

Support Vector Machines and Kernel Methods

UNC-CH COMP 755

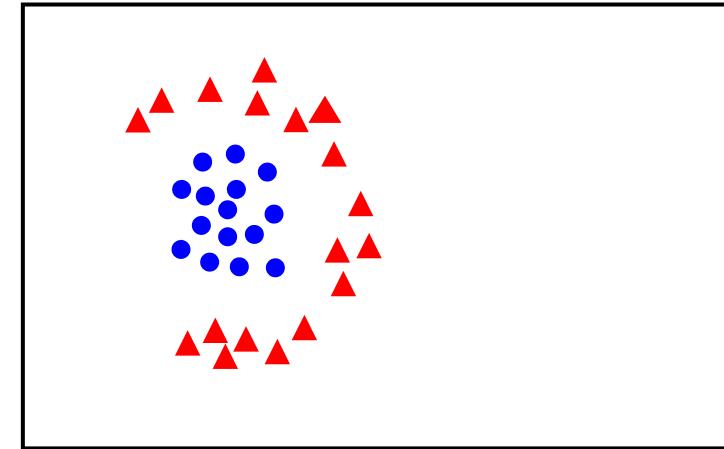
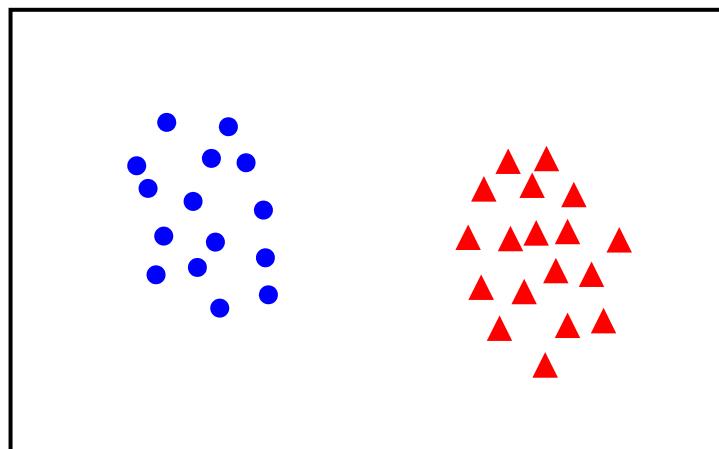
Slides adapted from: A. Zisserman, B. Schölkopf, D. Sontag,
L. Zettlemoyer, C. Guestrin, et al.

Binary Classification

Given training data (\mathbf{x}_i, y_i) for $i = 1 \dots N$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, learn a classifier $f(\mathbf{x})$ such that

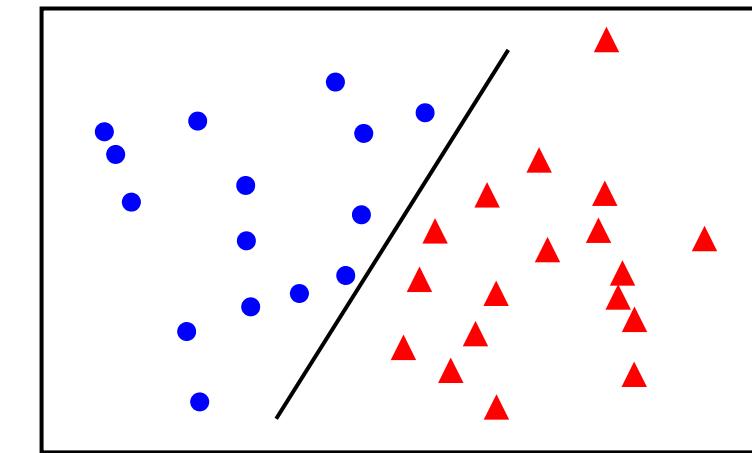
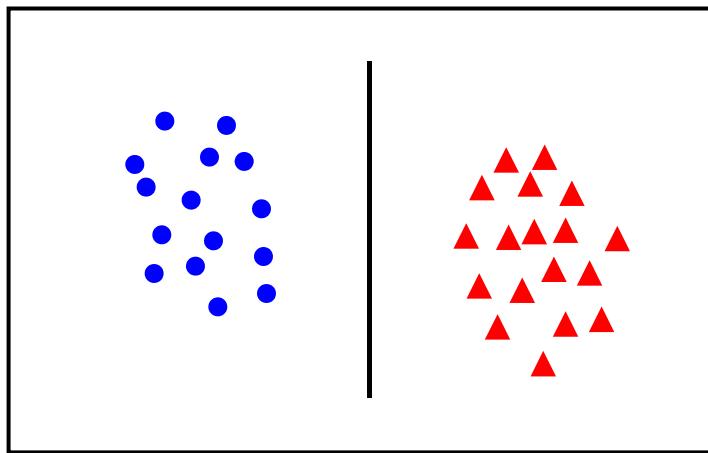
$$f(\mathbf{x}_i) \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases}$$

i.e. $y_i f(\mathbf{x}_i) > 0$ for a correct classification.

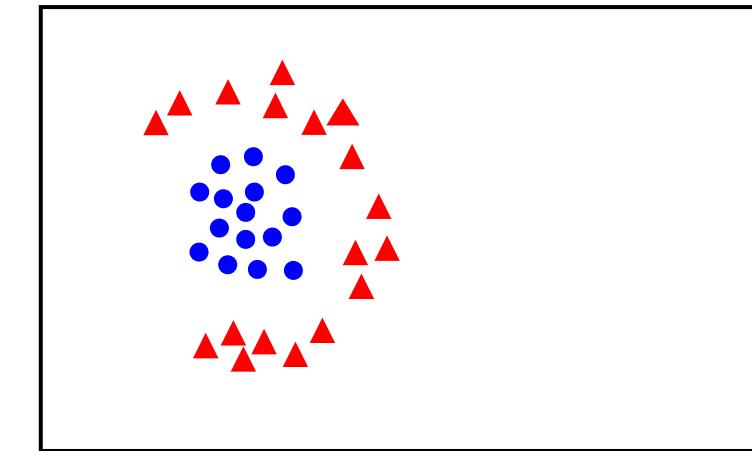
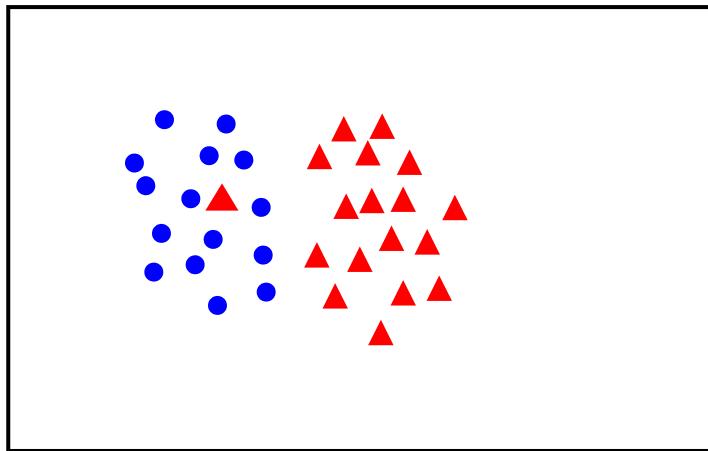


Linear separability

linearly
separable



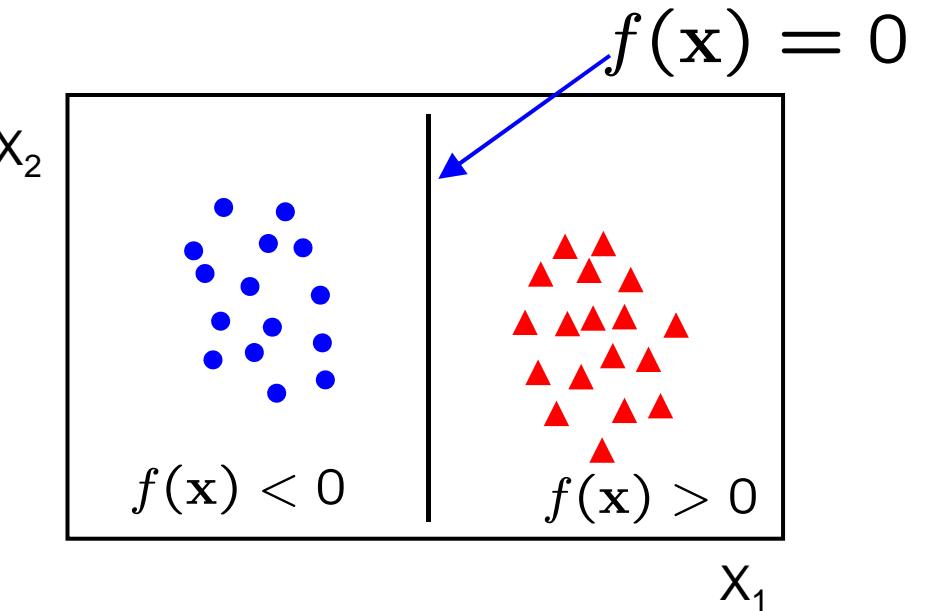
not
linearly
separable



Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

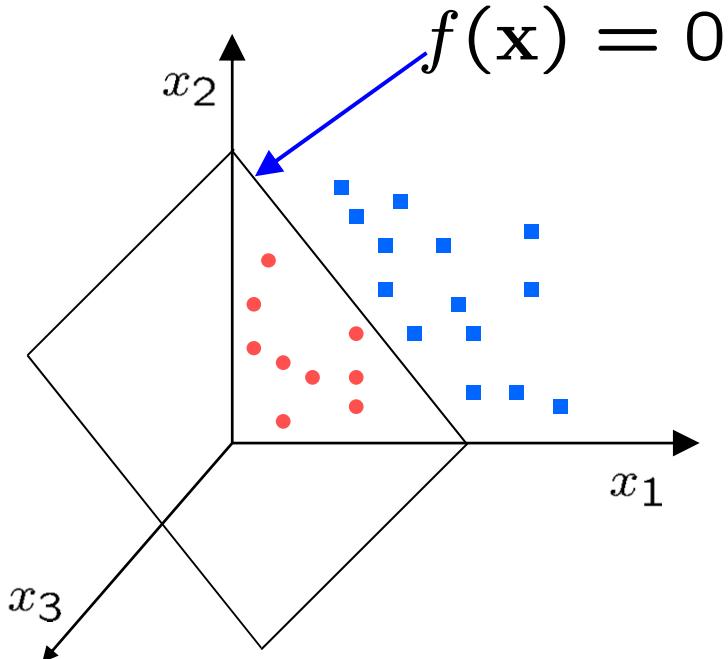


- in 2D the discriminant is a line
- \mathbf{w} is the **normal** to the line, and b the **bias**
- \mathbf{w} is known as the **weight vector**

Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 3D the discriminant is a plane, and in nD it is a hyperplane

For a K-NN classifier it was necessary to ‘carry’ the training data

For a linear classifier, the training data is used to learn \mathbf{w} and then discarded

Only \mathbf{w} is needed for classifying new data

The Perceptron Classifier

Given linearly separable data \mathbf{x}_i labelled into two categories $y_i = \{-1, 1\}$, find a weight vector \mathbf{w} such that the discriminant function

$$f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$

separates the categories for $i = 1, \dots, N$

- how can we find this separating hyperplane ?

The Perceptron Algorithm

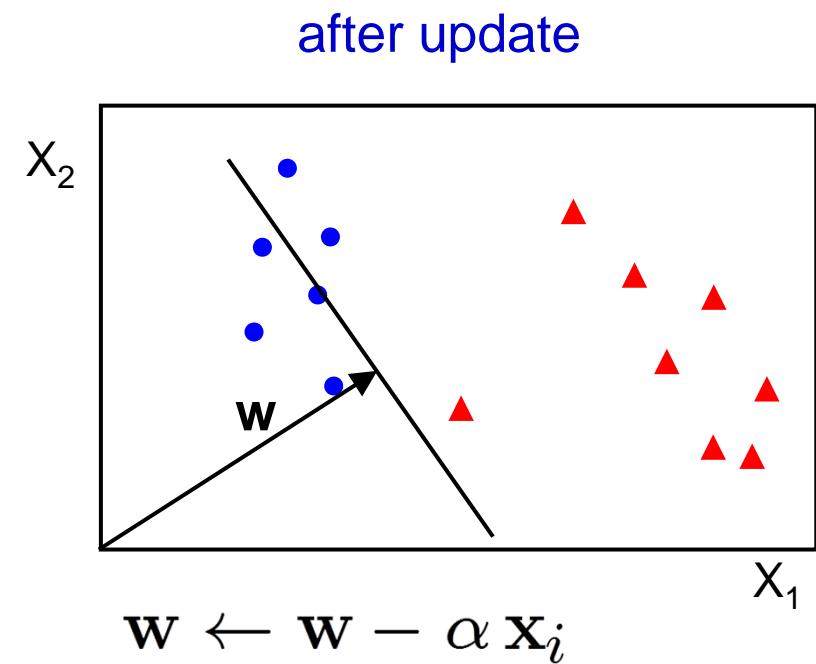
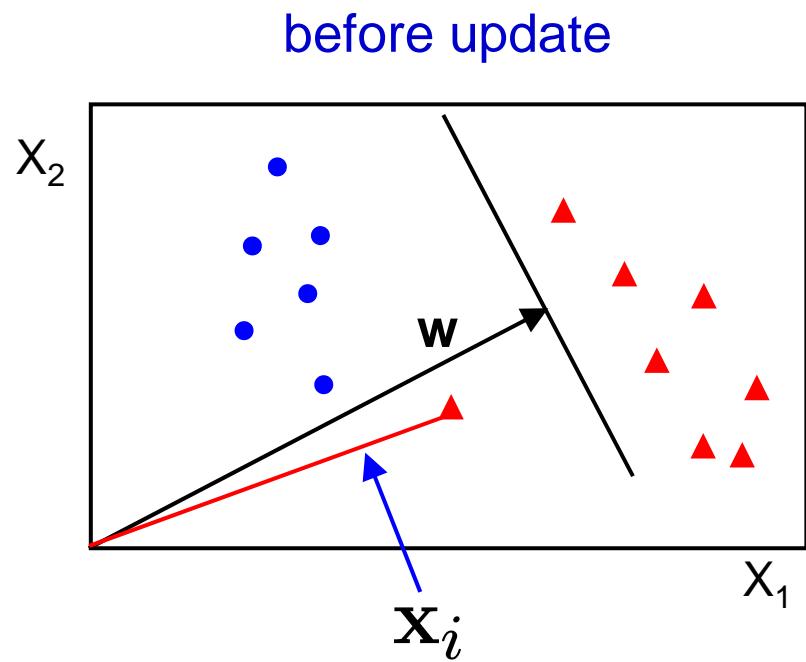
Write classifier as $f(\mathbf{x}_i) = \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i + w_0 = \mathbf{w}^\top \mathbf{x}_i$

where $\mathbf{w} = (\tilde{\mathbf{w}}, w_0)$, $\mathbf{x}_i = (\tilde{\mathbf{x}}_i, 1)$

- Initialize $\mathbf{w} = 0$
- Cycle though the data points $\{ \mathbf{x}_i, y_i \}$
 - if \mathbf{x}_i is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha \text{sign}(f(\mathbf{x}_i)) \mathbf{x}_i$
- Until all the data is correctly classified

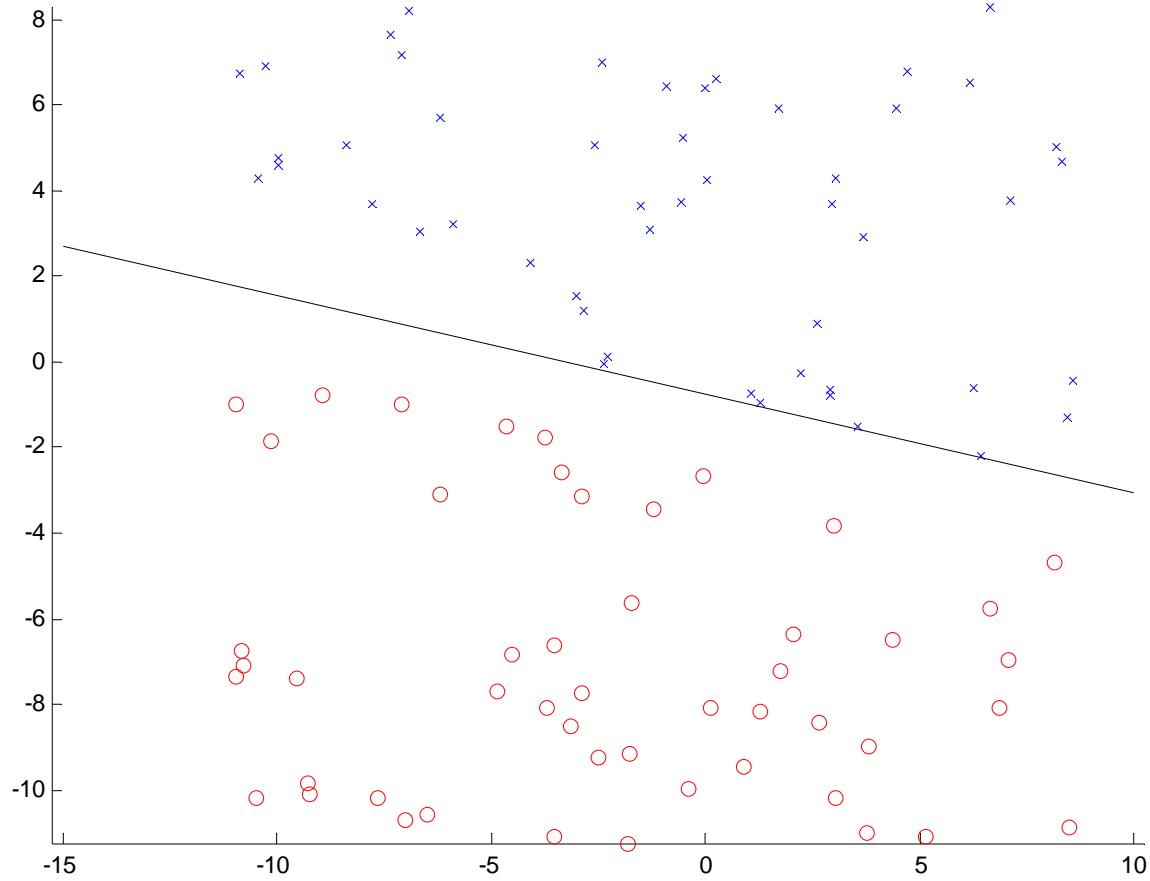
For example in 2D

- Initialize $\mathbf{w} = 0$
- Cycle though the data points $\{ \mathbf{x}_i, y_i \}$
 - if \mathbf{x}_i is misclassified then $\mathbf{w} \leftarrow \mathbf{w} + \alpha \text{sign}(f(\mathbf{x}_i)) \mathbf{x}_i$
- Until all the data is correctly classified



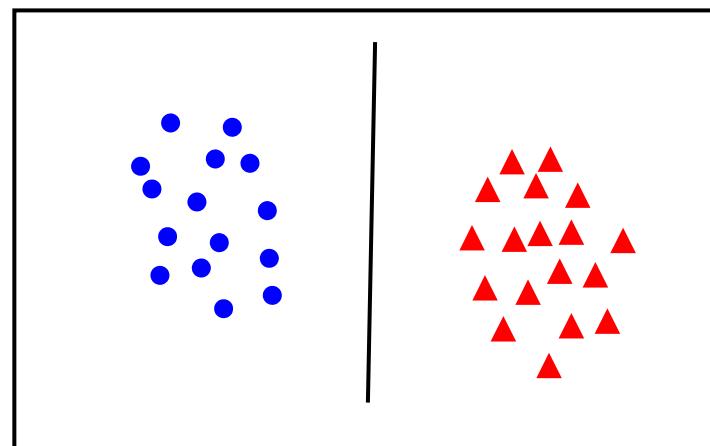
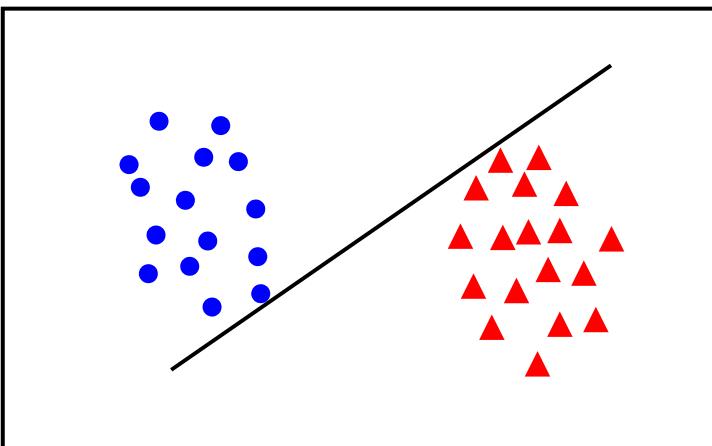
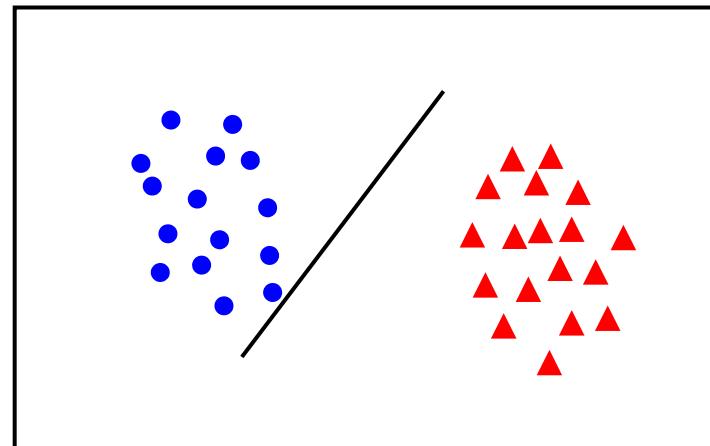
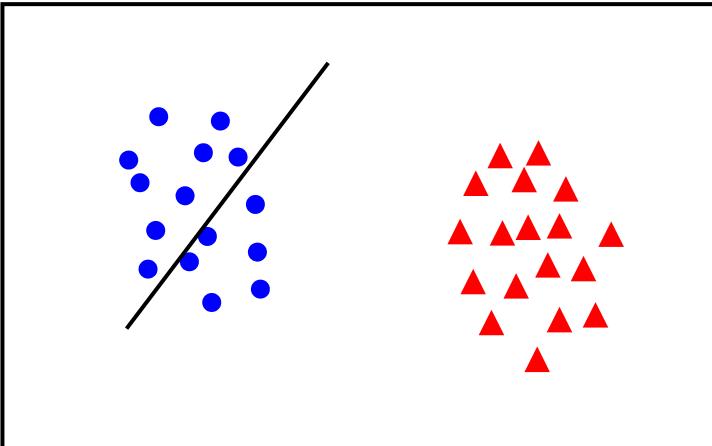
NB after convergence $\mathbf{w} = \sum_i^N \alpha_i \mathbf{x}_i$

Perceptron example



- if the data is linearly separable, then the algorithm will converge
- convergence can be slow ...
- separating line close to training data
- we would prefer a larger **margin** for generalization

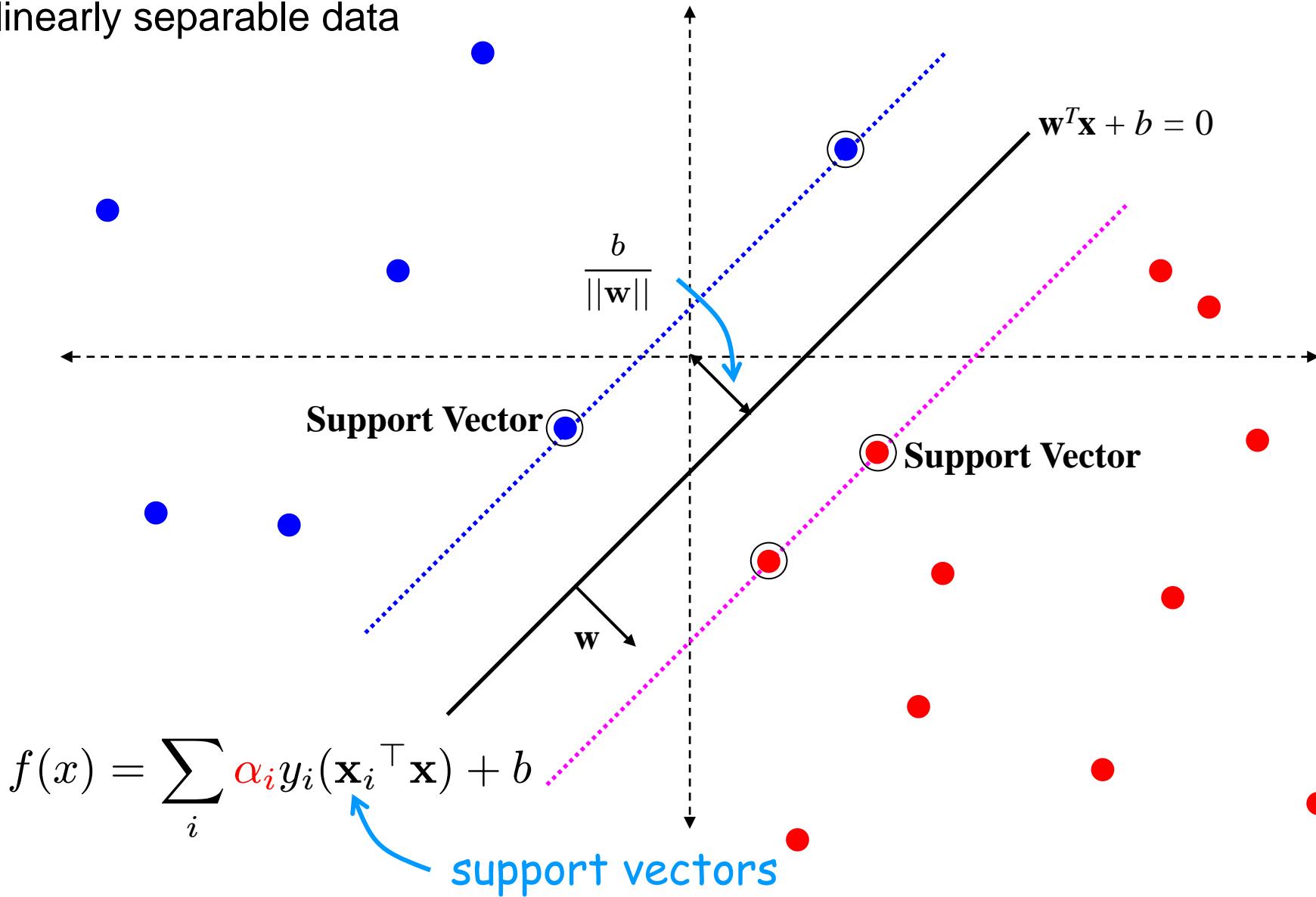
What is the best w ?



- maximum margin solution: most stable under perturbations of the inputs

Support Vector Machine

linearly separable data



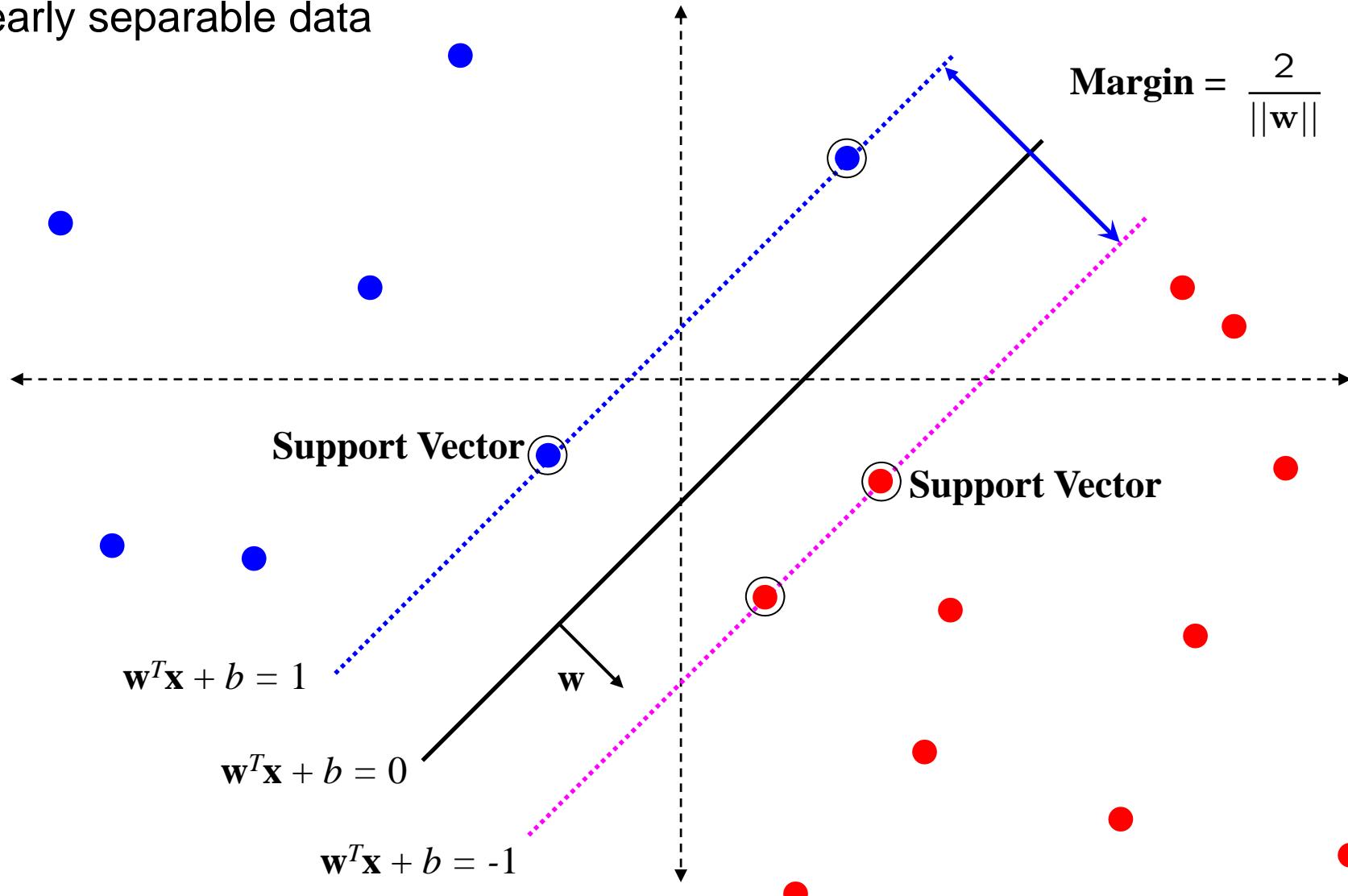
SVM – sketch derivation

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of \mathbf{w}
- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively
- Then the margin is given by

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = \frac{\mathbf{w}^\top (\mathbf{x}_+ - \mathbf{x}_-)}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

Support Vector Machine

linearly separable data



SVM – Optimization

- Learning the SVM can be formulated as an optimization:

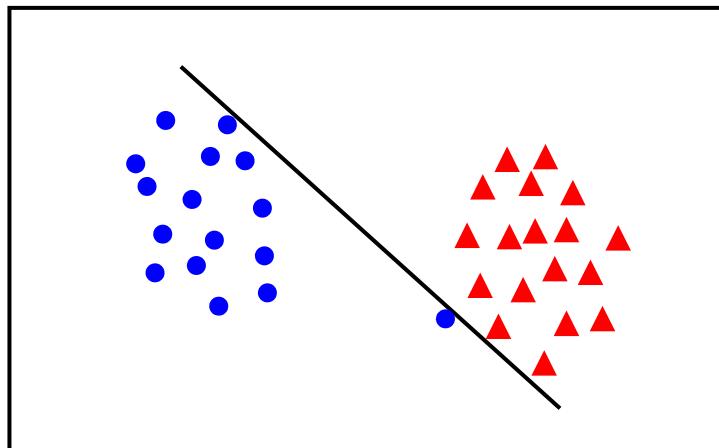
$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{cases} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \text{ for } i = 1 \dots N$$

- Or equivalently

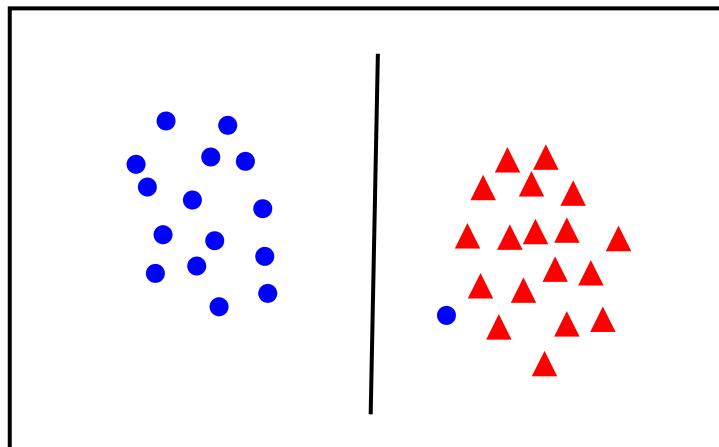
$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \text{ subject to } y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \text{ for } i = 1 \dots N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

Linear separability again: What is the best w?



- the points can be linearly separated but there is a very narrow margin



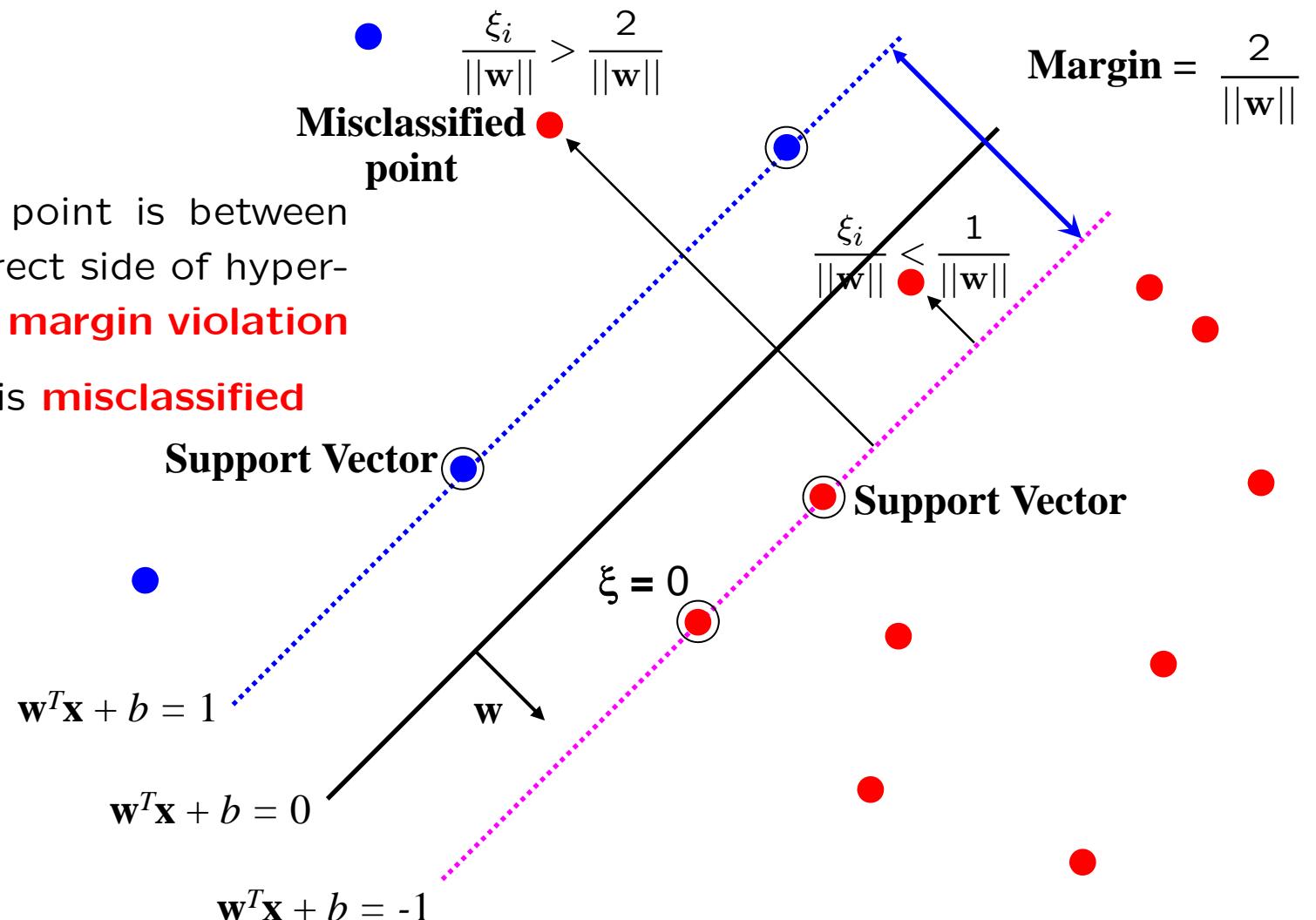
- but possibly the large margin solution is better, even though one constraint is violated

In general there is a trade off between the margin and the number of mistakes on the training data

Introduce “slack” variables

$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyperplane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



“Soft” margin solution

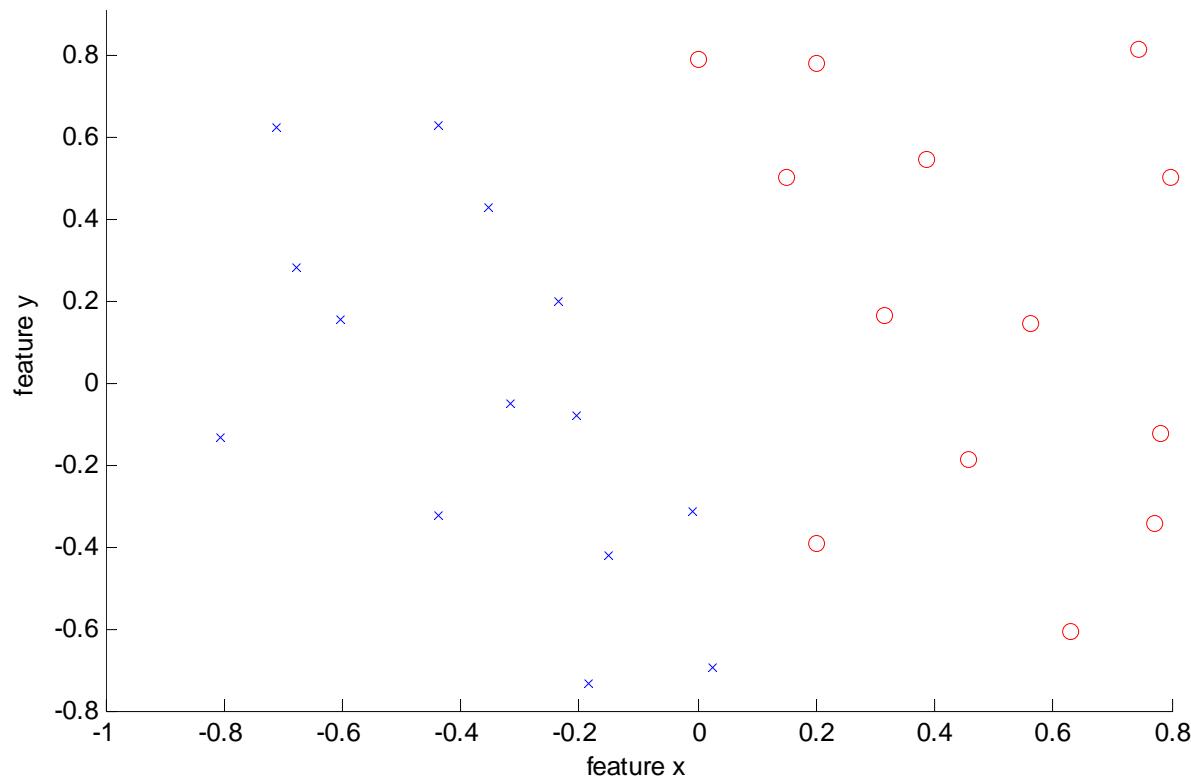
The optimization problem becomes

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i$$

subject to

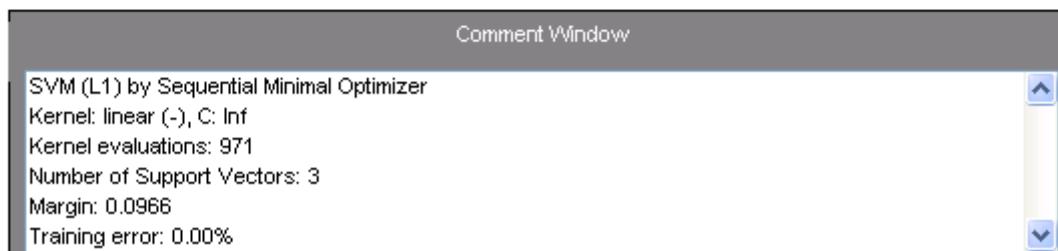
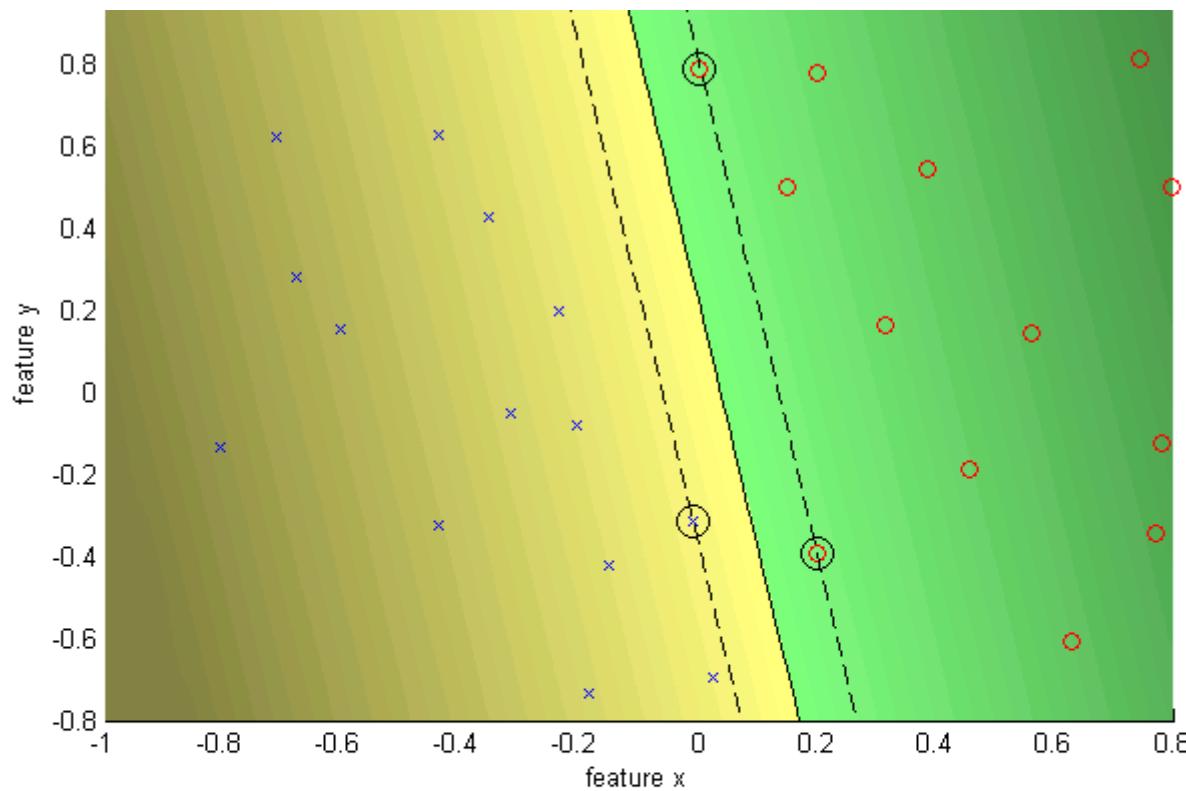
$$y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a regularization parameter:
 - small C allows constraints to be easily ignored \rightarrow large margin
 - large C makes constraints hard to ignore \rightarrow narrow margin
 - $C = \infty$ enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note, there is only one parameter, C .

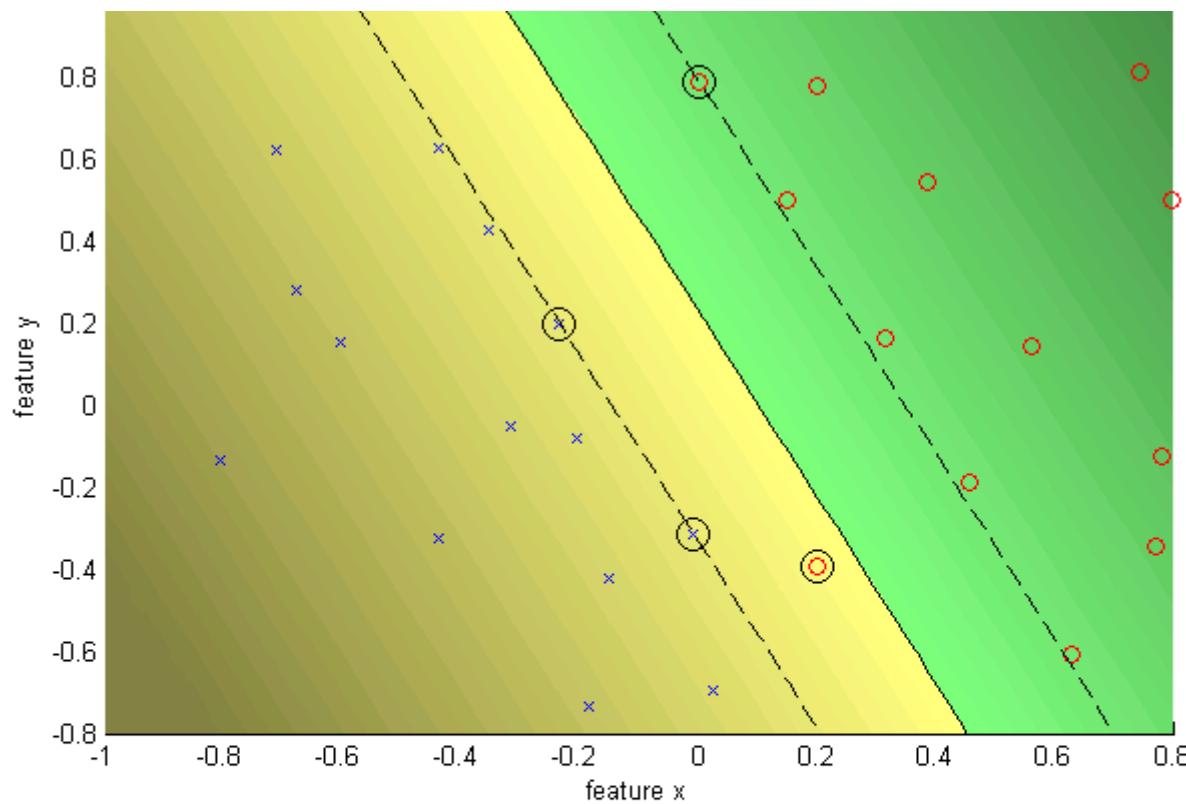


- data is linearly separable
- but only with a narrow margin

$C = \text{Infinity}$ hard margin



$C = 10$ soft margin



Comment Window

SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: 10.0000
Kernel evaluations: 2645
Number of Support Vectors: 4
Margin: 0.2265
Training error: 3.70%

Application: Pedestrian detection in Computer Vision

Objective: detect (localize) standing humans in an image

- cf face detection with a sliding window classifier



- reduces object detection to binary classification

- does an image window contain a person or not?

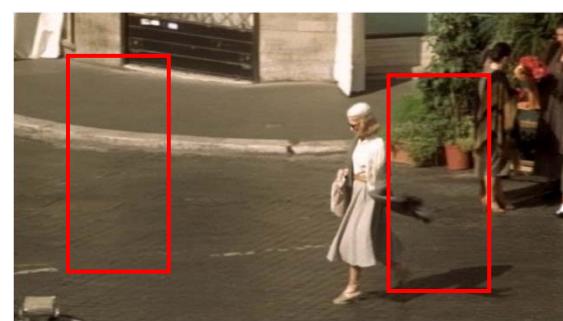
Method: the HOG detector

Training data and features

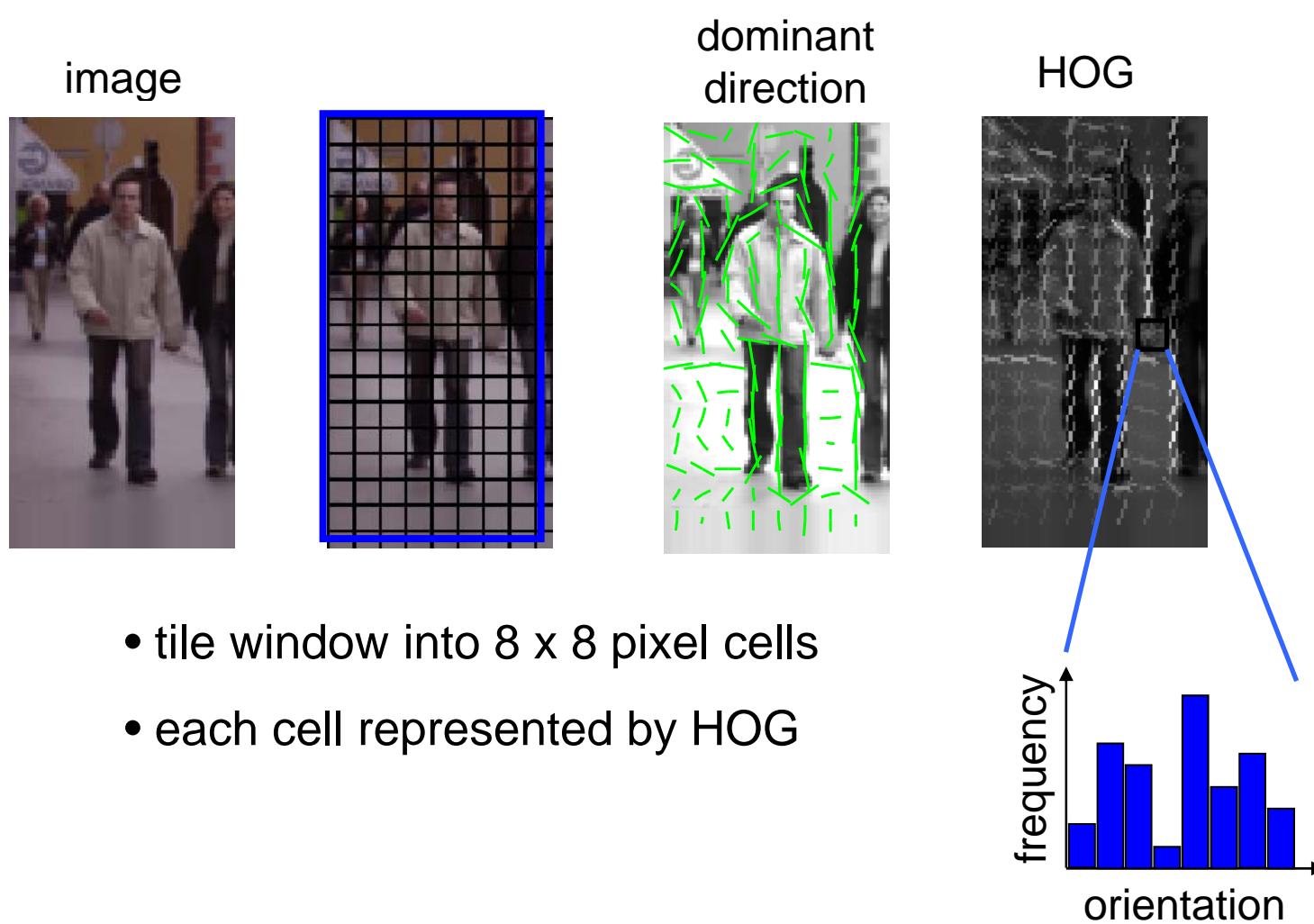
- Positive data – 1208 positive window examples



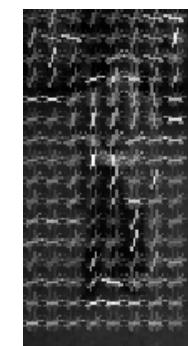
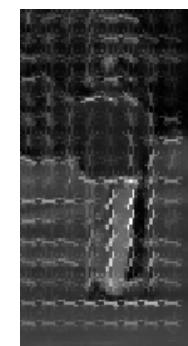
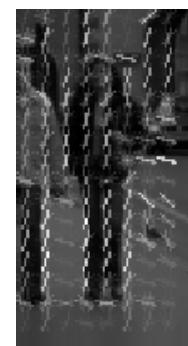
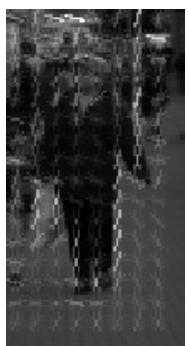
- Negative data – 1218 negative window examples (initially)



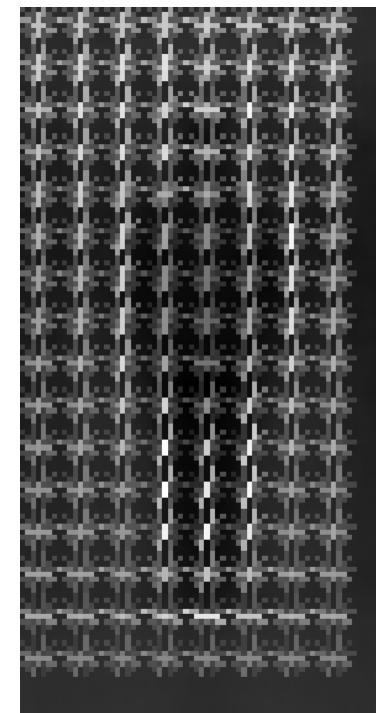
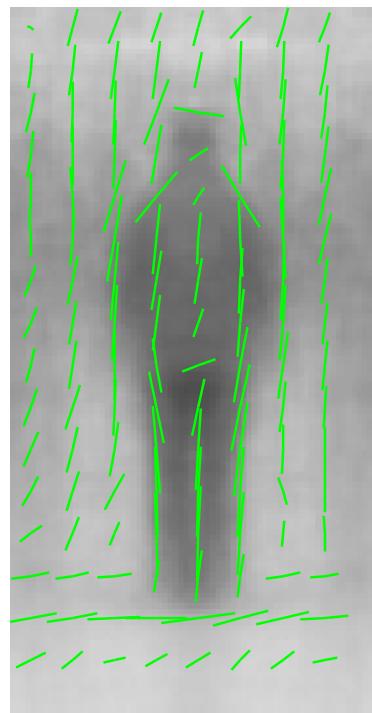
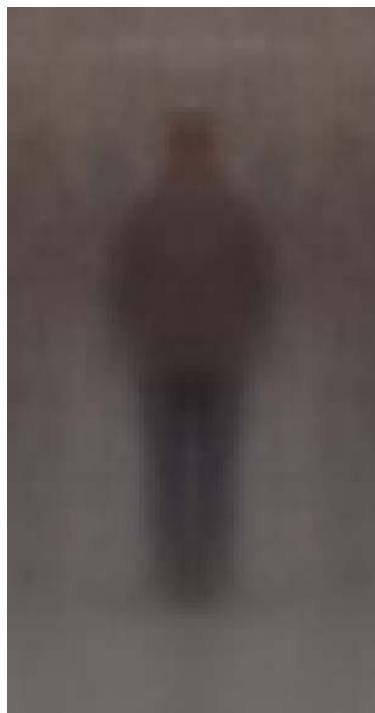
Feature: histogram of oriented gradients (HOG)



Feature vector dimension = 16×8 (for tiling) $\times 8$ (orientations) = 1024



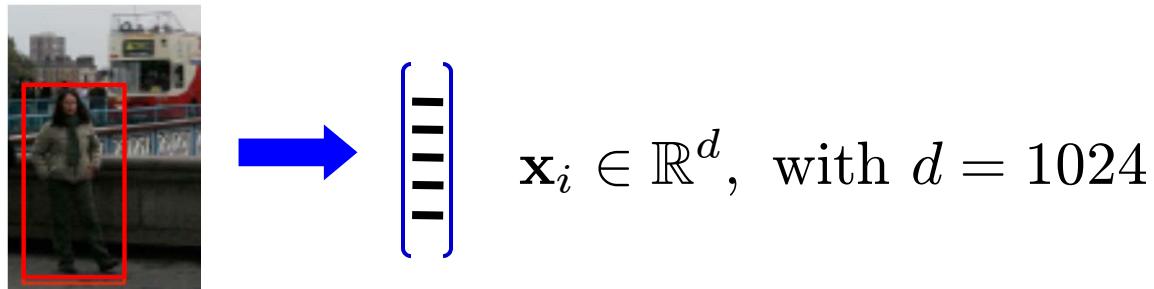
Averaged positive examples



Algorithm

Training (Learning)

- Represent each example window by a HOG feature vector



- Train a SVM classifier

Testing (Detection)

- Sliding window classifier

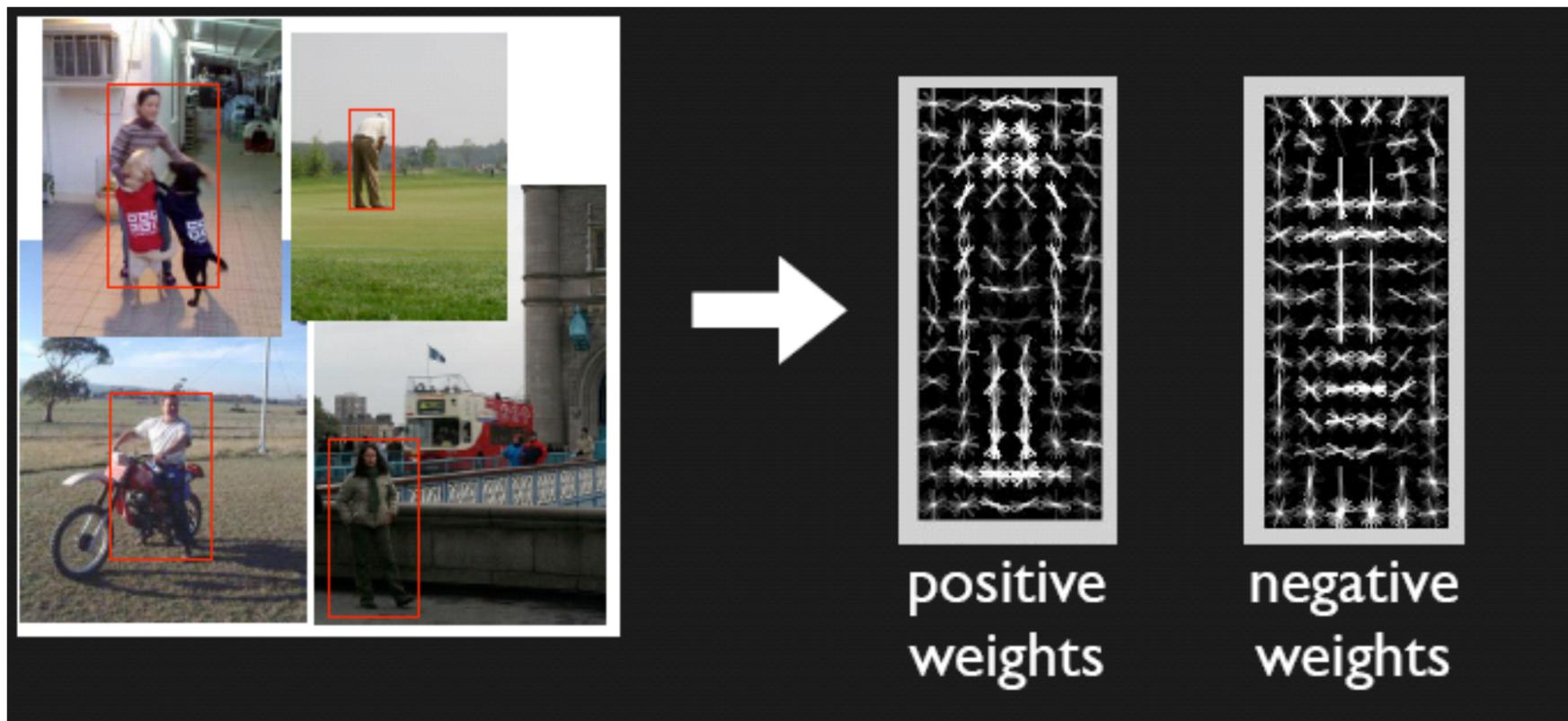
$$f(x) = \mathbf{w}^\top \mathbf{x} + b$$



Dalal and Triggs, CVPR 2005

Learned model

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



Slide from Deva Ramanan

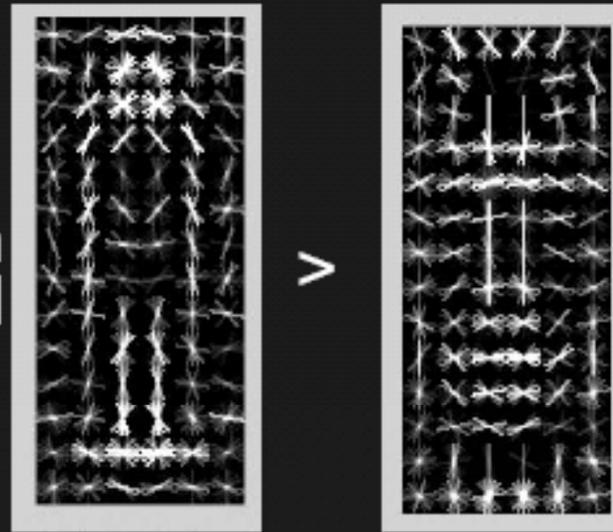
What do negative weights mean?

$$wx > 0$$

$$(w_+ - w_-)x > 0$$

$$w_+ > w_-x$$

pedestrian
model



pedestrian
background
model

Complete system should compete pedestrian/pillar/doorway models

Discriminative models come equipped with own bg
(avoid firing on doorways by penalizing vertical edges)

Optimization

Learning an SVM has been formulated as a [constrained](#) optimization problem over \mathbf{w} and ξ

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i \text{ subject to } y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

The constraint $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max(0, 1 - y_i f(\mathbf{x}_i))$$

Hence the learning problem is equivalent to the [unconstrained](#) optimization problem over \mathbf{w}

$$\min_{\mathbf{w} \in \mathbb{R}^d} \underbrace{\|\mathbf{w}\|^2}_{\text{regularization}} + C \sum_i^N \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Loss function

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

loss function

Points are in three categories:

1. $y_i f(\mathbf{x}_i) > 1$

Point is outside margin.

No contribution to loss

2. $y_i f(\mathbf{x}_i) = 1$

Point is on margin.

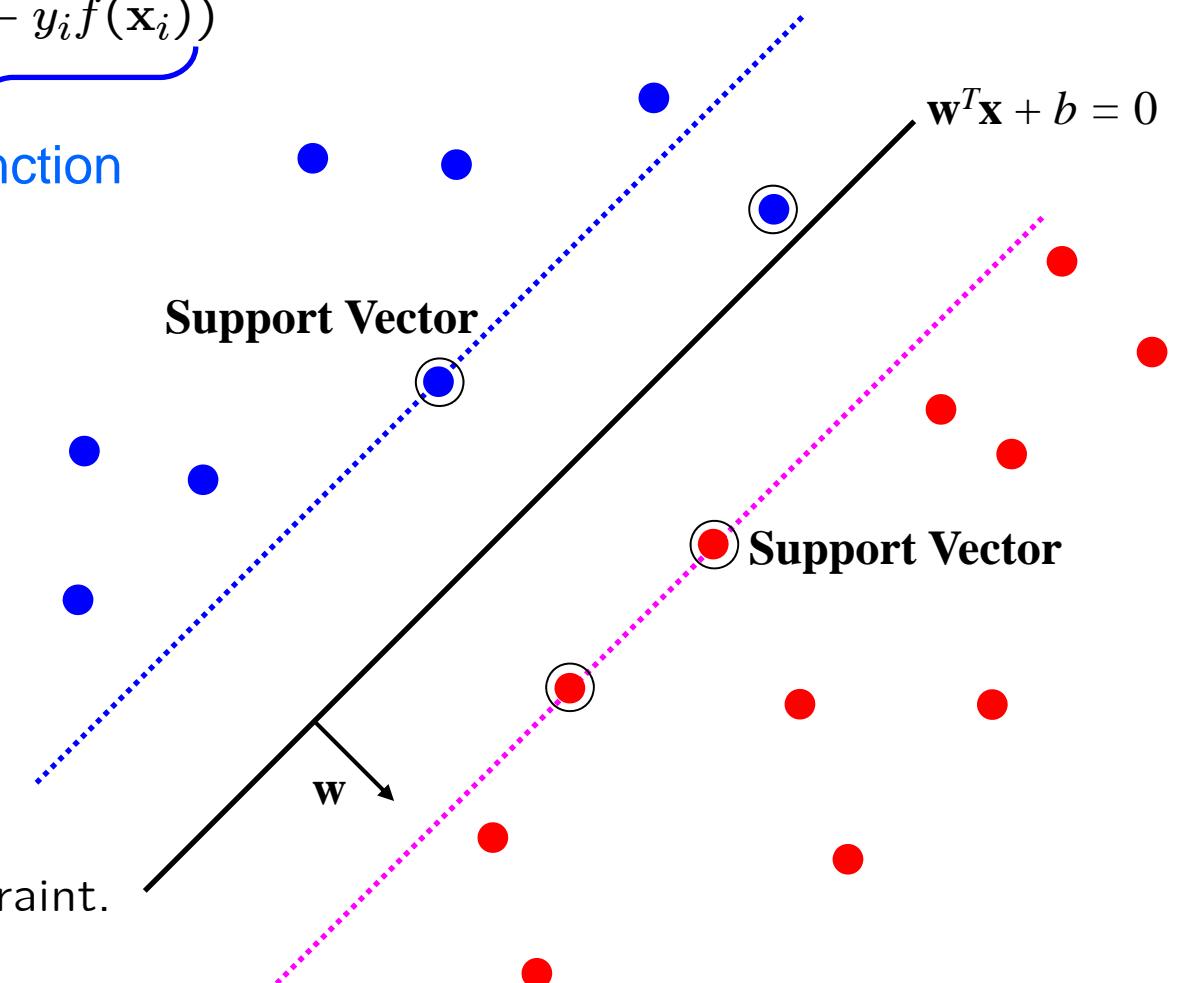
No contribution to loss.

As in hard margin case.

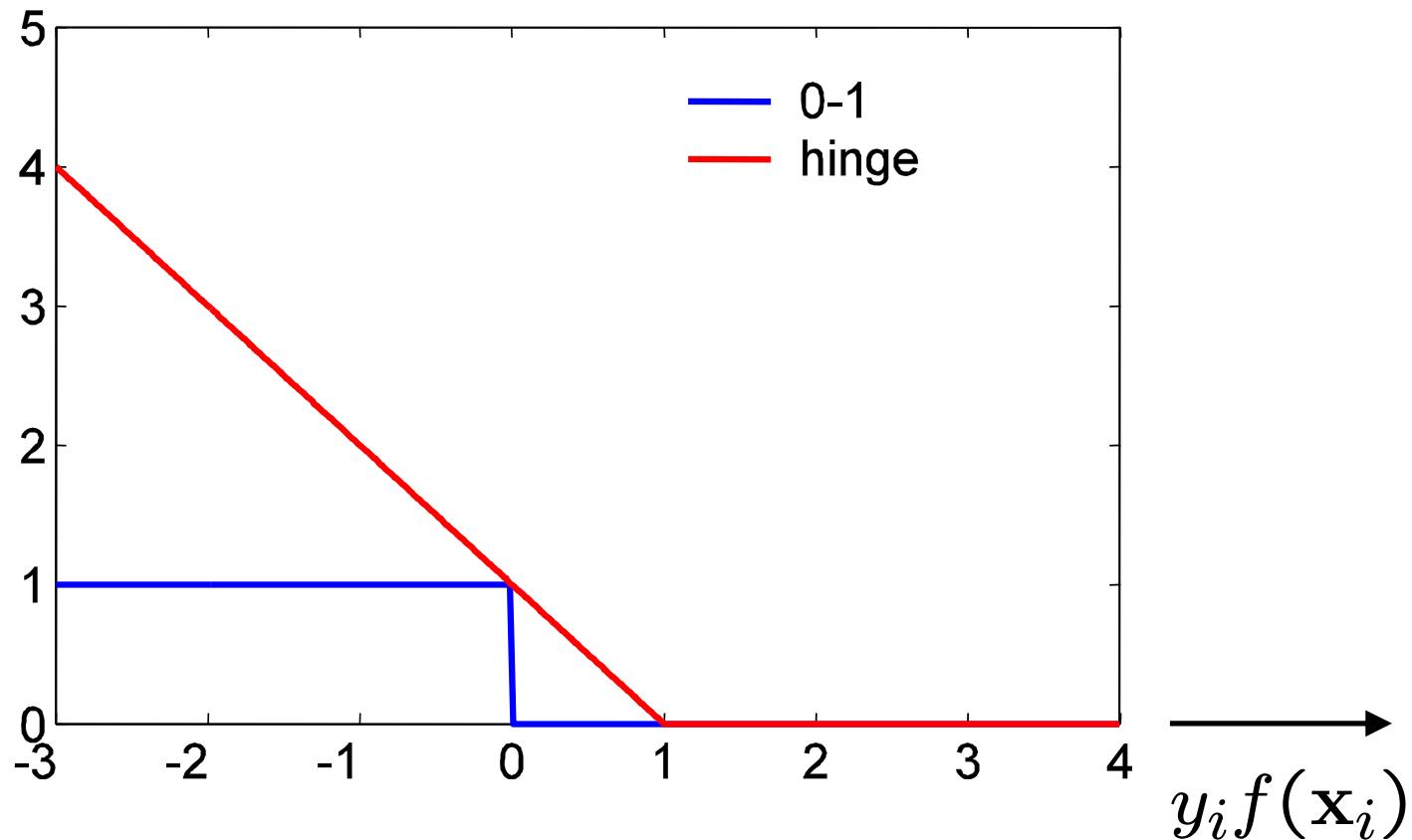
3. $y_i f(\mathbf{x}_i) < 1$

Point violates margin constraint.

Contributes to loss



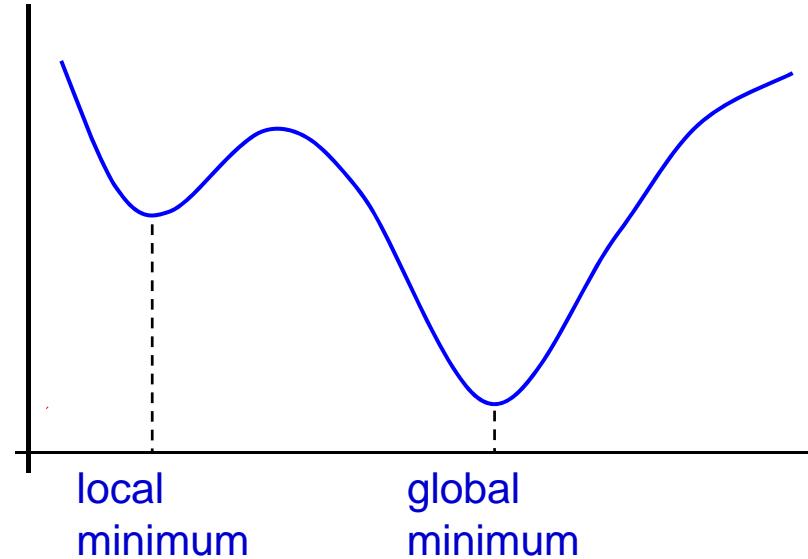
Loss functions



- SVM uses “hinge” loss $\max(0, 1 - y_i f(\mathbf{x}_i))$
- an approximation to the 0-1 loss

Optimization continued

$$\min_{\mathbf{w} \in \mathbb{R}^d} C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + \|\mathbf{w}\|^2$$



- Does this cost function have a unique solution?
- Does the solution depend on the starting point of an iterative optimization algorithm (such as gradient descent)?

If the cost function is **convex**, then a locally optimal point is globally optimal (provided the optimization is over a convex set, which it is in our case)

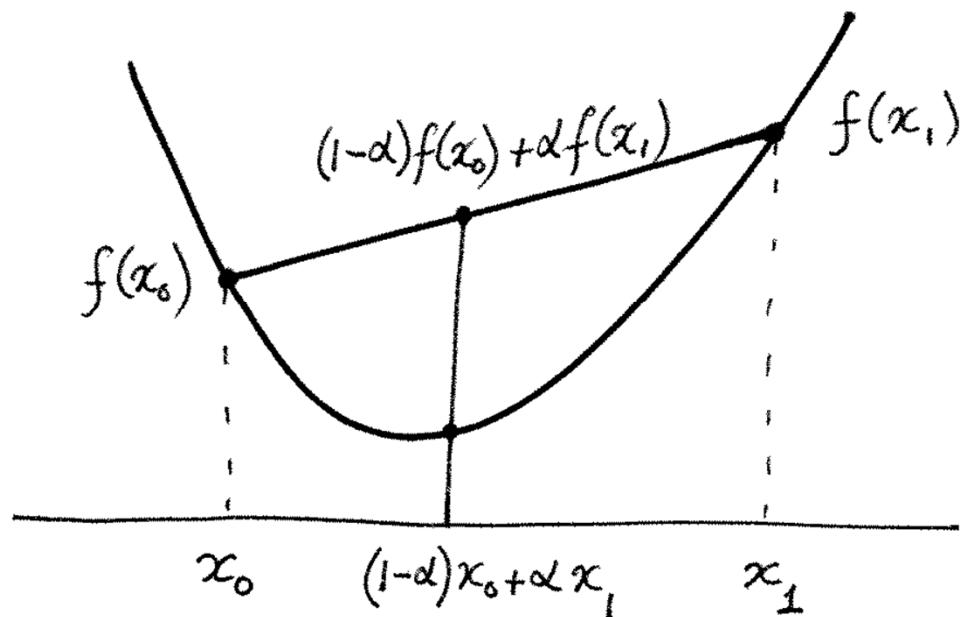
Convex functions

D – a domain in \mathbb{R}^n .

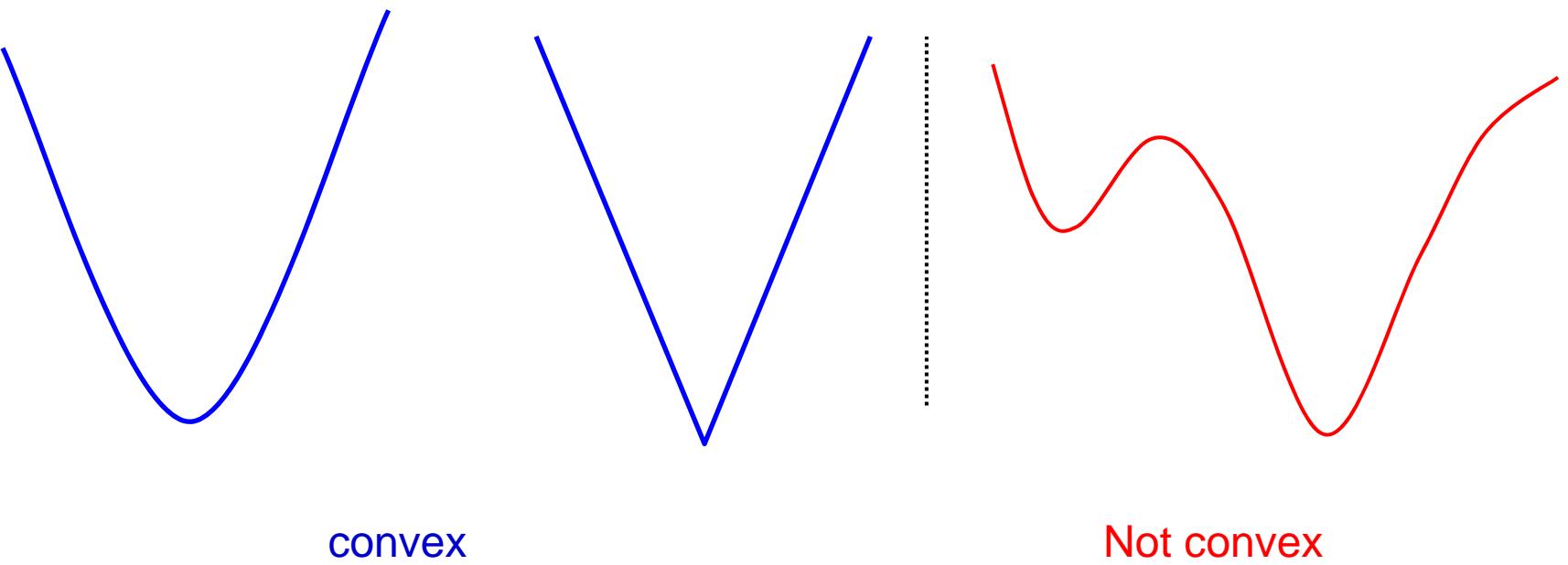
A **convex function** $f : D \rightarrow \mathbb{R}$ is one that satisfies, for any x_0 and x_1 in D :

$$f((1 - \alpha)x_0 + \alpha x_1) \leq (1 - \alpha)f(x_0) + \alpha f(x_1) .$$

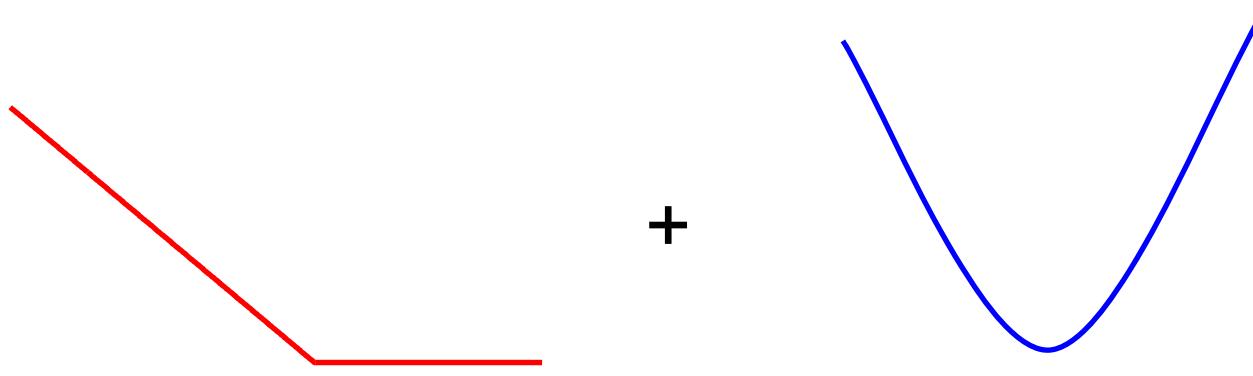
Line joining $(x_0, f(x_0))$ and $(x_1, f(x_1))$ lies above the function graph.



Convex function examples



A non-negative sum of convex functions is convex



SVM

$$\min_{\mathbf{w} \in \mathbb{R}^d} C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) + \|\mathbf{w}\|^2$$

convex

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an [average](#)

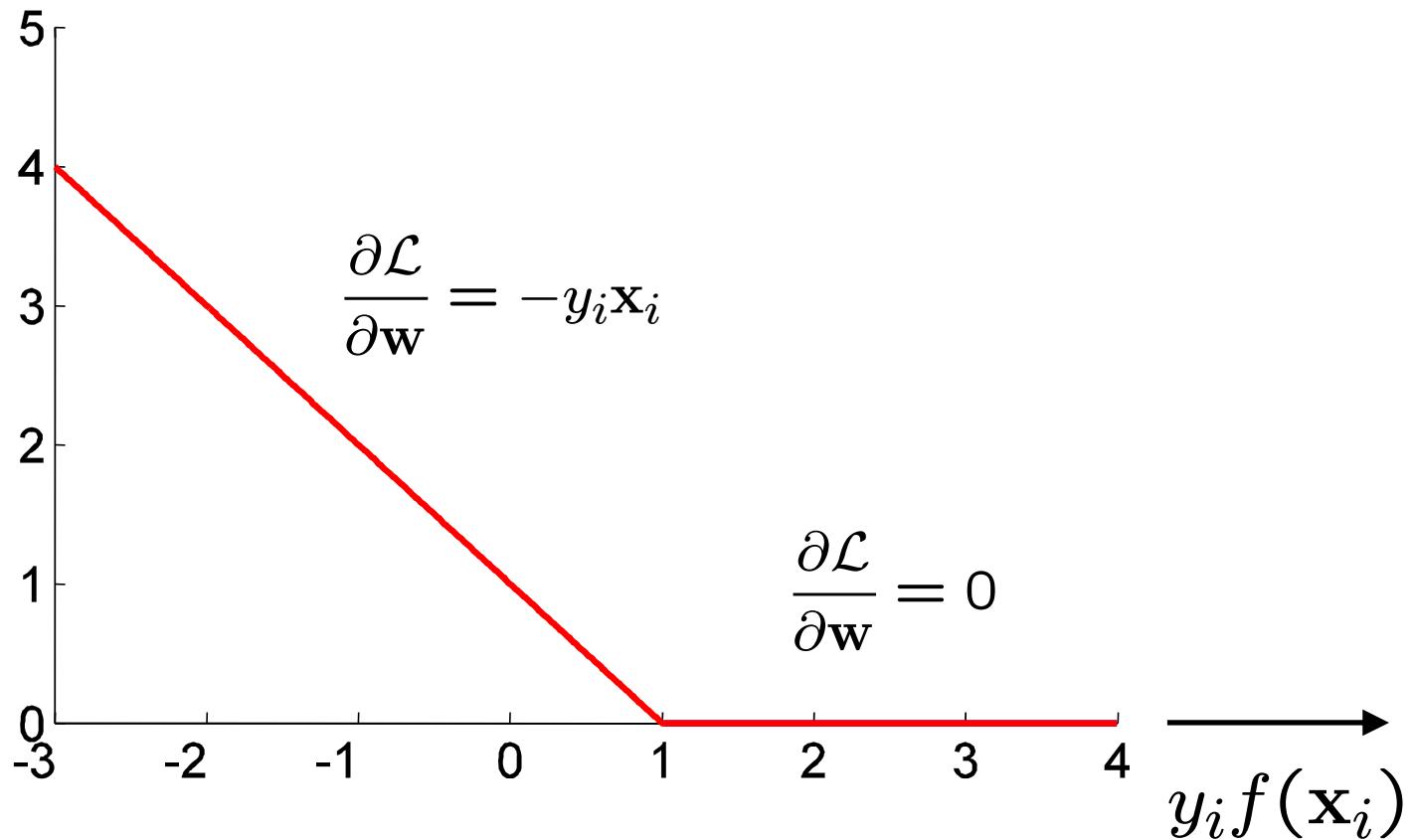
$$\begin{aligned}\min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \max(0, 1 - y_i f(\mathbf{x}_i)) \right)\end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and
 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a [sub-gradient](#) is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \quad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) \right)$$

The iterative update is

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta \frac{1}{N} \sum_i^N (\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t)) \end{aligned}$$

where η is the learning rate.

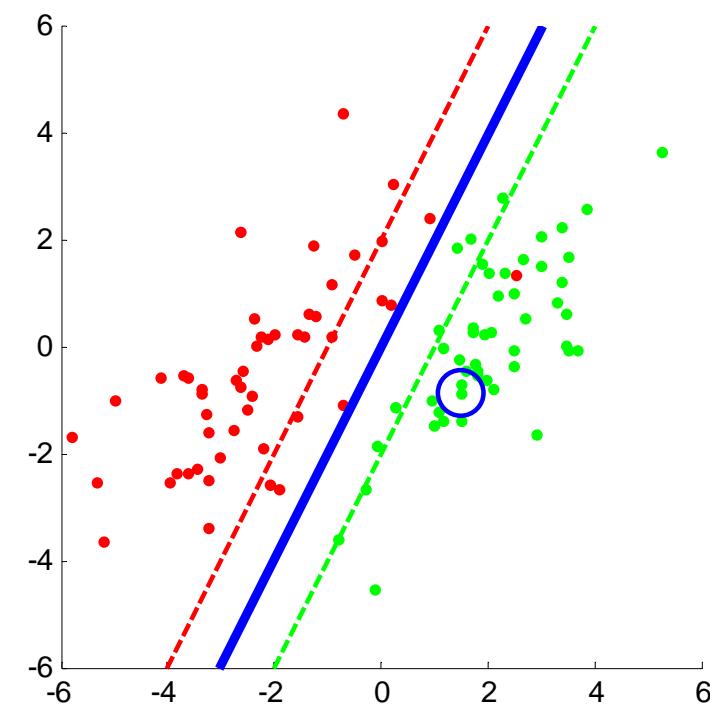
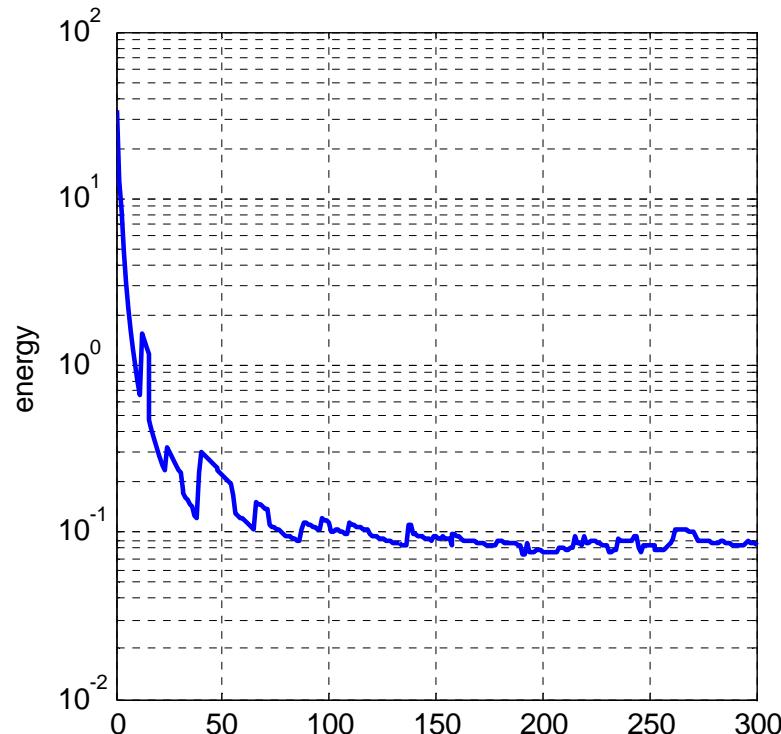
Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta \lambda \mathbf{w}_t && \text{otherwise} \end{aligned}$$

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$

Pegasos – Stochastic Gradient Descent Algorithm

Randomly sample from the training data



Background reading and more ...

- Next lecture – see that the SVM can be expressed as a sum over the support vectors:

$$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$


support vectors

- On web page:
<http://www.robots.ox.ac.uk/~az/lectures/ml>
- links to SVM tutorials and video lectures
- MATLAB SVM demo