizer, i.e. weight and *b* value that provides the minimum
- Non-solvable if the data is not linearly separable
- Quadratic Programming
  - Very efficient computationally with modern constraint optimization engines (handles thousands of constraints and training instances).

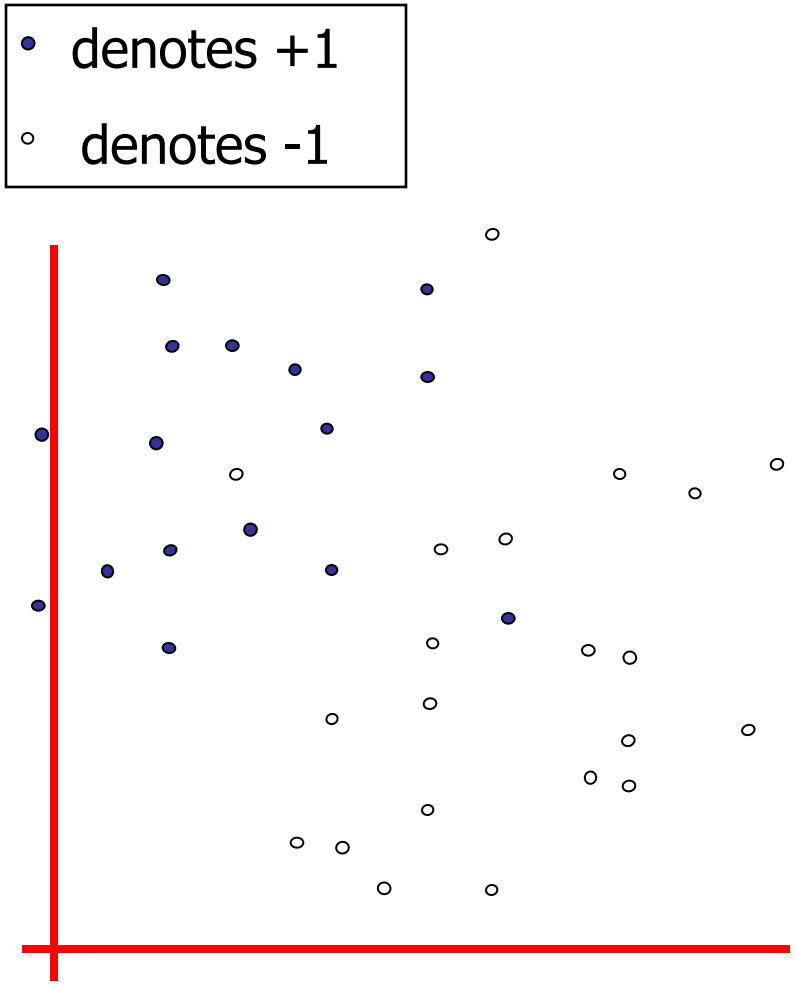
Uh-oh!

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!

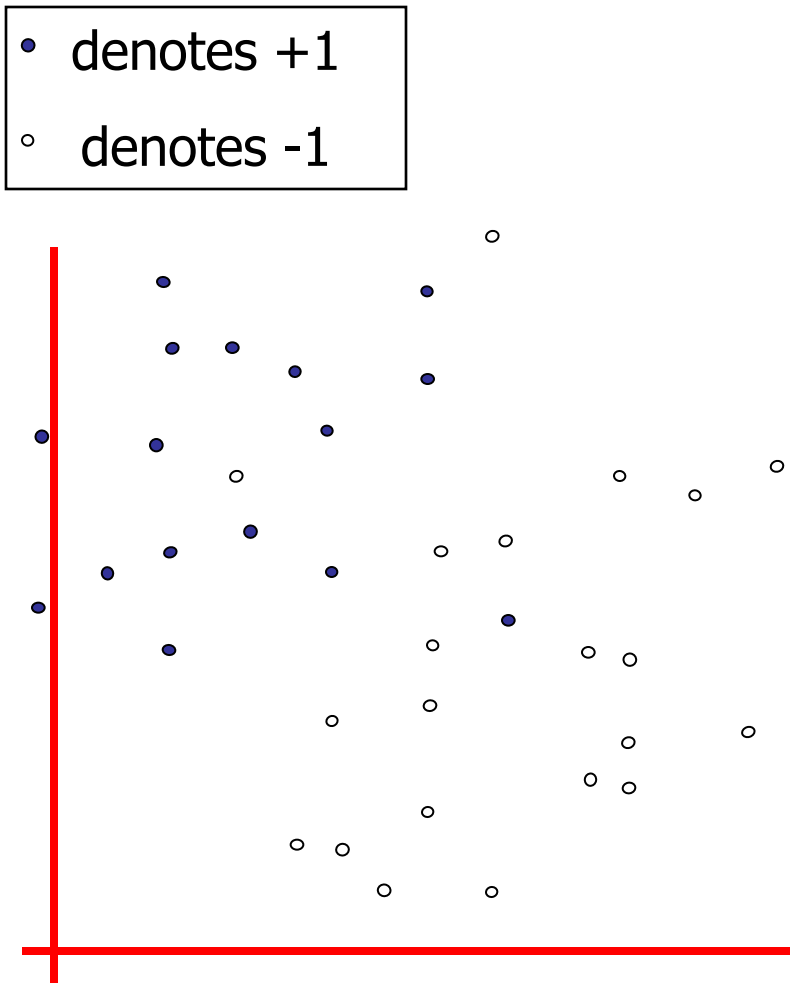
What should we do?

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!

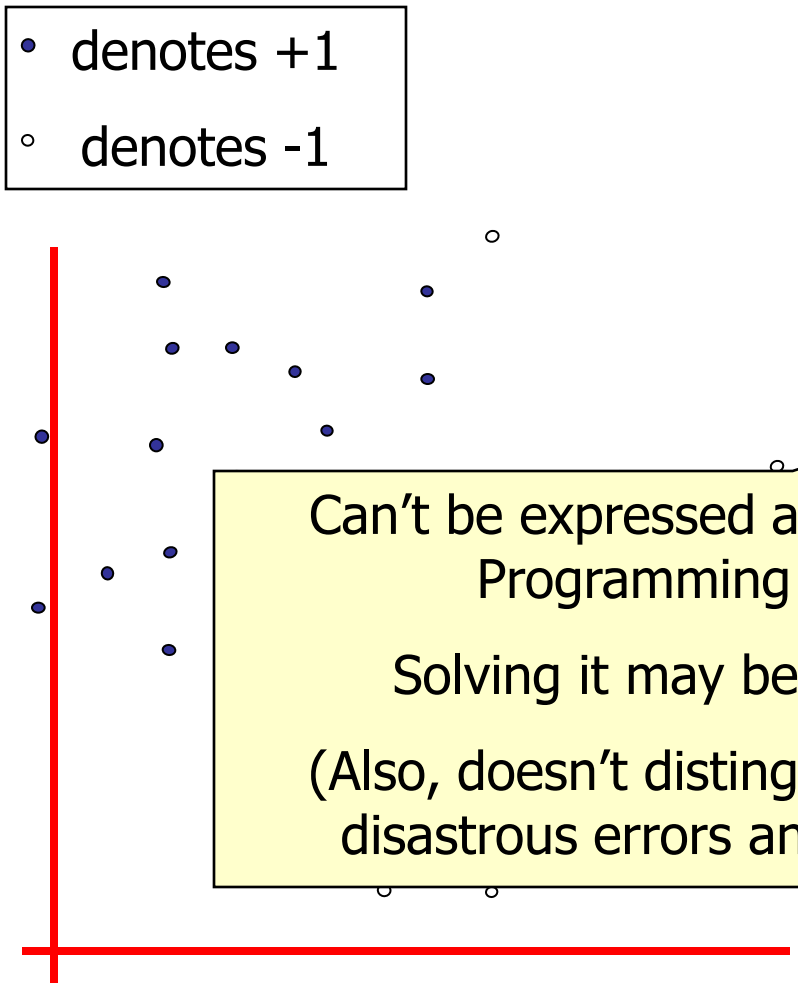
What should we do?

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!

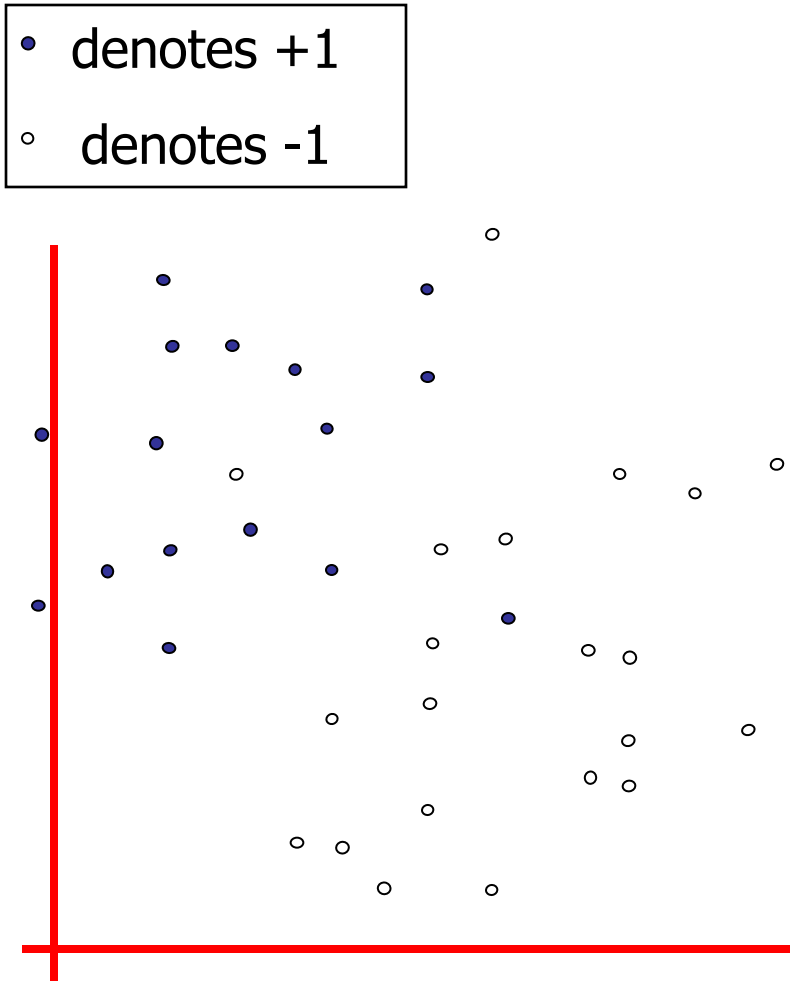
This is going to be a problem!

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

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

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

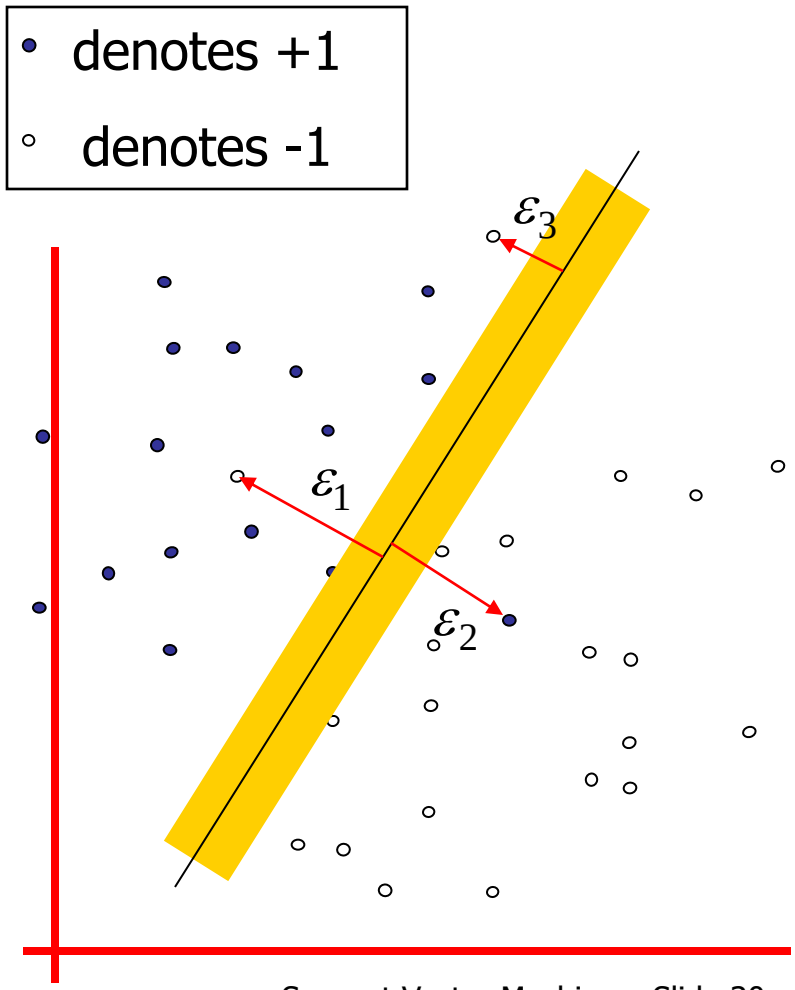
$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1 - \varepsilon_2$$

...

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

ξ_i are called "slack" variables

- Any problem with the above formulism?



Support Vector Machine (SVM) for Noisy Data

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

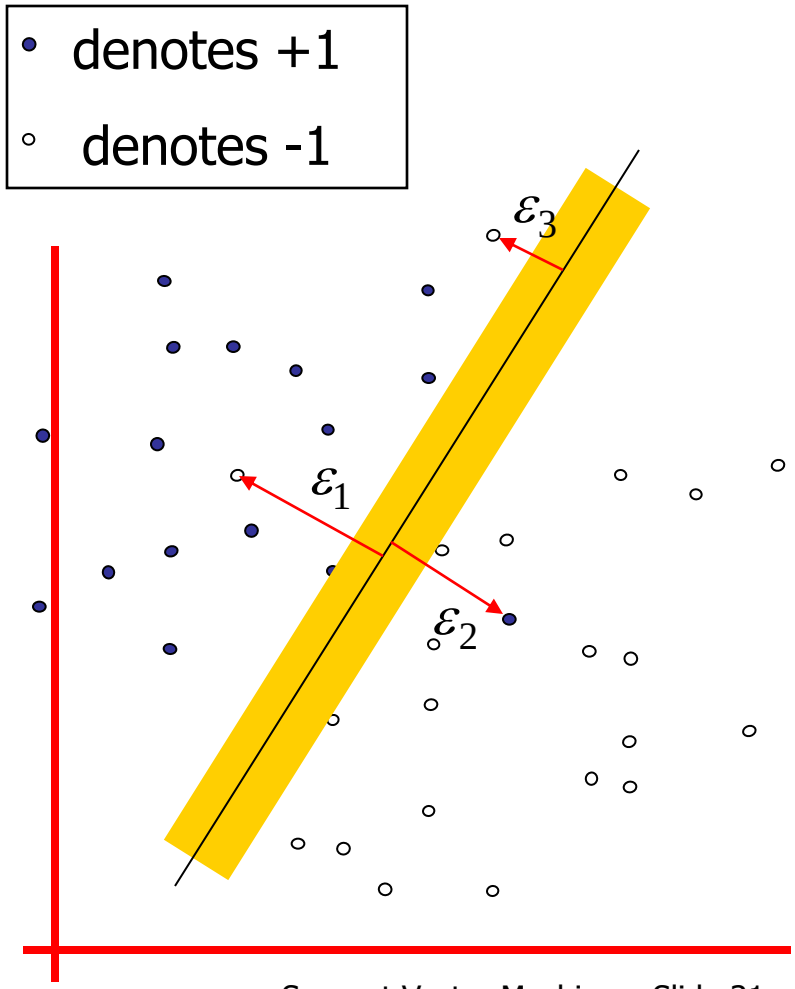
$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1-\varepsilon_1, \varepsilon_1 \geq 0$$

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

...

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

- Balance the tradeoff between margin and classification errors



SVM for Noisy Data (Linear, Soft-Margin SVMs)

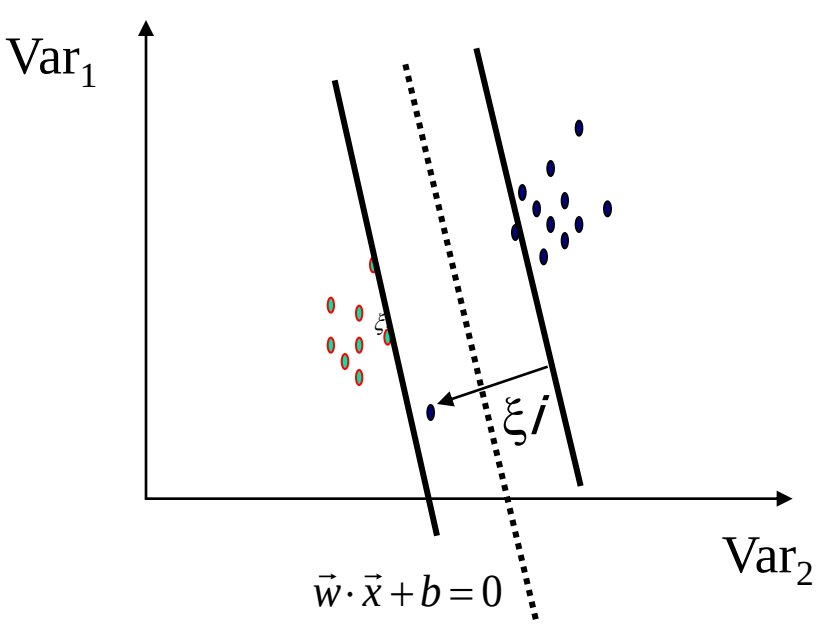
$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \ \forall x_i$$

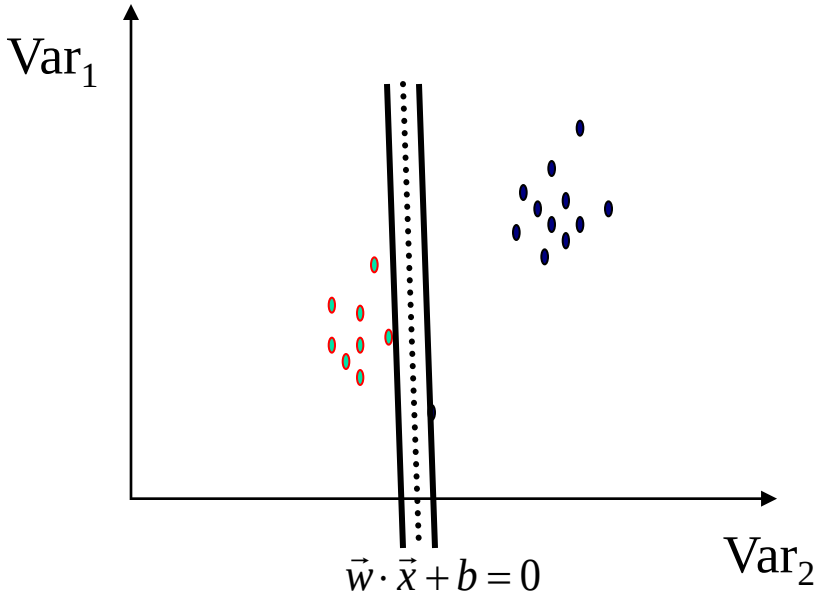
$$\xi_i \geq 0$$

- Algorithm tries to maintain ξ_i to zero while maximizing margin
- Notice: algorithm does not minimize the *number* of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes
- Other formulations use ξ_i^2 instead
- As $C \to \infty$, we get closer to the hard-margin solution
- How do we determine the appropriate value for c ?

Robustness of Soft vs Hard Margin SVMs



Soft Margin SVN



Hard Margin SVN

Soft-Margin always have a solution

Soft-Margin is more robust to outliers

Smoother surfaces (in the non-linear case)

Hard-Margin does not require to guess the cost parameter (requires no parameters at all)

SVM for Noisy Data

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

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

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

The Dual Form of QP

Maximize $\sum_{n=1}^{R} \alpha_n - \frac{1}{2} \sum_{n=1}^{R} \sum_{m=1}^{R} \alpha_n \alpha_m Q_{nm}$ where $Q_{nm} = y_n y_m (\mathbf{x}_n \bullet \mathbf{x}_m)$

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:

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

An Equivalent QP

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

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

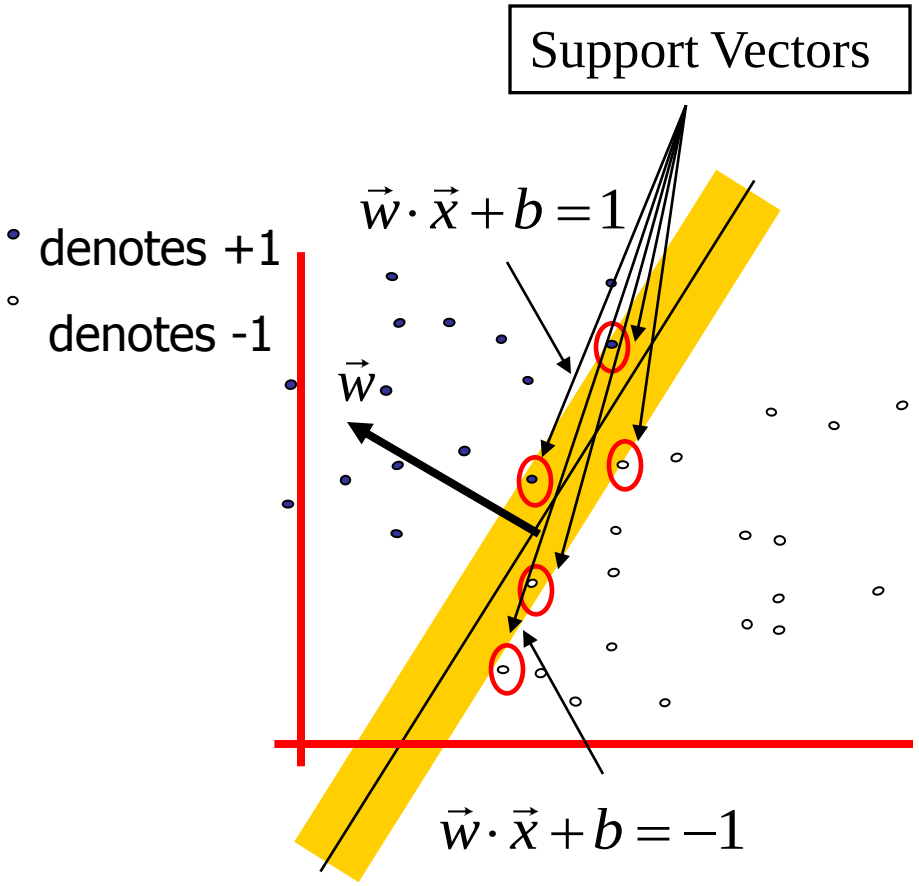
Then define:

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

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

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

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

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

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

Then define:

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

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

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

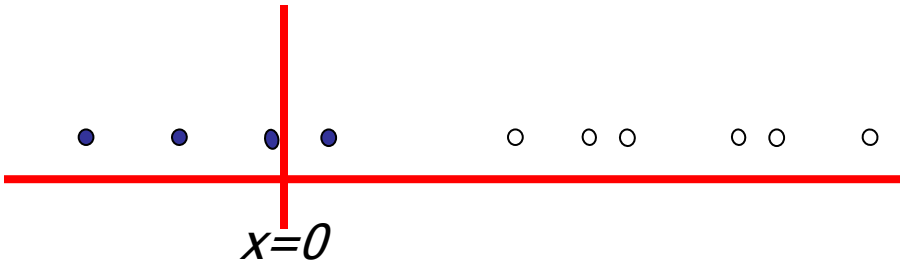
Then classify with:

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

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

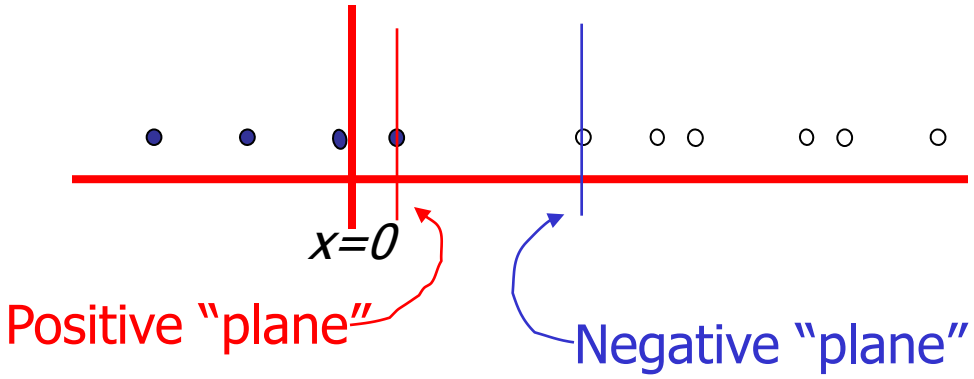
Suppose we're in 1-dimension

What would SVMs do with this data?



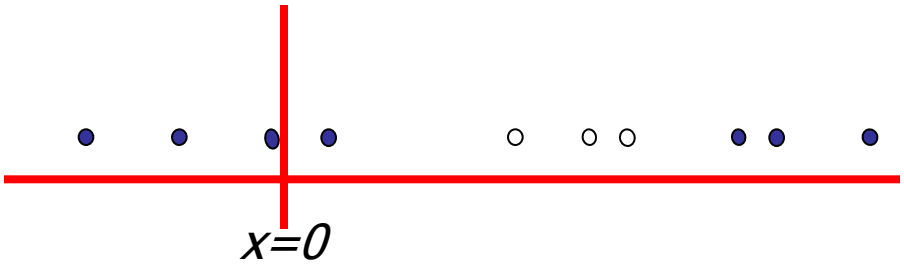
Suppose we're in 1-dimension

Not a big surprise

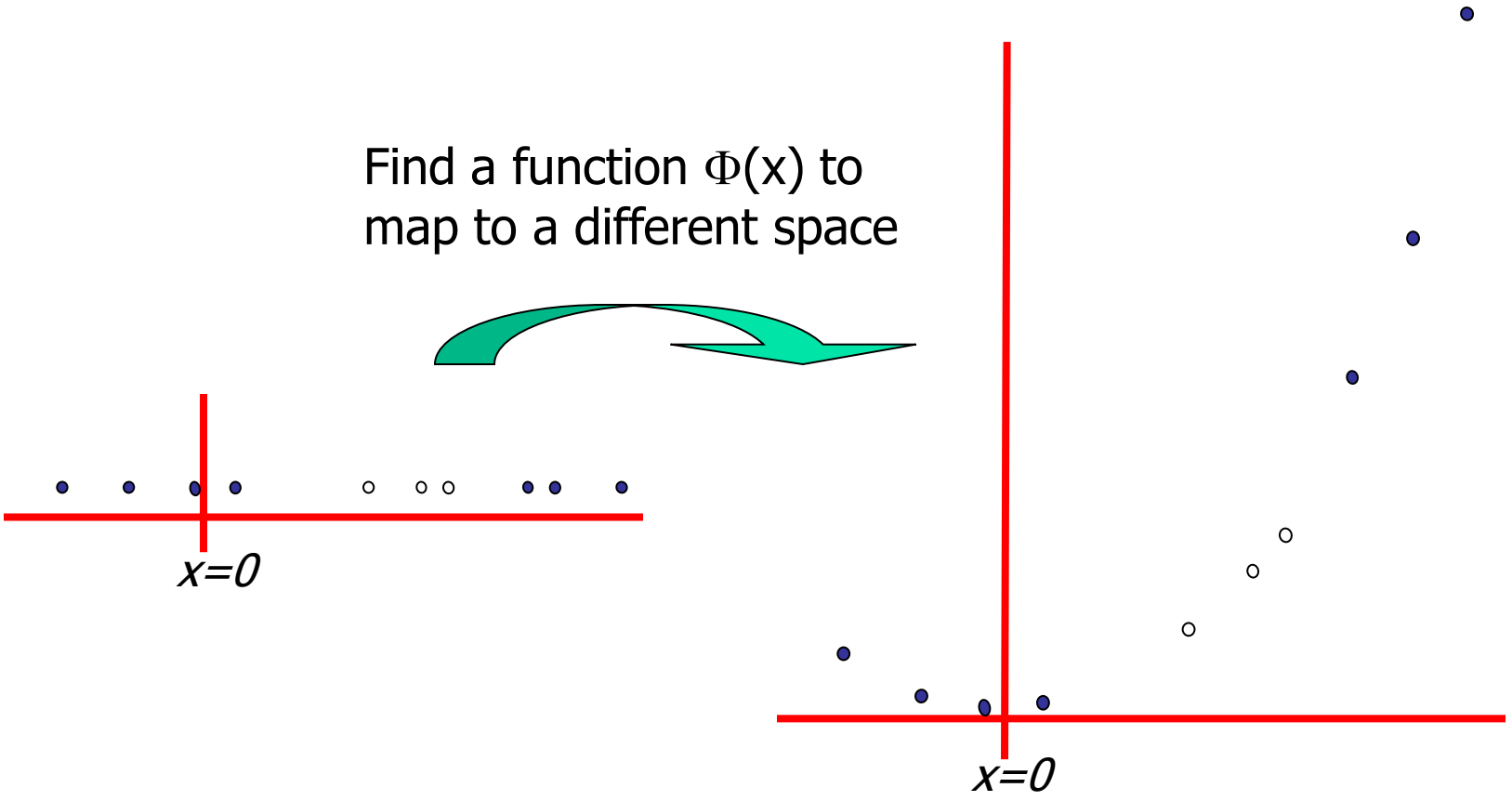


Harder 1-dimensional dataset

What can be done about this?



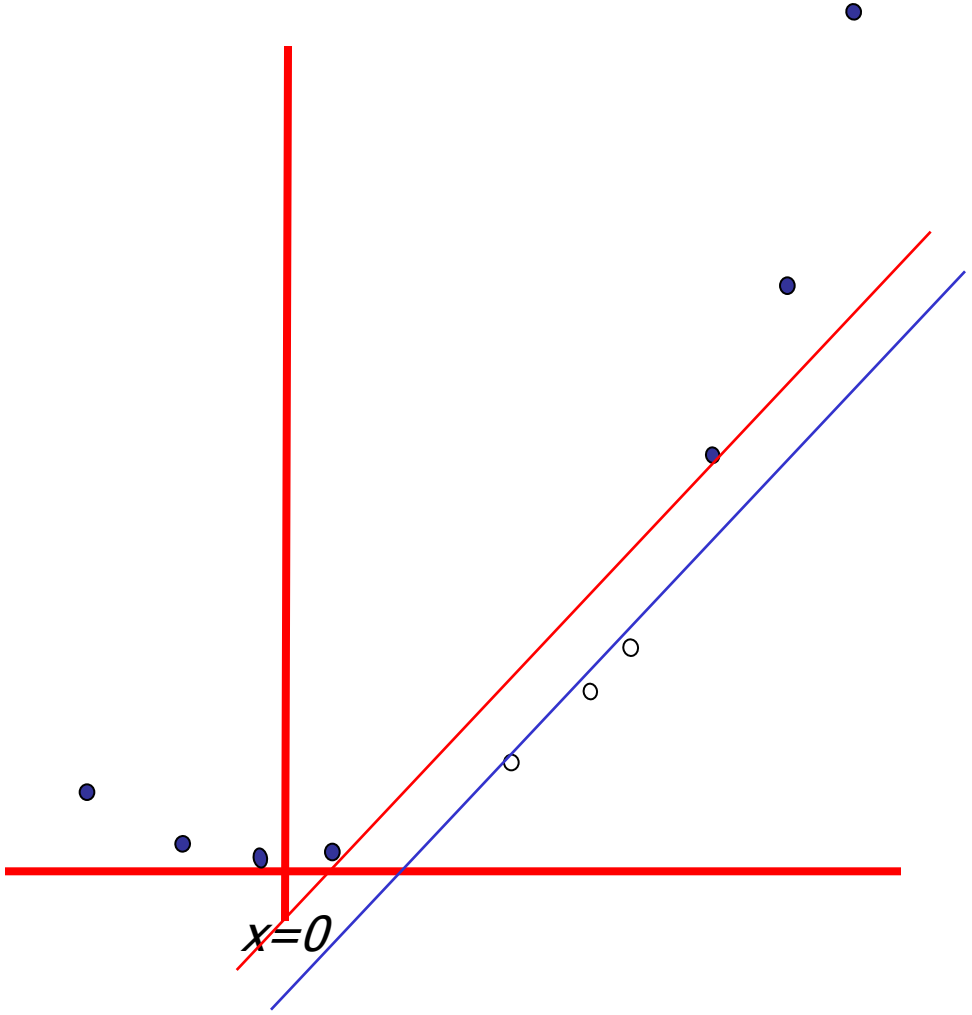
Harder 1-dimensional dataset



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

Let's permit them here too: $\Phi(x_k) = (x_k, x_k^2)$

Harder 1-dimensional dataset



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

Let's permit them here too

$$\Phi(x_k) = (x_k, x_k^2)$$

Common SVM basis functions

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

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

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

$\Phi(x_k)$ = (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)(m+1)/2

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

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:

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

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

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

Then classify with:

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

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

Then define:

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

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

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

Then classify with:

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

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

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:

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

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

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

f(x,w,b) = sign(w.* ϕ*(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 $\sum_{k=1}^{R}\alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R}\alpha_k\alpha_l Q_{kl}$ 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 now only requires *m* additions and multiplications

Then define:

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

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

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

Then classify with:

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

Higher Order Polynomials

Poly-nomial	$\phi(\boldsymbol{x})$	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

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?*

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

constraints:

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

Then define:

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

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

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

Then classify with:

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

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?

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

constrai[illegible]s.

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

Then [illegible]

- 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 classify with:

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

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

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. What are they?

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

The use of Maximum Margin magically makes this not a problem. (Not always!)

$$\alpha_k y_k = 0$$

Then

- 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 classify with:

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

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

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 What are they?

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

The use of Maximum Margin magically makes this not a problem

$$\alpha_k y_k = 0$$

Then classify with:

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

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

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

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

QP with Quintic basis functions

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

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

Then define:

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

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

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

Then classify with:

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

QP with 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:

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

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

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

Then classify with:

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

Most important change:

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

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
    - $K(a,b) = \phi(a) \cdot \phi(b)$
  - 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

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

Kernel Tricks

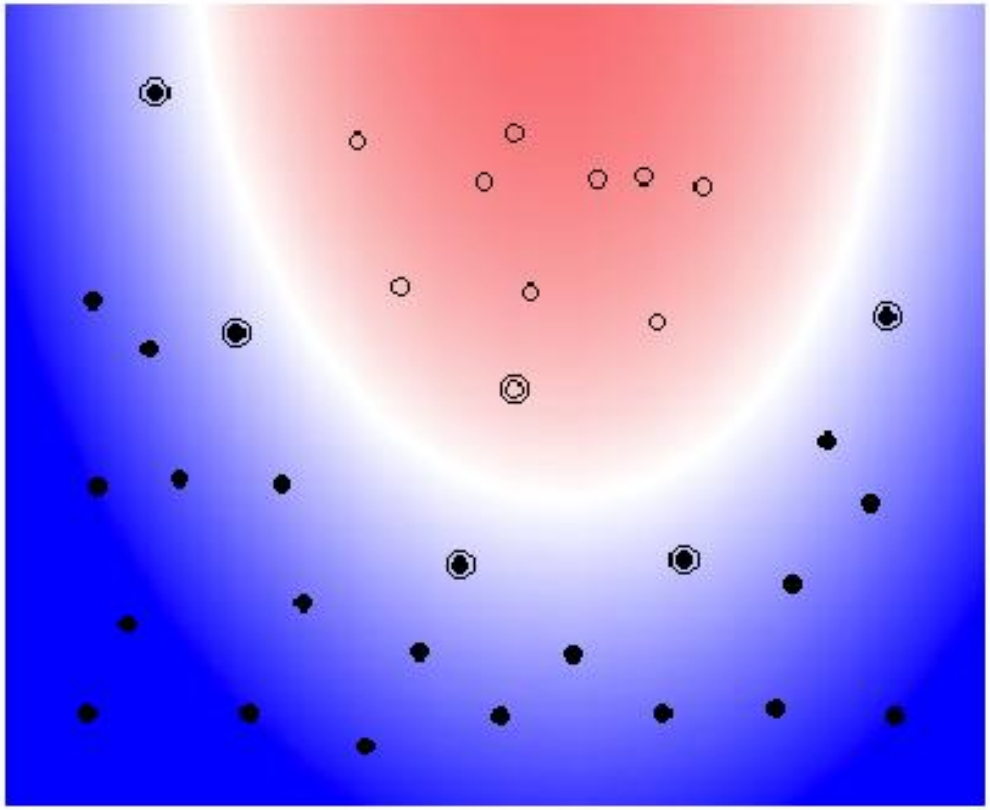
- Pro
  - Introducing nonlinearity into the model
  - Computational cheap
- Con
  - Still have potential overfitting problems

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

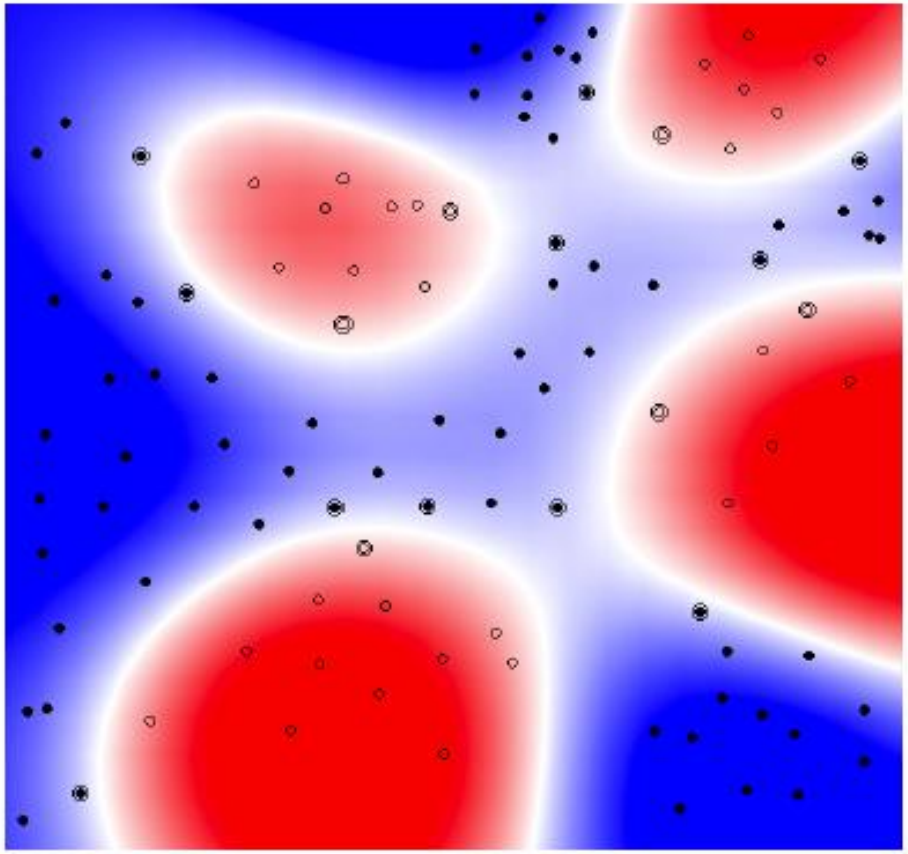


Nonlinear Kernel (II)

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



Doing multi-class classification

- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?

- One-versus-all
  - Train *n* binary classifiers, one for each class against all other classes. (SVM 1 learns "Output==1" vs "Output != 1")
  - Predicted class is the class of the most confident classifier (the furthest into the positive region)

- One-versus-one
  - Train *n(n-1)/2* classifiers, each discriminating between a pair of classes. (SVM 1 learns "Output==1" vs "Output = 2")
  - Several strategies for selecting the final classification based on the output of the binary SVMs

- Truly MultiClass SVMs
  - Generalize the SVM formulation to multiple categories

Other Types of Kernel Methods

- SVMs that perform regression
- SVMs that perform clustering
- ν-Support Vector Machines: maximize margin while bounding the number of margin errors
- Leave One Out Machines: minimize the bound of the leave-one-out error
- SVM formulations that take into consideration difference in cost of misclassification for the different classes
- Kernels suitable for sequences of strings, or other specialized kernels

Variable Selection with SVMs

- Recursive Feature Elimination
  - Train a linear SVM
  - Remove the variables with the lowest weights (those variables affect classification the least), e.g., remove the lowest 50% of variables
  - Retrain the SVM with remaining variables and repeat until classification is reduced
- Very successful
- Other formulations exist where minimizing the number of variables is folded into the optimization problem
- Similar algorithm exist for non-linear SVMs
- Some of the best and most efficient variable selection methods

Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters, but the rest is automatic.
  - The test performance is very good.
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the square of the number of training cases.
  - We need to store all the support vectors.
- SVM's are very good if you have no idea about what structure to impose on the task.
- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space.

Support Vector Machines are Perceptrons!

- SVM's use each training case, x, to define a feature K(x, .) where K is chosen by the user.
  - So the user designs the features.
- Then they do "feature selection" by picking the support vectors, and they learn how to weight the features by solving a big optimization problem.
- An SVM is just a very clever way to train a standard perceptron.
  - All of the things that a perceptron cannot do cannot be done by SVM's.

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: SVM-light, http://svmlight.joachims.org/, free download