Kernel Methods

1. Maximum margin classifier
   (explicit feature space, requires linearly separable data)

2. Support vector classifier
   (explicit feature space, linear boundaries)

3. Kernel functions
   (making features space transforms easy)

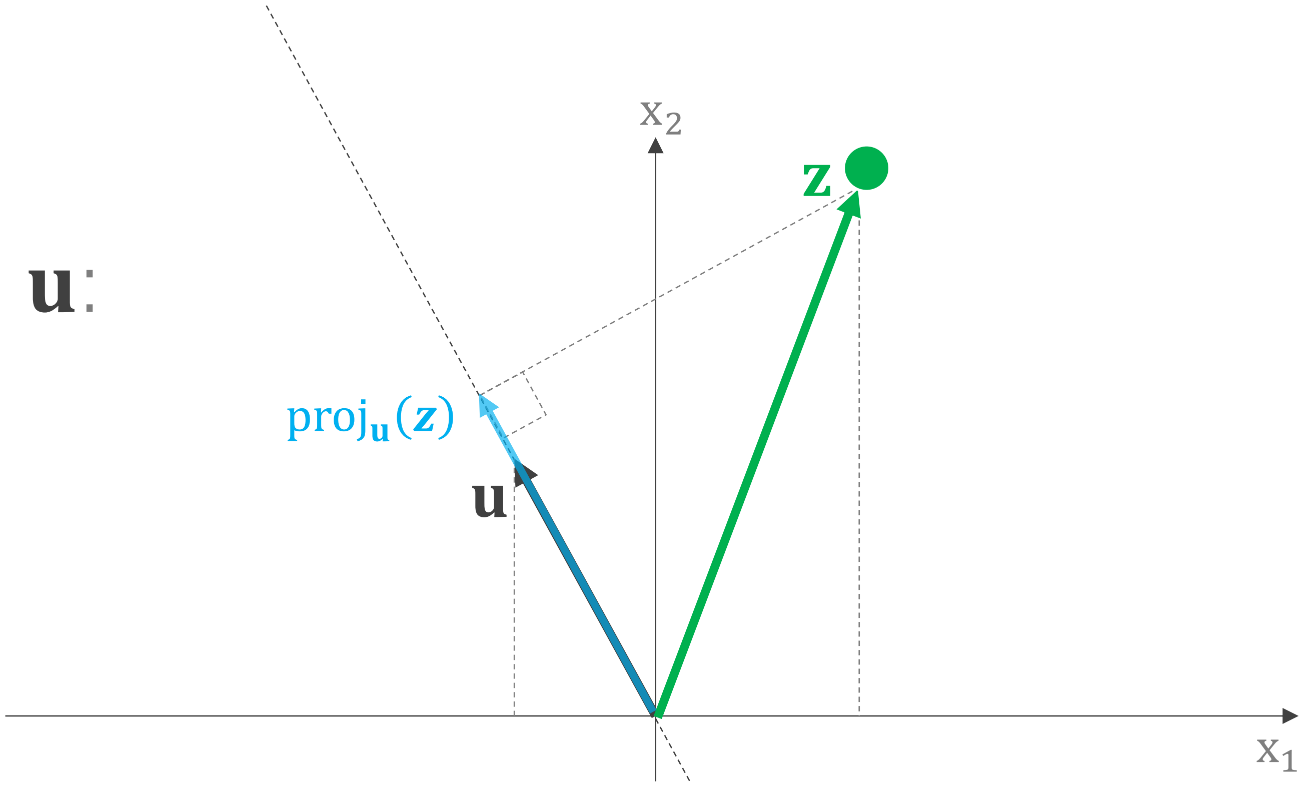
4. Perceptron → kernel perceptron
   (linearly separable data, the kernel trick)

5. Support vector machine
   (kernel-transformed implicit feature space, non-linear boundaries)

Projections

The vector projection of $\boldsymbol{z}$ onto $\mathbf{u}$:

$$\mathrm{proj}_{\mathbf{u}}(\boldsymbol{z}) = \left(\frac{\mathbf{u}^T\mathbf{z}}{\|\mathbf{u}\|}\right)\frac{\mathbf{u}}{\|\mathbf{u}\|}$$

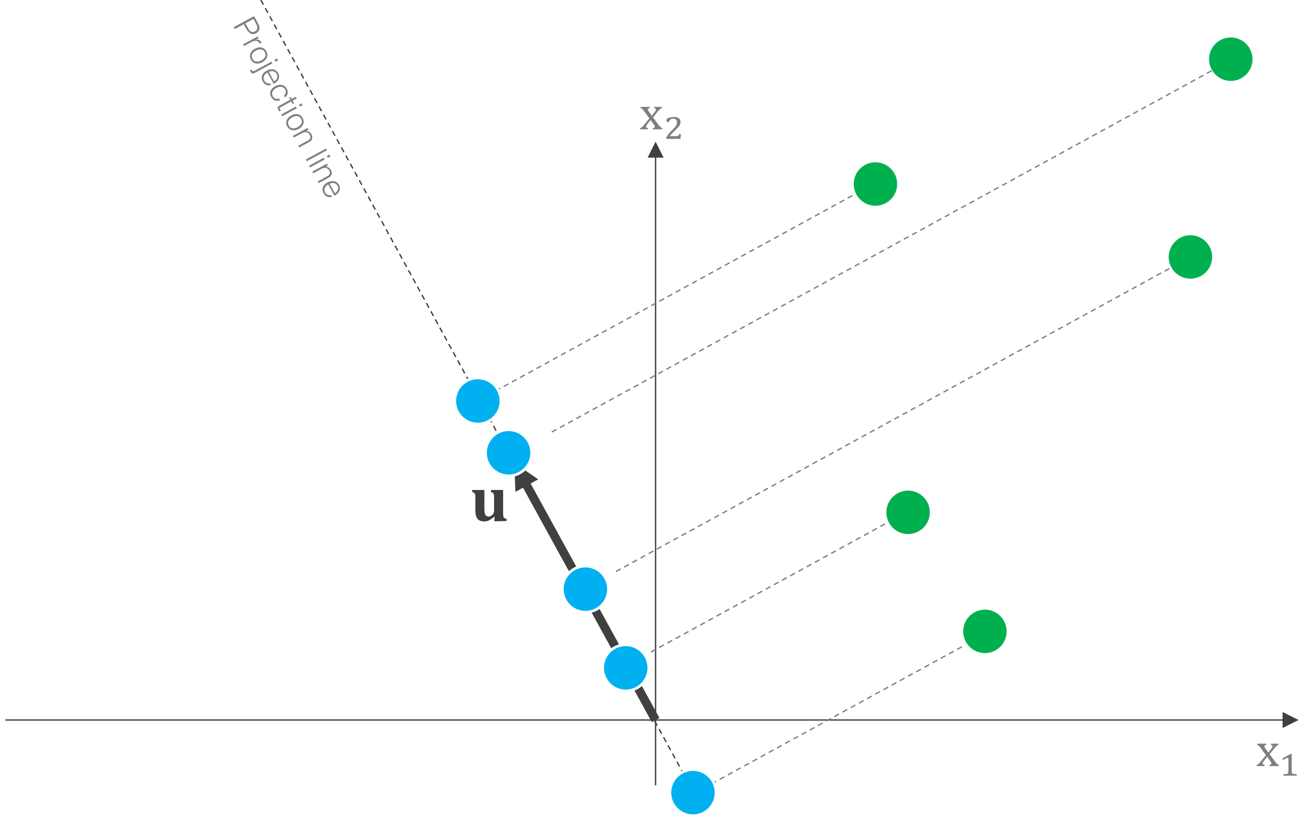


The scalar length (Euclidean or L_2 norm) of the vector $\mathbf{u}$ is $\|\mathbf{u}\|$

If we assume $\mathbf{u}$ is a unit vector then $\|\mathbf{u}\| = 1$ $\qquad$ $\mathrm{proj}_{\mathbf{u}}(\boldsymbol{z}) = (\mathbf{u}^T\mathbf{z})\mathbf{u}$

Magnitude of projection onto direction of $\mathbf{u}$

Projections

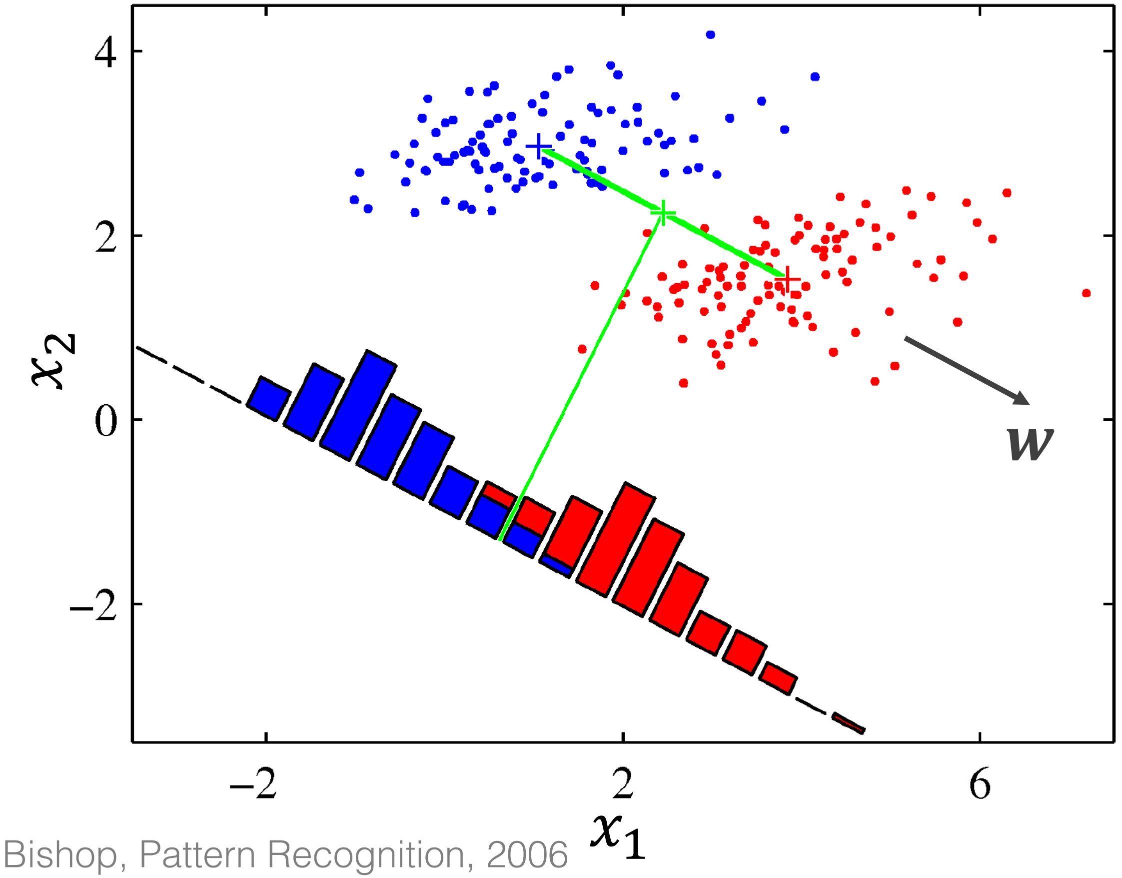


We could project any points in this space onto the line defined by the direction of unit vector $\mathbf{u}$

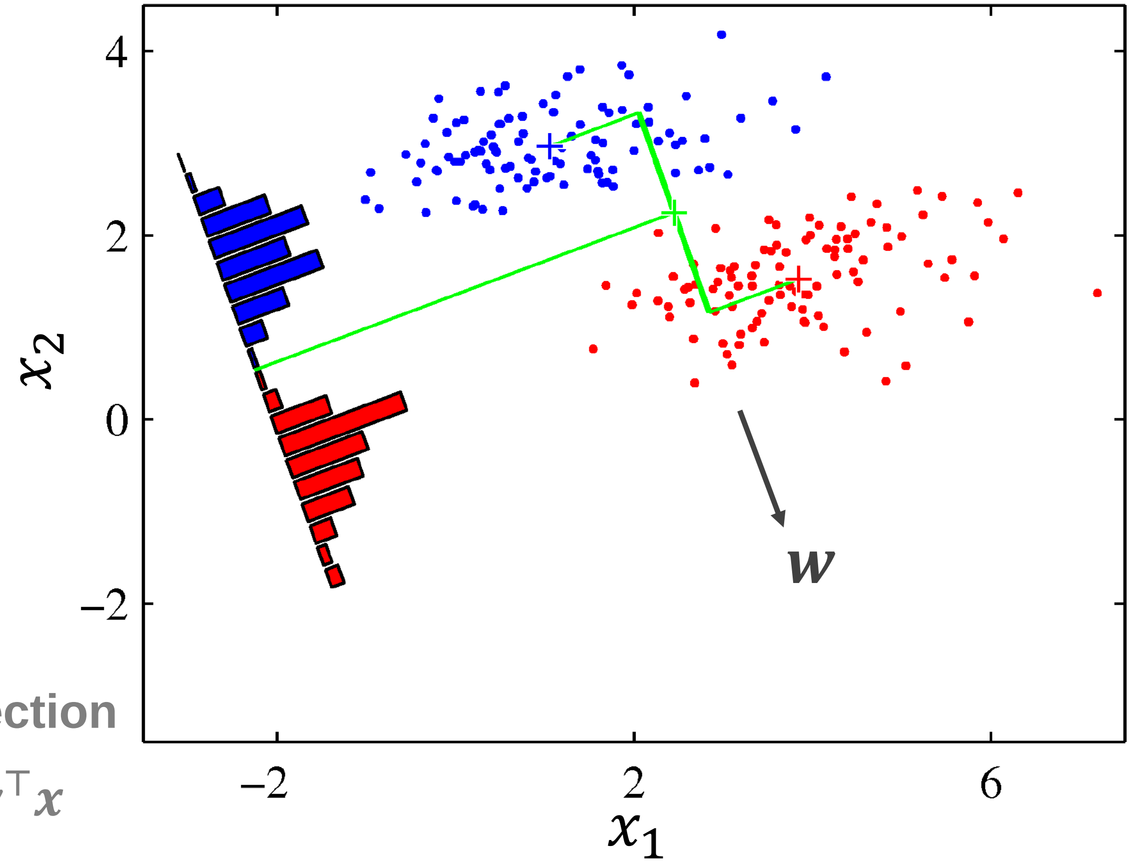
Linear Discriminant Analysis

Looks for the projection into the one dimension that “best” separates the classes

Projection onto line connecting the means



Projection onto a line providing improved class separation



Linear projection

$$\hat{f}(x) = w^\top x$$

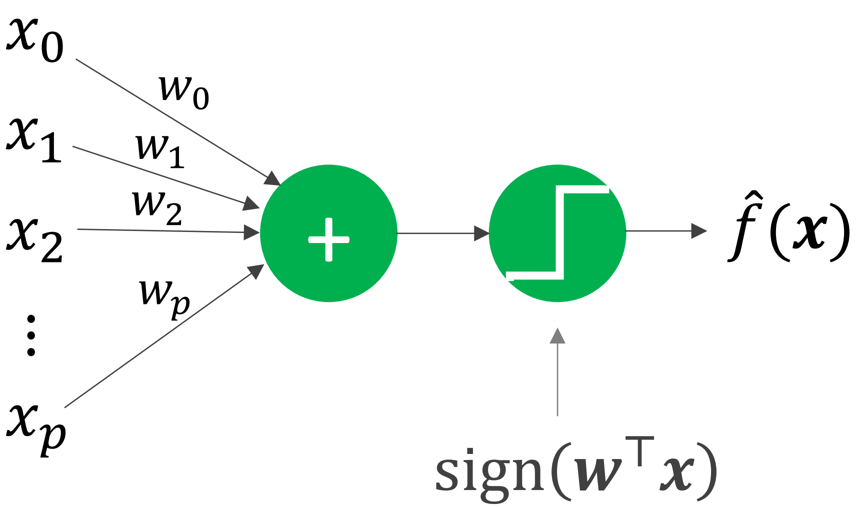
Bishop, Pattern Recognition, 2006

Linear classifier

Linear Classification

$$\hat{f}(\boldsymbol{x}) = f\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$= f(\boldsymbol{w}^\top \boldsymbol{x})$$



Training data: $(\boldsymbol{x}_i, y_i), \quad i = 1, \dots, N$
with binary $y_i = \{-1,1\}$

Decision rule based on $sign(\boldsymbol{w}^T \boldsymbol{x})$:
if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$
if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$

Source: Abu-Mostafa, Learning from Data, Caltech

When we see the expression:

$$\boldsymbol{w}^\top \boldsymbol{x} > 0$$

…we're typically using a separating hyperplane as a decision rule

In a 1-D feature space, this is a point
In a 2-D feature space, this is a line
In a 3-D feature space, this is a plane
In a 4-D and higher feature space, this is a hyperplane

The separating hyperplane

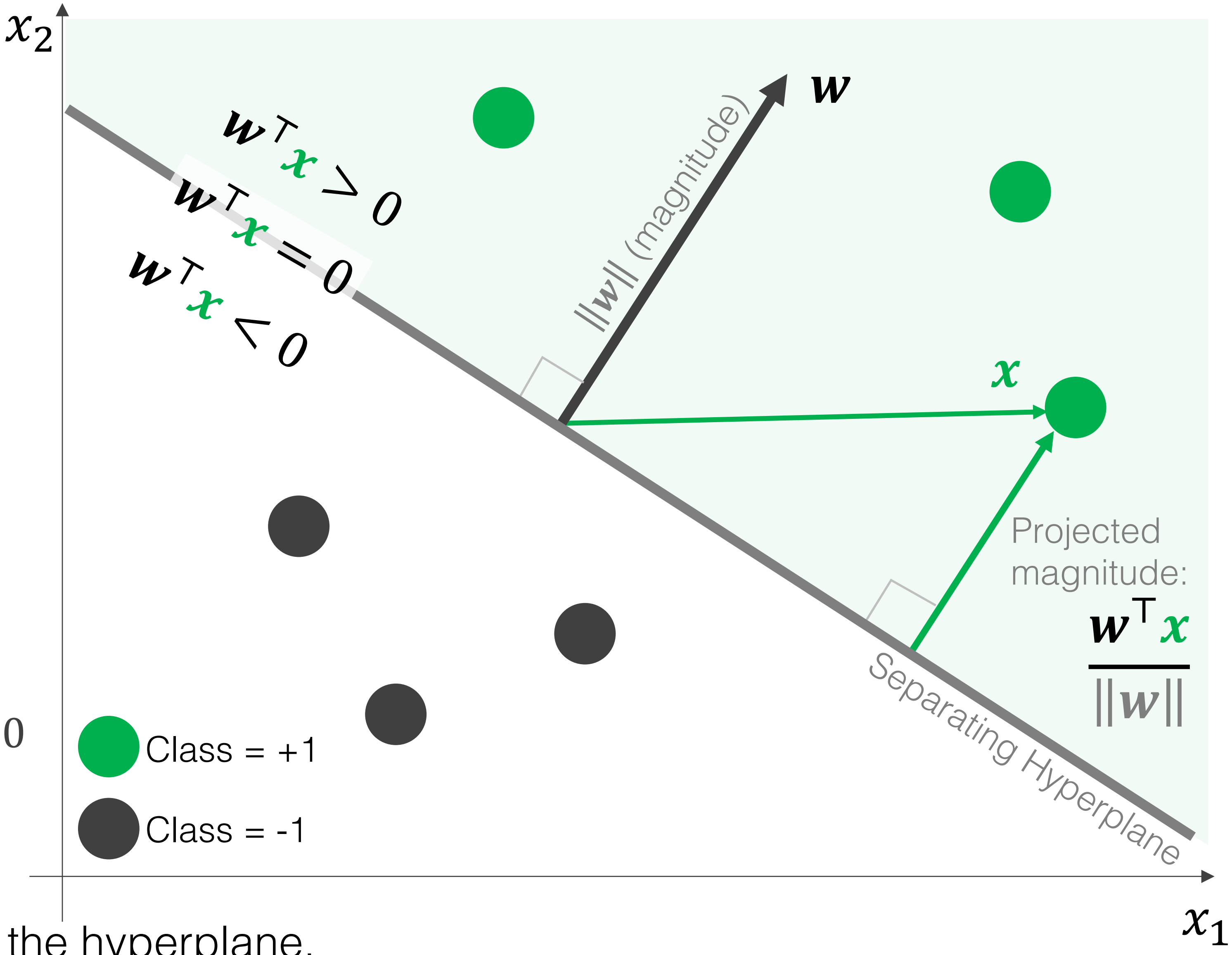
$\boldsymbol{w}$ defines and is orthogonal to the separating hyperplane

$$\hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^\top \boldsymbol{x})$$

Decision rule based on $sign(\boldsymbol{w}^T\boldsymbol{x})$:

if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$
if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$



Interpretation: if a point is on one side of the hyperplane, assign one class, if it's on the other, assign the other class

We constrain $\|\boldsymbol{w}\| = 1$

The separating hyperplane

$\boldsymbol{w}$ defines and is orthogonal to the separating hyperplane

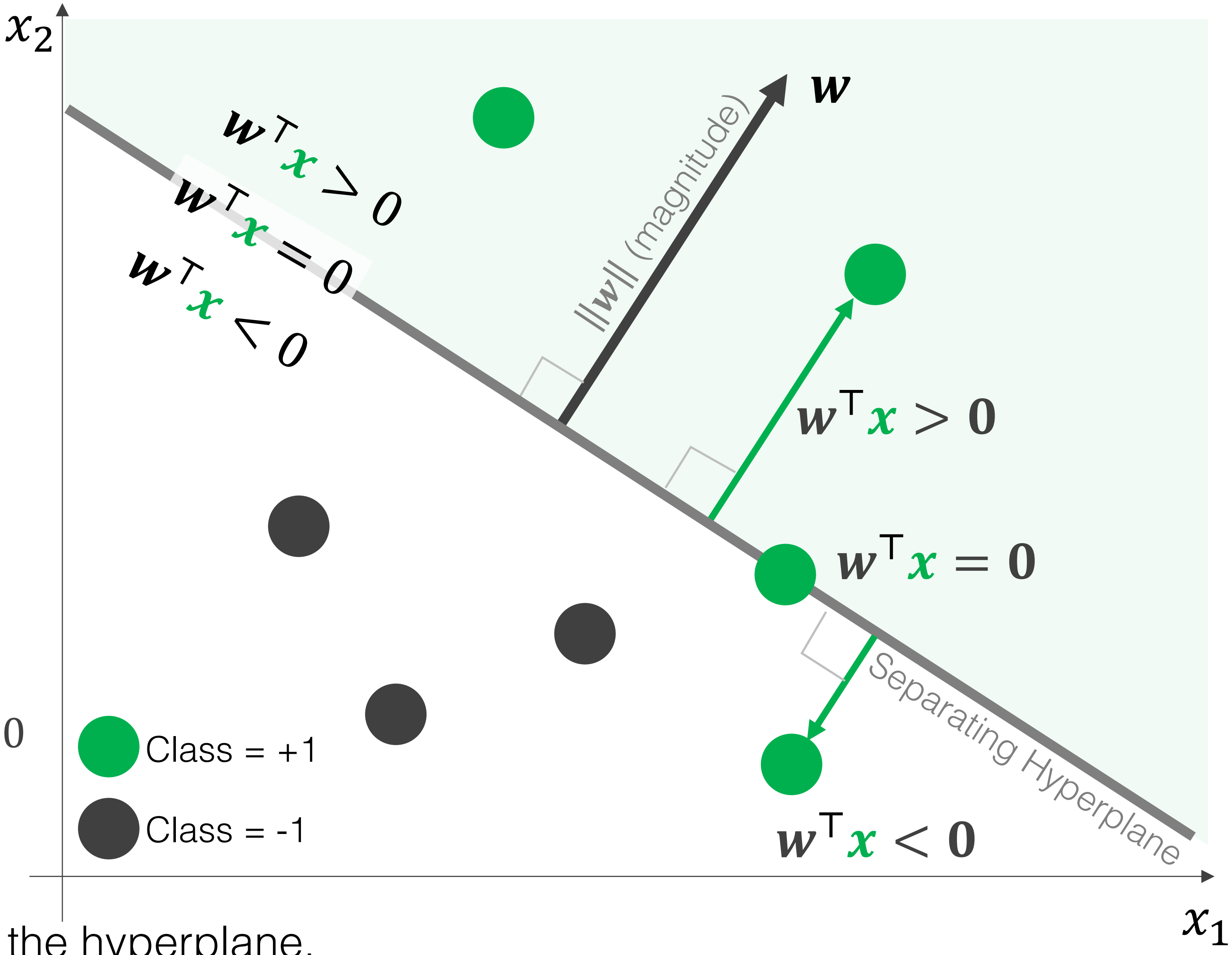
$$\hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^\top \boldsymbol{x})$$

Decision rule based on $sign(\boldsymbol{w}^T \boldsymbol{x})$:

if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$

if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$



Interpretation: if a point is on one side of the hyperplane, assign one class, if it's on the other, assign the other class

We constrain $||\boldsymbol{w}|| = 1$

The separating hyperplane

$\boldsymbol{w}$ defines and is orthogonal to the separating hyperplane

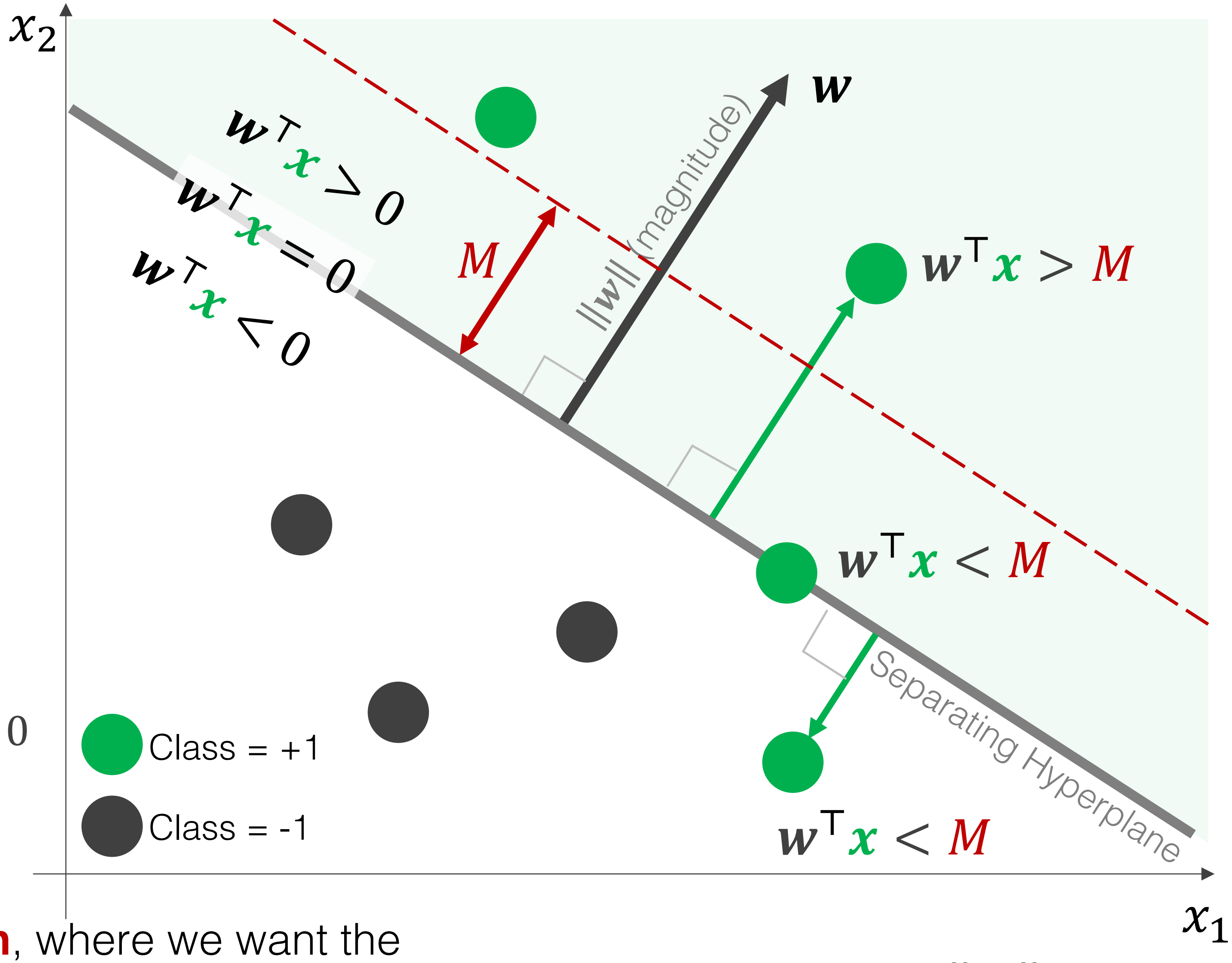
$$\hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^\top \boldsymbol{x})$$

Decision rule based on $sign(\boldsymbol{w}^T \boldsymbol{x})$:

if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$

if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$



We can introduce the concept of **margin**, where we want the point to be even further beyond the separating hyperplane

We constrain $||\boldsymbol{w}|| = 1$

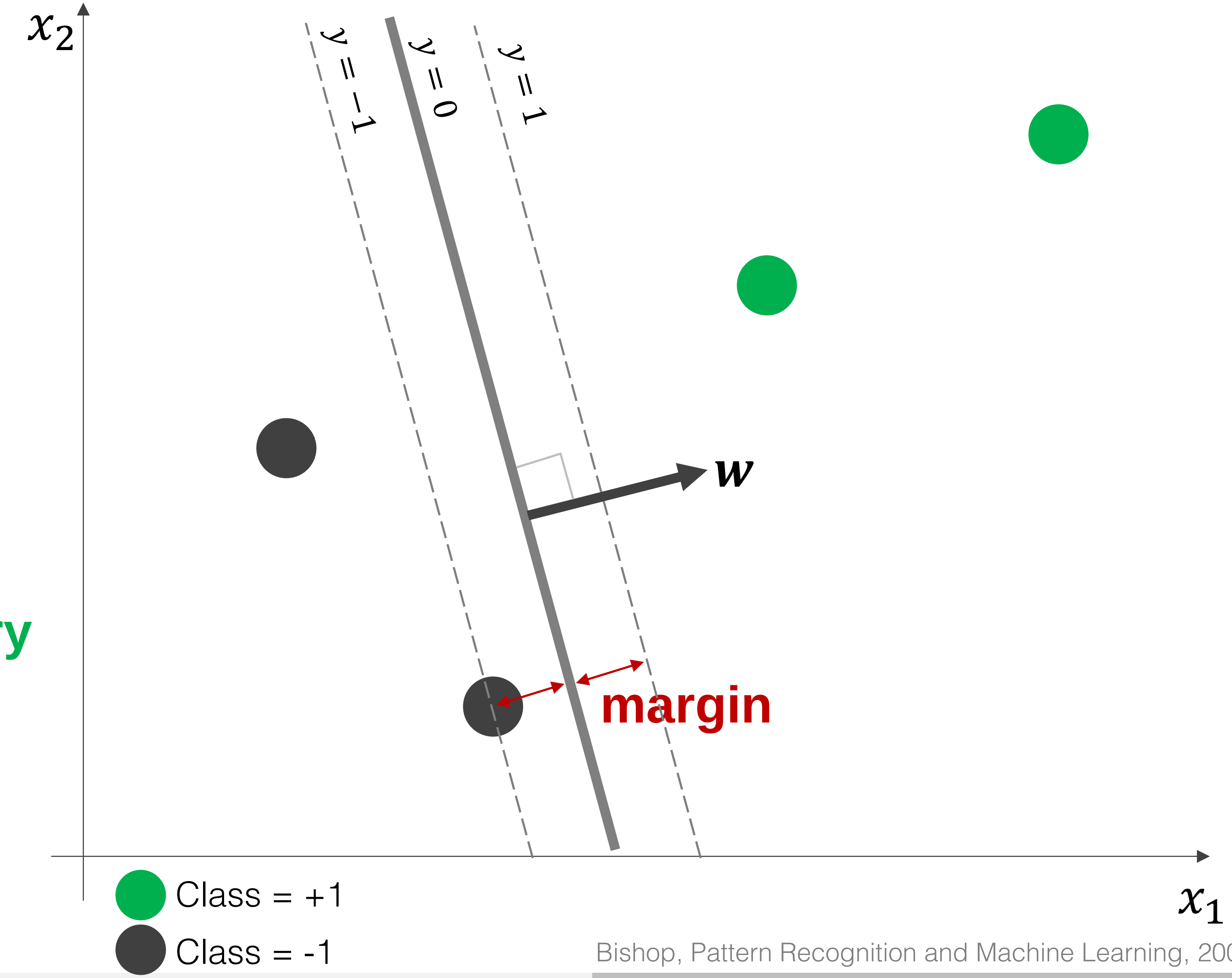
The concept of margin

Assume our data are linearly separable

How do we pick the "best" separating line (hyperplane)?

Maximize the **margin**

Margin = the smallest distance between the **decision boundary** and **any** of the samples



Bishop, Pattern Recognition and Machine Learning, 2006

Maximum margin classifier

The decision boundary is determined by the weight, $\boldsymbol{w}$, as with the perceptron

Pick $\boldsymbol{w}$ to maximize the margin

Assumes linear separability

Hard margin classifier

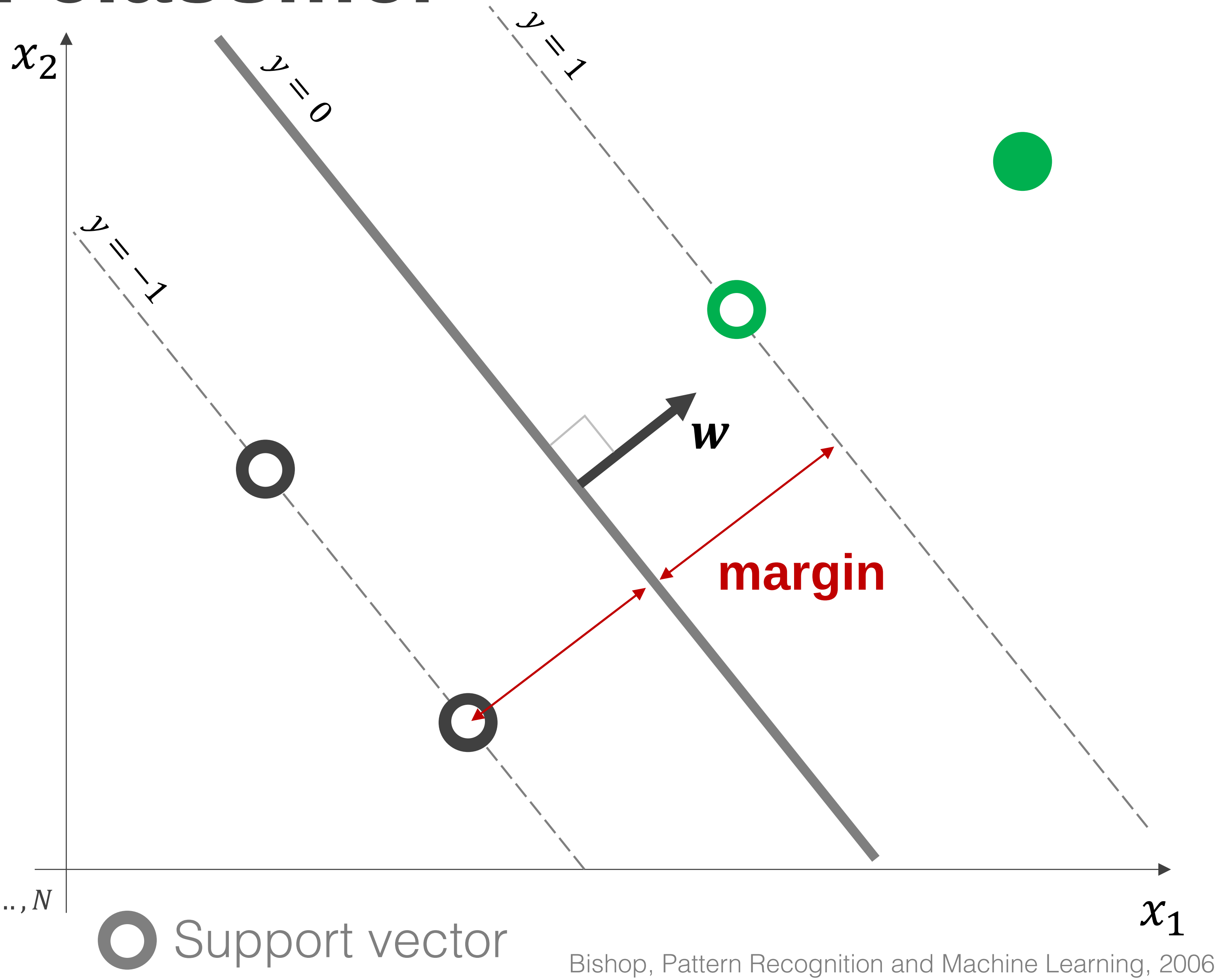
Optimization Problem:

$\max_{\boldsymbol{w}} M$ (Maximize the margin by changing the weights)

subject to $\|\boldsymbol{w}\| = 1$ (Unit norm – sum of squares of parameters is 1)

$y_i \boldsymbol{w}^\top \boldsymbol{x}_i > M$ for all $i = 1,2,\dots,N$

(every training sample must be correctly classified and a distance M from the hyperplane)



Bishop, Pattern Recognition and Machine Learning, 2006

Hard margin classifiers

May be sensitive to small changes in training data

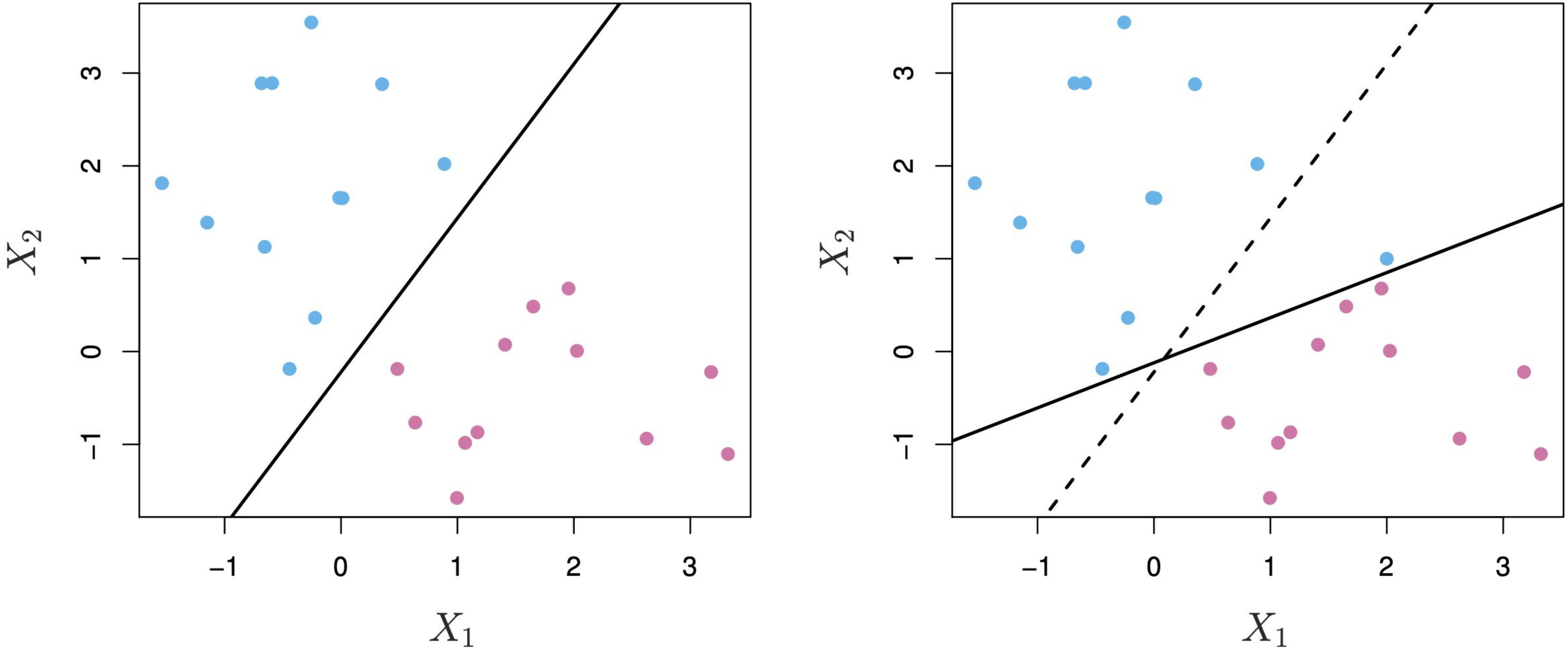


Figure from Introduction to Statistical Learning

In most cases are data are not linearly separable…

Maximum margin classifiers can’t handle this…

Support Vector Classifiers can!

Support vector classifier

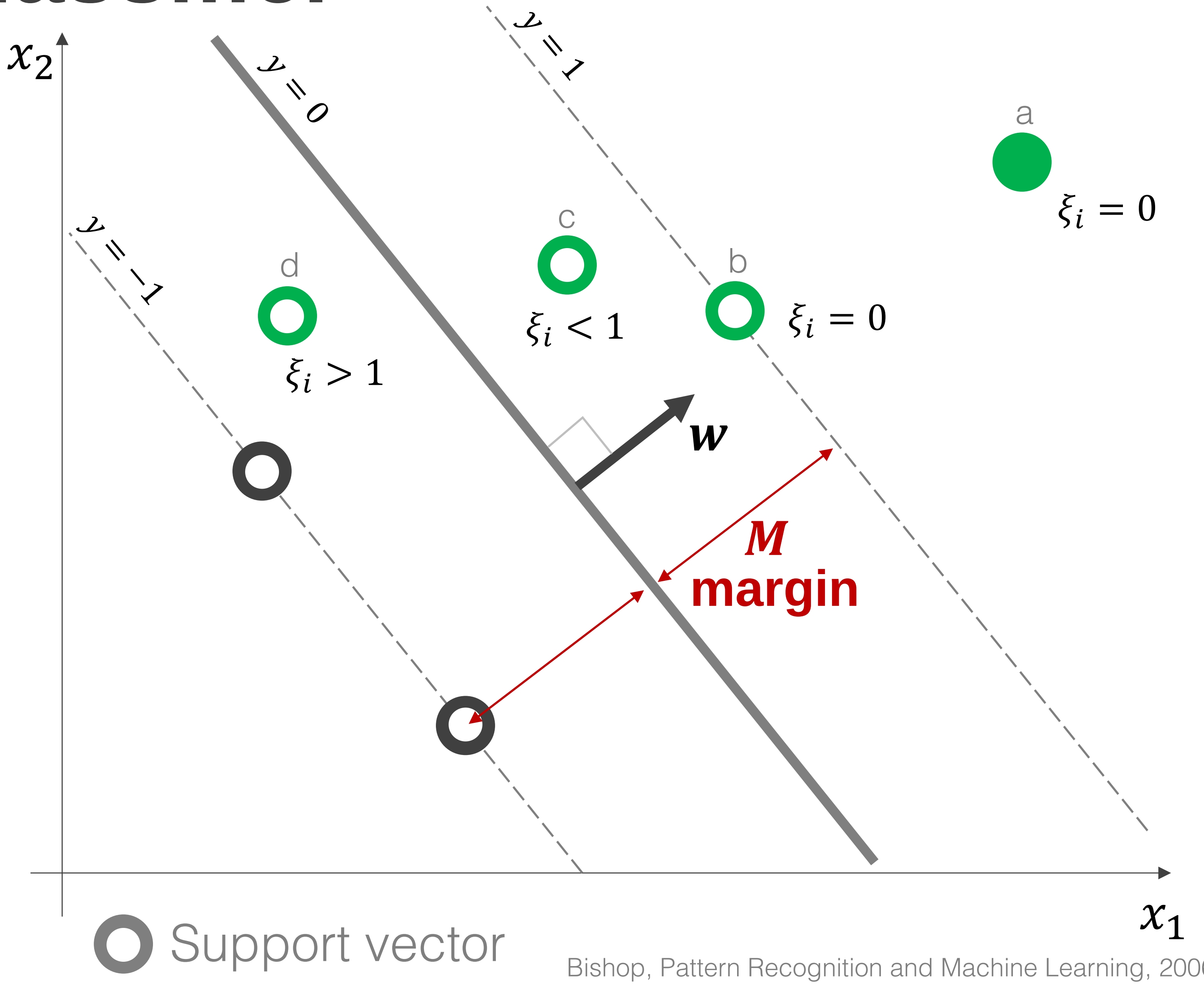
We allow samples to violate the margin and assign a penalty for each, $\xi_i \geq 0$

Correct Prediction
If the sample is outside the margin (a) or at the margin (b) there is no penalty $\xi_i = 0$

Margin violation
If the sample is inside the margin but on the correct side of the separating hyperplane (c) $\xi_i < 1$

Incorrect Prediction
If the sample is on the wrong side of the separating hyperplane (d) $\xi_i > 1$

$\xi_i \geq 0$ in all cases – it represents how much slack we give to violate the margin



Bishop, Pattern Recognition and Machine Learning, 2006

Support vector classifier

Optimization:

$\max_{\boldsymbol{w},\xi_1,\xi_2,\ldots,\xi_N} M$ (Maximize the margin by changing the weights and margin violation penalties)

subject to $\|\boldsymbol{w}\| = 1$ (Unit norm – sum of squares of parameters is 1)

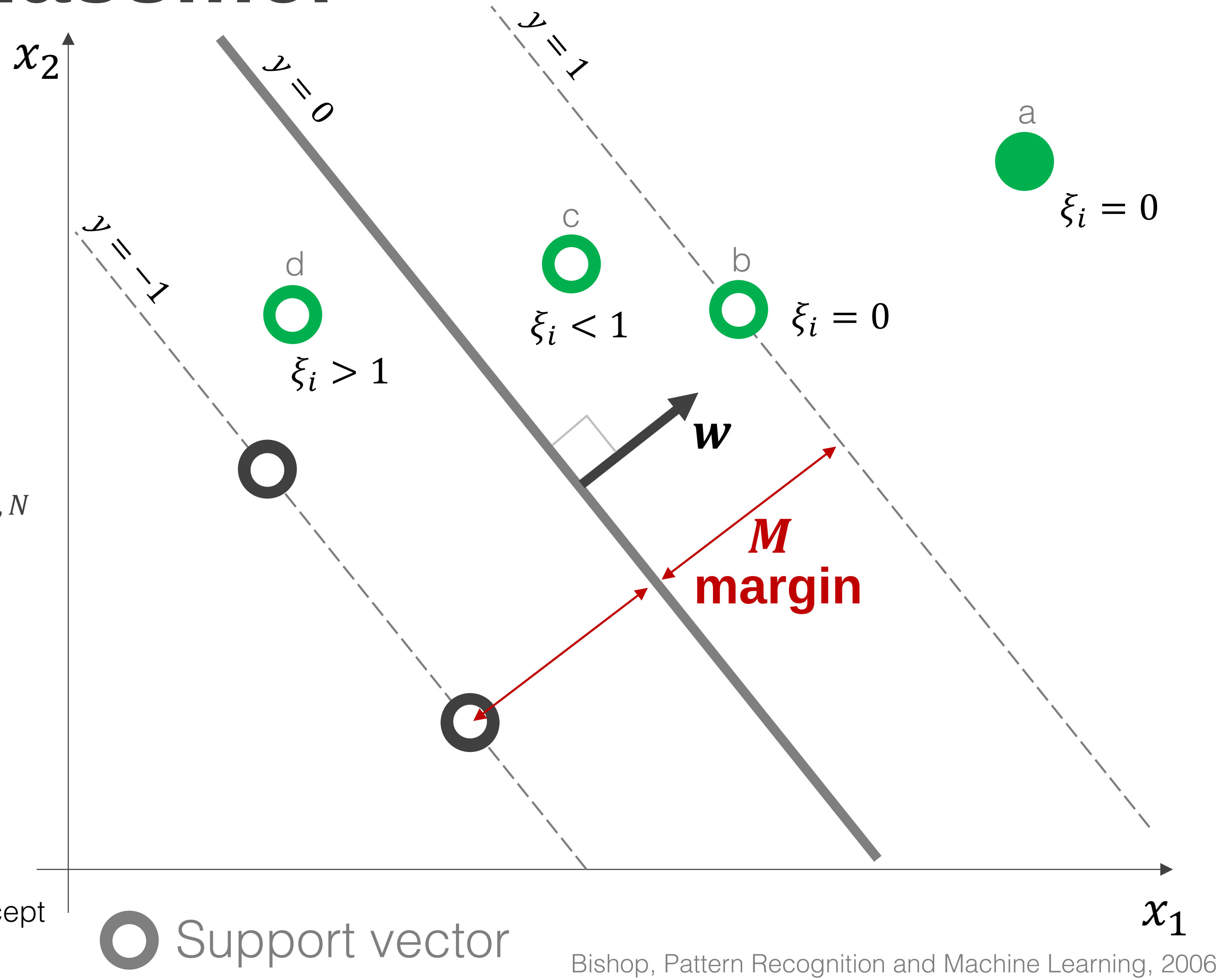
$$y_i \boldsymbol{w}^\top \boldsymbol{x}_i > M(1 - \xi_i)$$

for all $i = 1,2,\ldots,N$

(every training sample must be correctly classified and a distance M from the hyperplane, with the exception of the allowed margin violations)

$$\sum_{i=1}^{N} \xi_i \leq C \qquad \text{where } \xi_i \geq 0$$

(We fix the total amount of "slack" we're willing to allow)



C controls how much margin violation we are willing to accept and controls the bias-variance tradeoff for the SVC

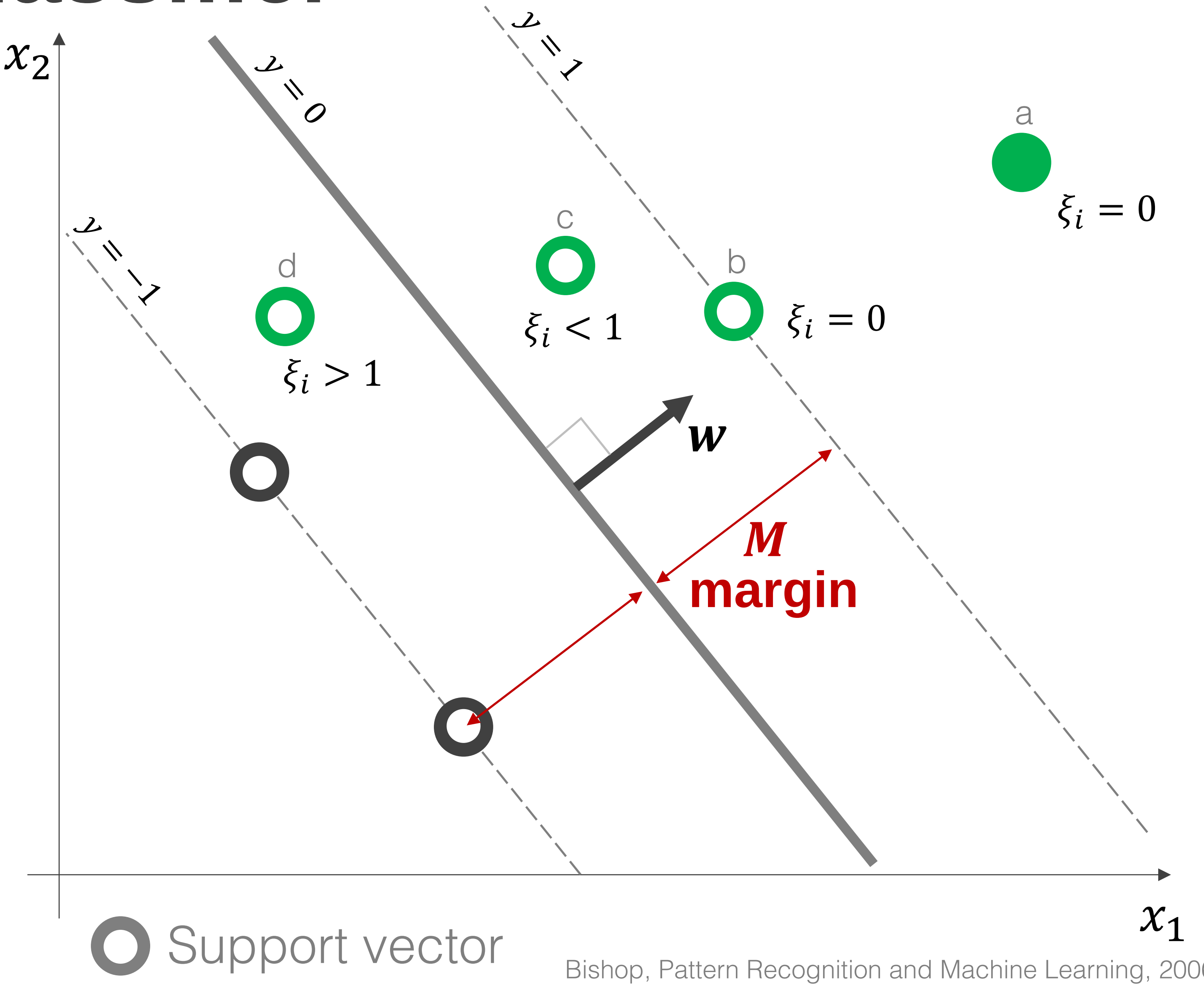
Bishop, Pattern Recognition and Machine Learning, 2006

Support vector classifier

Only the points at the margin or violating the margin affect the hyperplane

These are known as **support vectors**

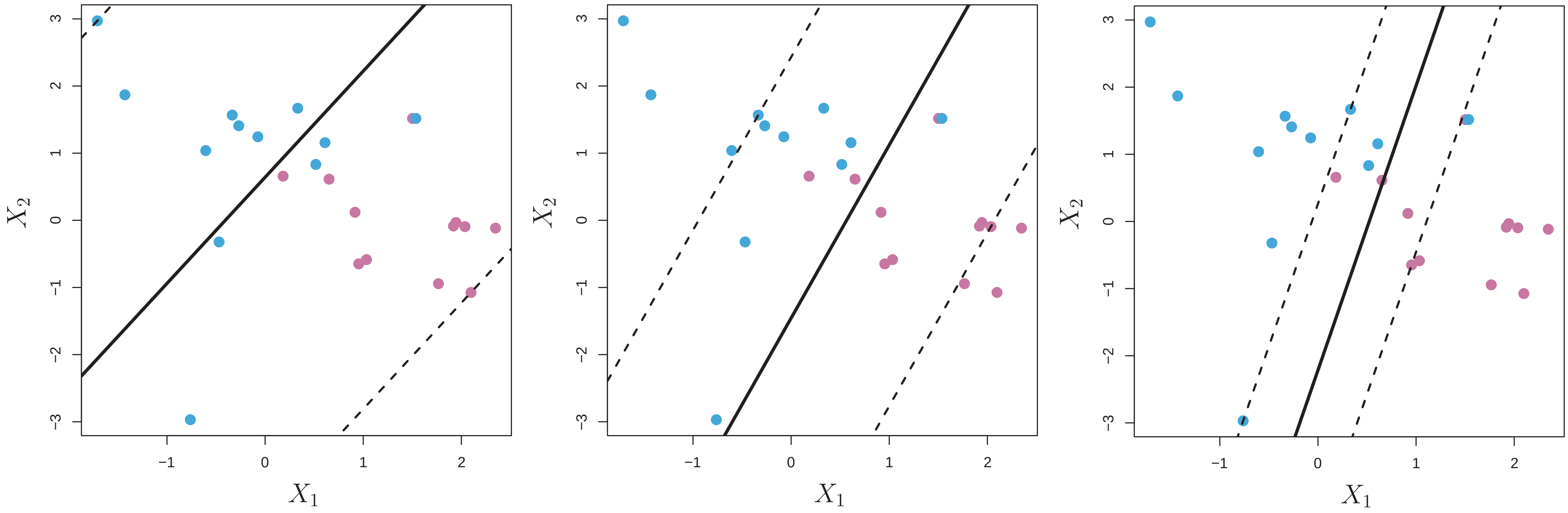
The more "slack" allowed, the wider the margin, the more points are support vectors

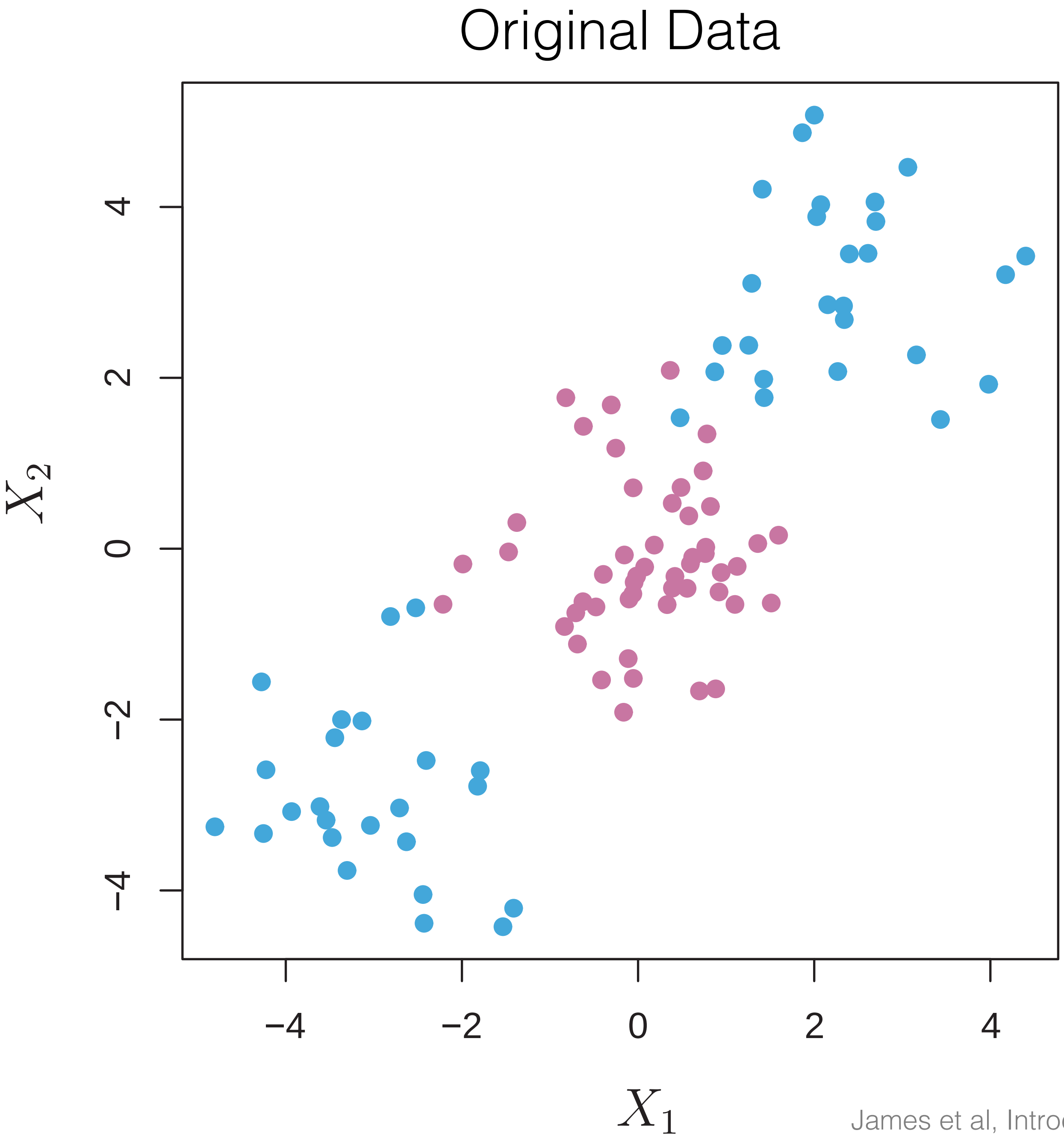


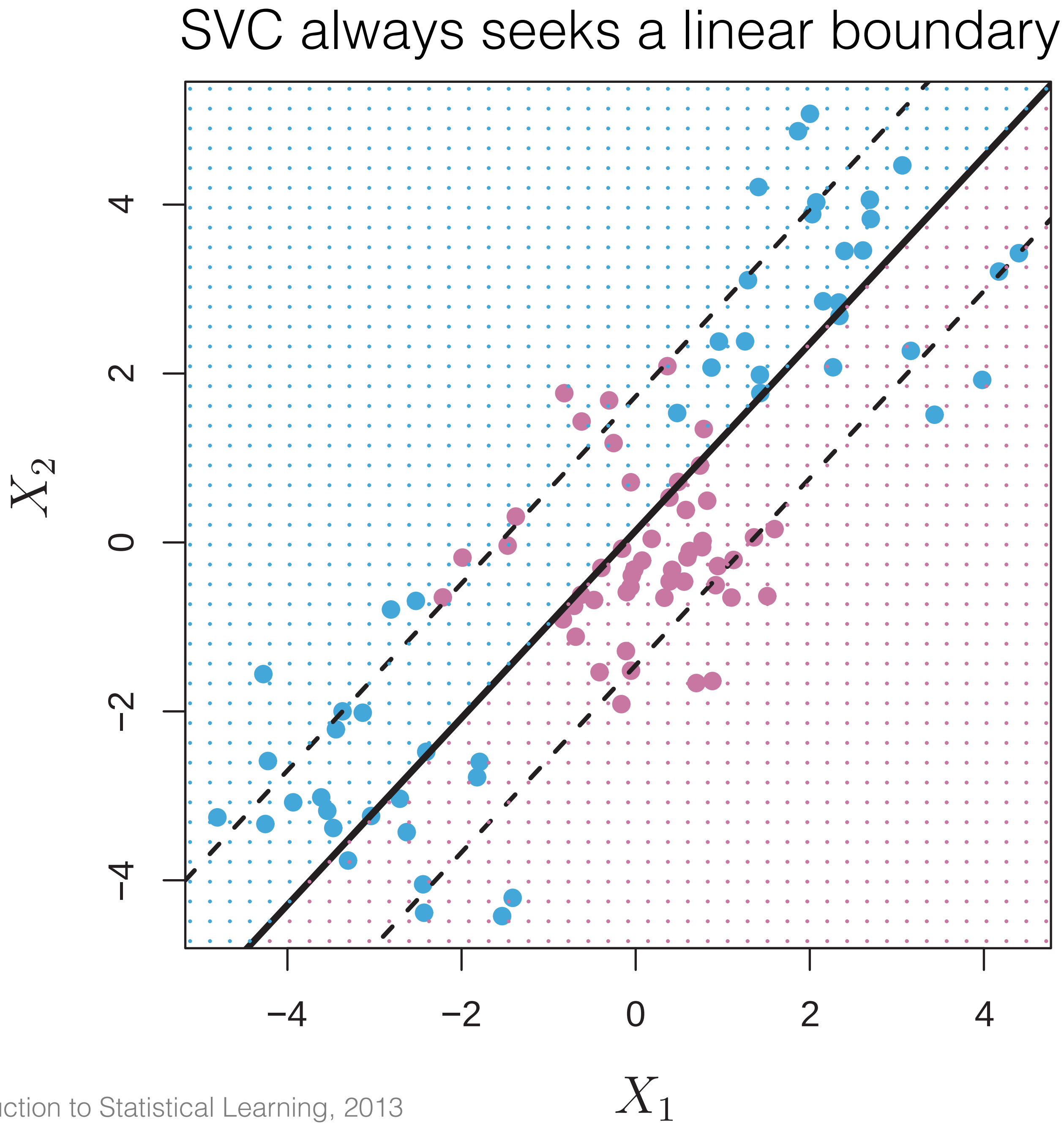
Bishop, Pattern Recognition and Machine Learning, 2006

SVC Margin Violation Slack (C)

large C margin violation allowance small C







James et al, Introduction to Statistical Learning, 2013

Support Vector Machines (SVMs) extend Support Vector Classifiers to be able to produce nonlinear decision boundaries

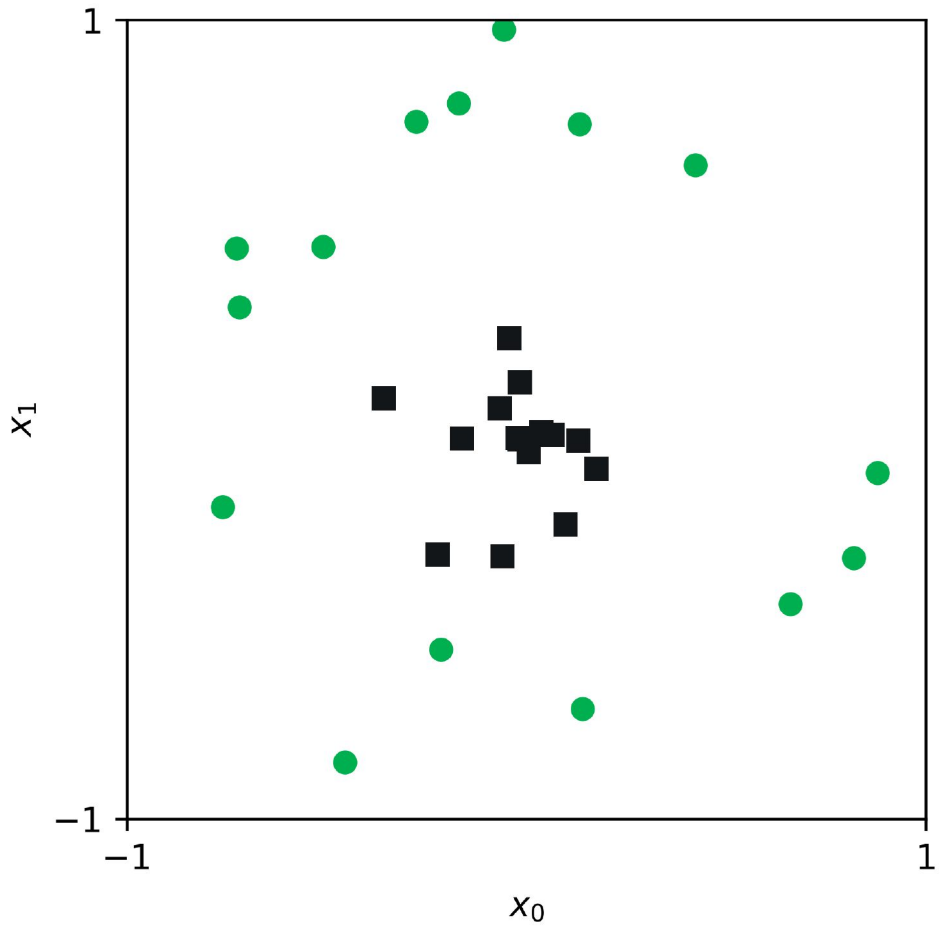
To understand SVMs, we need to understand kernels

What are kernel functions and why are they useful?

Limitations of linear decision boundaries

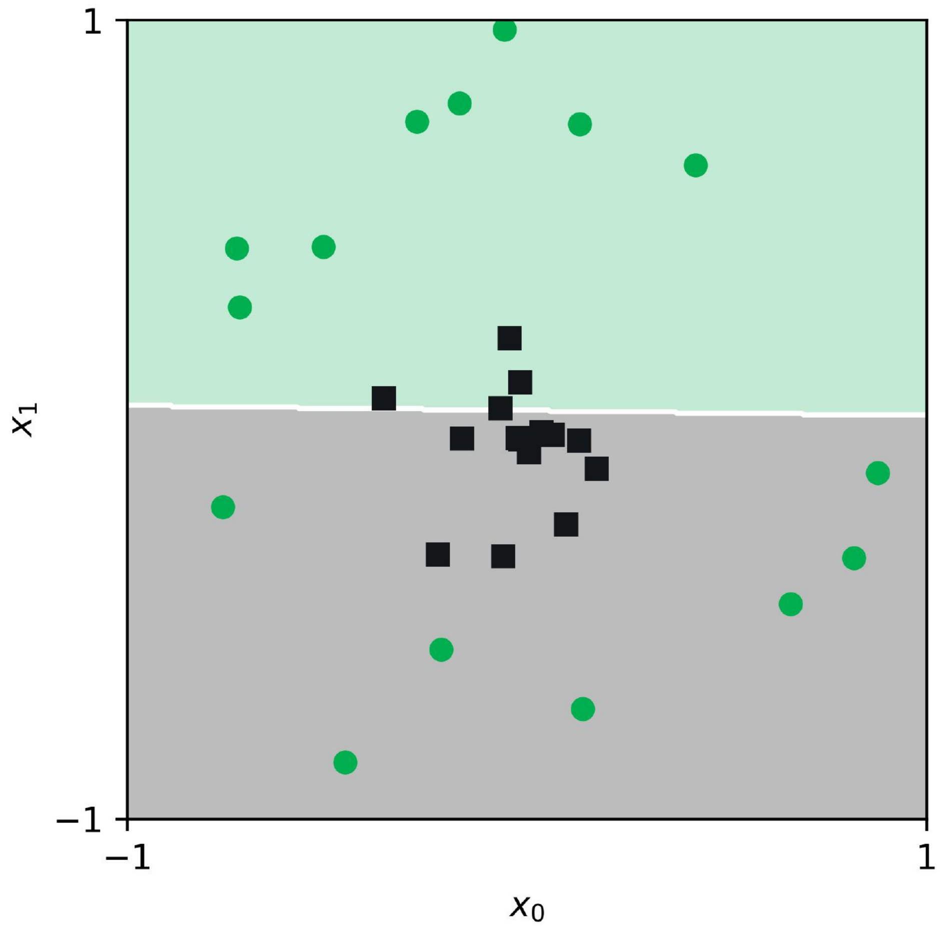
Original data

$$\boldsymbol{x}$$



Classify the features in this X-space

$$\hat{f}_{\boldsymbol{x}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x})$$



Explicit transformations of features

(data representations)

Recall the digits example…

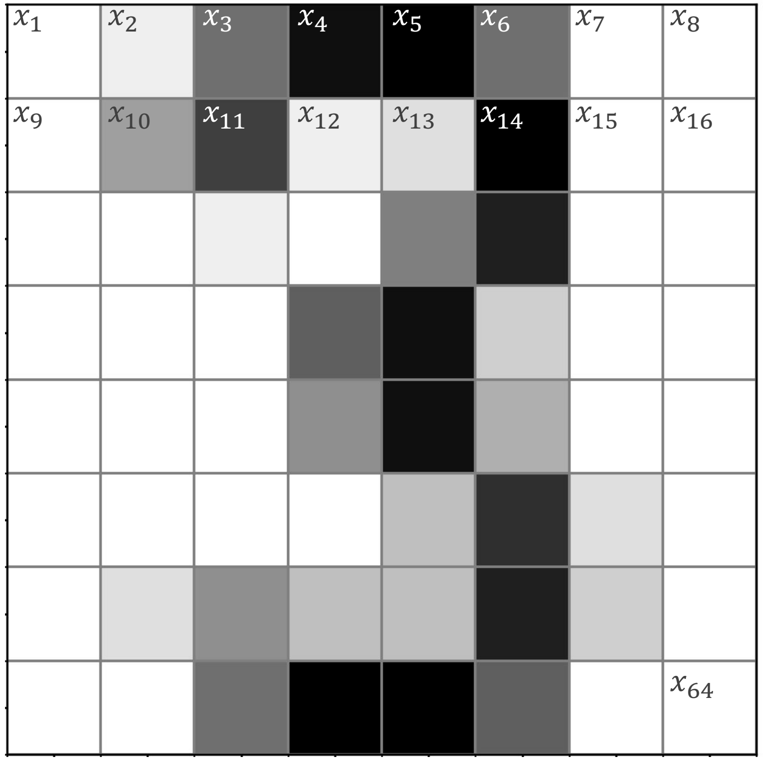
$$\boldsymbol{x} = [x_1, x_2, x_3, \dots, x_{64}]$$

We could **design features** based on the original features. For example:

$$\boldsymbol{z} = \left[x_5 x_{11}, x_{14}^2, \frac{x_{64}}{x_{14}}\right]$$

Which can be written simply as variables in a new feature space:

$$\boldsymbol{z} = [z_1, z_2, z_3]$$



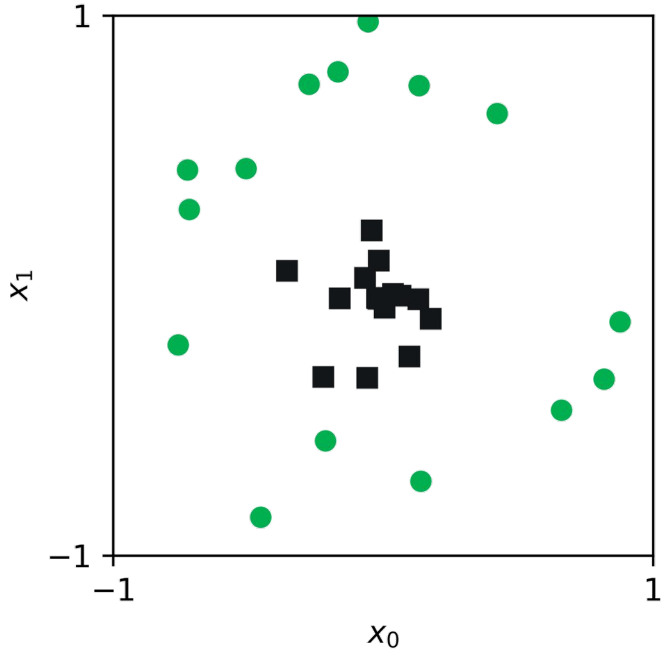
The new feature space could be smaller OR larger than the original

Source: Abu-Mostafa, Learning from Data, Caltech

1

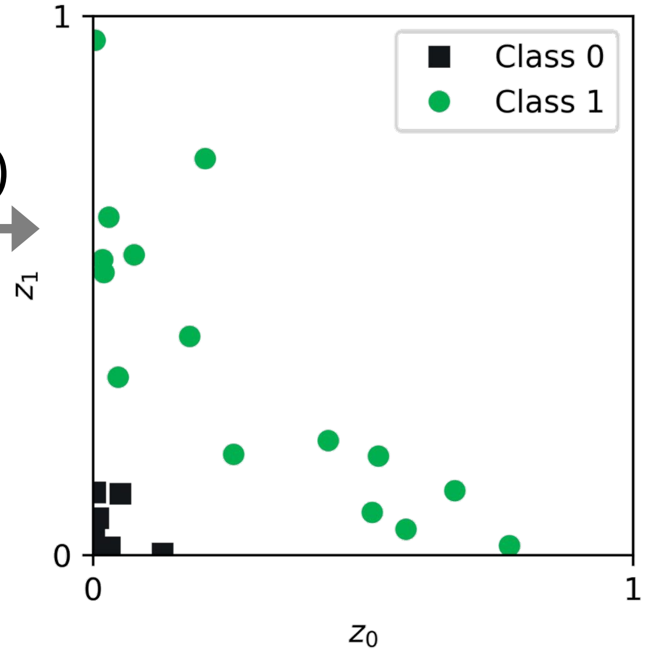
Original data

$\boldsymbol{x}$



transform
the data

$\boldsymbol{z} = \Phi(\boldsymbol{x})$



2

This example transform
is quadratic

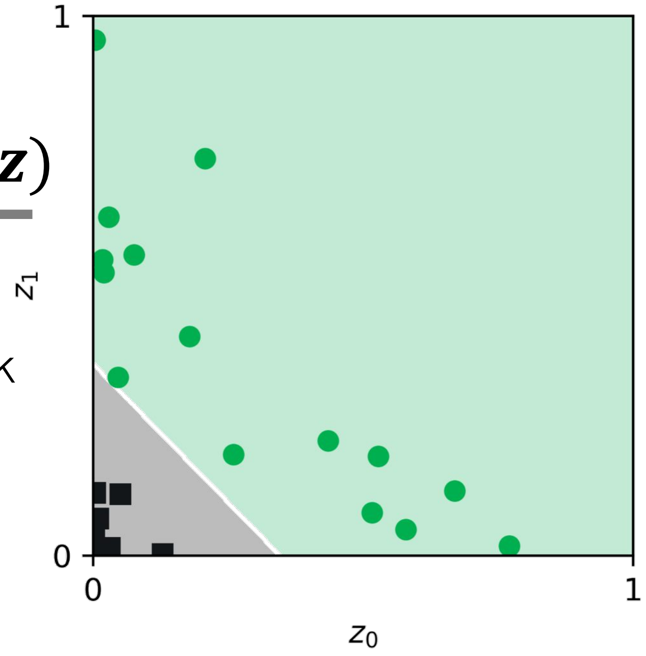
$z_i = \Phi(x_i) = x_i^2$

$z_0 = x_0^2$

$z_1 = x_1^2$

Classify the features
in this Z-space

$\hat{f}_z(\boldsymbol{z}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{z})$



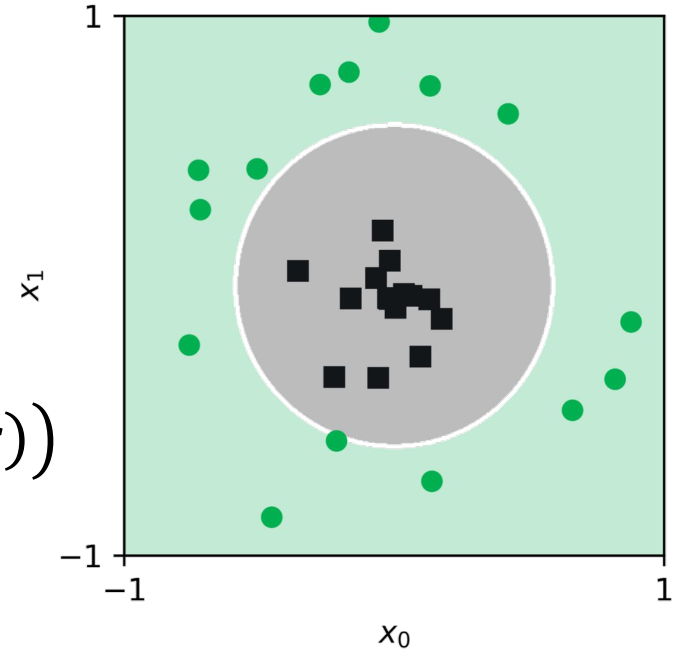
3

$\boldsymbol{x} = \Phi^{-1}(\boldsymbol{z})$

transform
the data back

$x_0 = z_0^{1/2}$

$x_1 = z_1^{1/2}$



Predictions in the
original X-space

$\hat{f}(\boldsymbol{x}) = \hat{f}_z(\Phi(\boldsymbol{x}))$

4

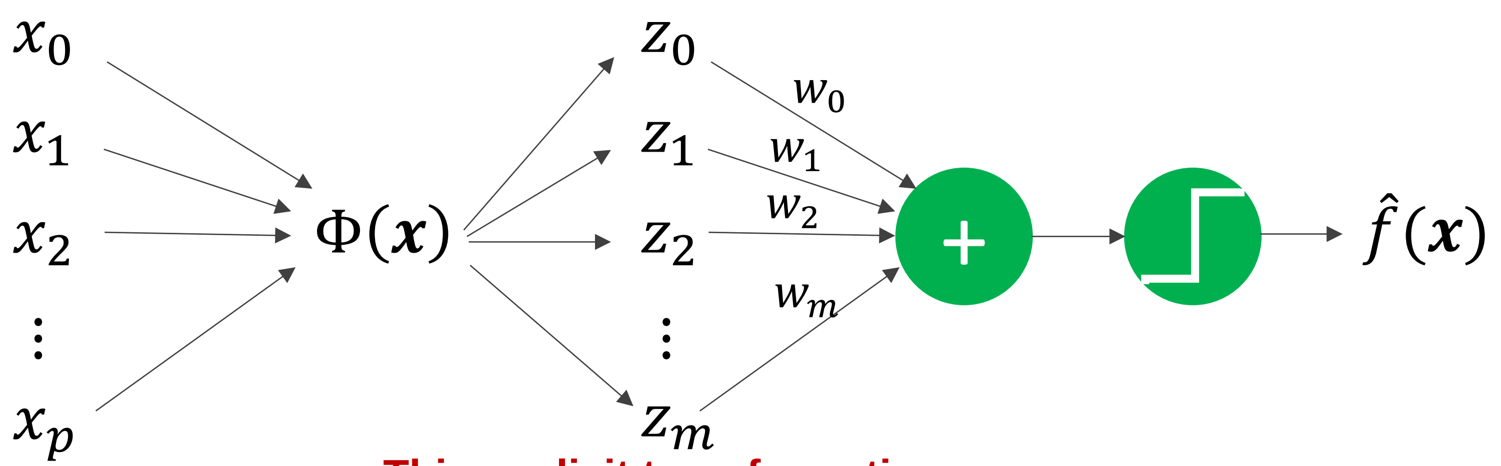
We can explicitly transform the feature space

Transform the feature space

$$\boldsymbol{z} = \Phi(\boldsymbol{x})$$

Linear Classifier

$$\hat{y} = \hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^\top \boldsymbol{z})$$



This explicit transformation can be expensive or impossible!

For example, a polynomial feature space

$$\boldsymbol{x} = [x_1 \quad x_2]^\mathsf{T}$$

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_2^2 \quad x_1 x_2]^\mathsf{T}$$

Transform into a 2nd-order polynomial feature space

This second order polynomial space with 2 features is simple enough

What about a 100th order polynomial space with 25 features?

That would be more than $\mathbf{10^{26}}$ terms!

(Not computationally feasible)

Transformations into alternative feature spaces may improve predictive performance

(better data representations)

Can be **computationally challenging** to compute the transformation into those feature spaces explicitly…

Solution: **kernel functions / the kernel trick**

Perform learning in the feature space without **explicitly** transforming features into it

Kernel function

Definition for kernel methods

Similarity measure between two points $\boldsymbol{x}$ and $\boldsymbol{x}$'

A **kernel function**, $K(\boldsymbol{x}, \boldsymbol{x}')$, represents an **inner product in some feature space**

$$\langle \boldsymbol{z}, \boldsymbol{z}' \rangle = \boldsymbol{z} \cdot \boldsymbol{z}' = \boldsymbol{z}^\top \boldsymbol{z}' \qquad \boldsymbol{z} = \Phi(\boldsymbol{x})$$

for Euclidean spaces

For a valid kernel, there is some feature transformation, $\boldsymbol{z} = \Phi(\boldsymbol{x})$, where:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^\top \boldsymbol{z}$$

Simplest example: the linear kernel $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$

Kernel function example

$$\boldsymbol{x} = [x_1 \quad x_2]^\top$$

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_2^2 \quad x_1 x_2]^\top$$

Transform into a 2nd-order polynomial feature space

The kernel function is:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^\top \boldsymbol{z}' = 1 + x_1 x_1' + x_2 x_2' + x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + x_1 x_1' x_2 x_2'$$

We haven't gained anything yet…
We want to compute $K(\boldsymbol{x}, \boldsymbol{x}')$ without the explicit $\boldsymbol{z} = \Phi(\boldsymbol{x})$ feature space transformation:

Kernel Trick

Source: Abu-Mostafa, Learning from Data, Caltech

Kernel trick

$\boldsymbol{x} = [x_1 \quad x_2]^\top$

Compute $K(\boldsymbol{x}, \boldsymbol{x}')$ without the $\boldsymbol{z} = \Phi(\boldsymbol{x})$ feature space transformation

Example:

$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^2$ This is **not** an inner product in X-space

$$= (1 + x_1 x_1' + x_2 x_2')^2$$

$$= 1 + x_1 x_1' + x_2 x_2' + 2 x_1^2 {x_1'}^2 + 2 x_2^2 {x_2'}^2 + 2 x_1 x_1' x_2 x_2'$$

Similar to the inner product for: $\boldsymbol{z} = \Phi(\boldsymbol{x}) = [1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_2^2 \quad x_1 x_2]^\top$

It **IS an inner product** in a **different** Z-space:

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) = \left[1 \quad x_1 \quad x_2 \quad \sqrt{2} x_1^2 \quad \sqrt{2} x_2^2 \quad \sqrt{2} x_1 x_2\right]^\top$$

$$K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{z}^\top \boldsymbol{z}'$$

Computing

$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^2$

Is much easier than the full Z-space transform. Imagine if this was $(1 + \boldsymbol{x}^\top \boldsymbol{x}')^{100}$!

Source: Abu-Mostafa, Learning from Data, Caltech

Common kernel functions

Linear kernel: $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$

Polynomial kernels: $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^d$
(all polynomials up to degree d)

Radial basis function kernel: $K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right)$
(infinite dimensional)

For an excellent explanation of how this is infinite dimensional, see [Yaser Abu-Mostafa's explanation]

Kernel function properties

Symmetric: $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}', \boldsymbol{x})$

All kernels are symmetric

Stationary kernels: $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x} - \boldsymbol{x}')$

Invariant to translation in the input space
Only a function of the difference between arguments

Homogeneous kernels: $K(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{K}(||\boldsymbol{x} - \boldsymbol{x}'||)$

Depend only on the magnitude of the distance between arguments

How do we use kernels in classification?

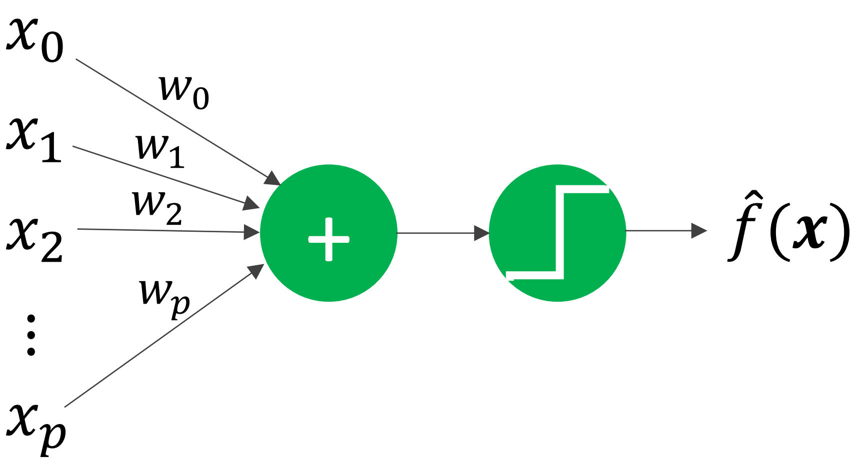
We'll build the infrastructure we need with the kernel perceptron then use that to explain how kernels extend SVCs into SVMs

Perceptron classifier

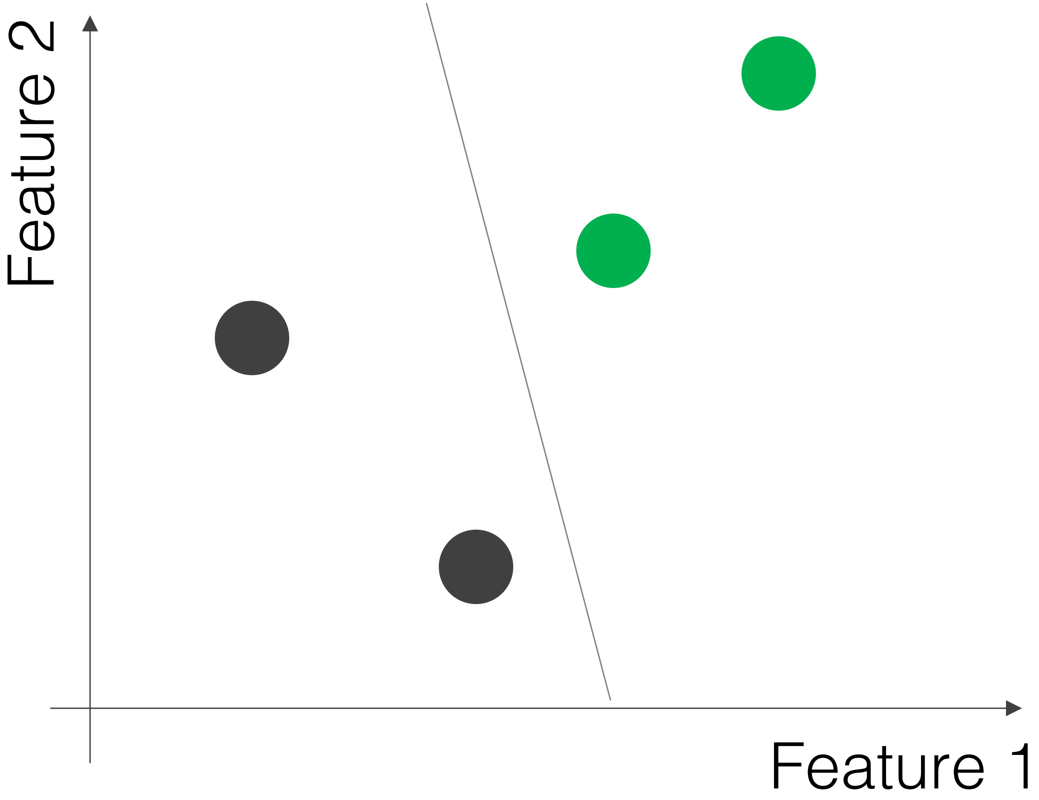
Linear Classification
(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$= sign(\boldsymbol{w}^\top \boldsymbol{x})$$



Idea: draw a line (hyperplane) that separates the classes



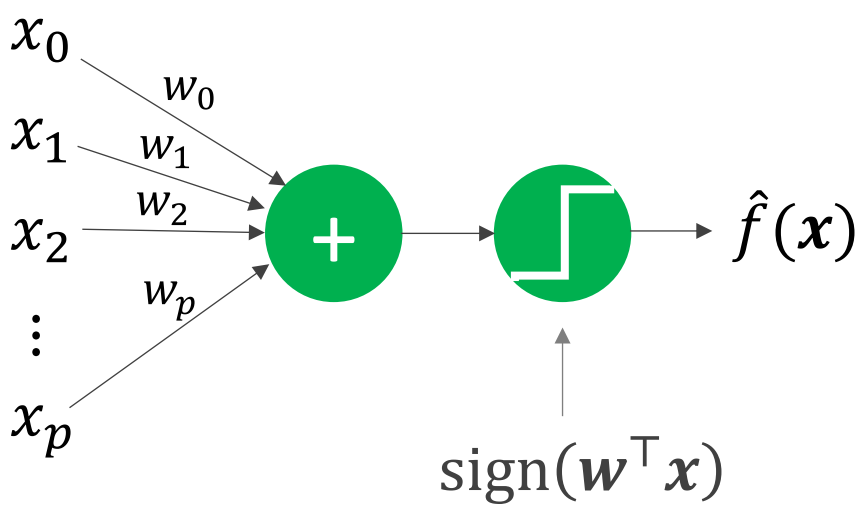
Source: Abu-Mostafa, Learning from Data, Caltech

Linear classifier

Linear Classification

$$\hat{f}(\boldsymbol{x}) = f\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$= f(\boldsymbol{w}^\top \boldsymbol{x})$$



Training data: $(\boldsymbol{x}_i, y_i), \quad i = 1, \ldots, N$
with binary $y_i = \{-1,1\}$

Decision rule based on $sign(\boldsymbol{w}^T \boldsymbol{x})$:
if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$
if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$

Source: Abu-Mostafa, Learning from Data, Caltech

The separating hyperplane

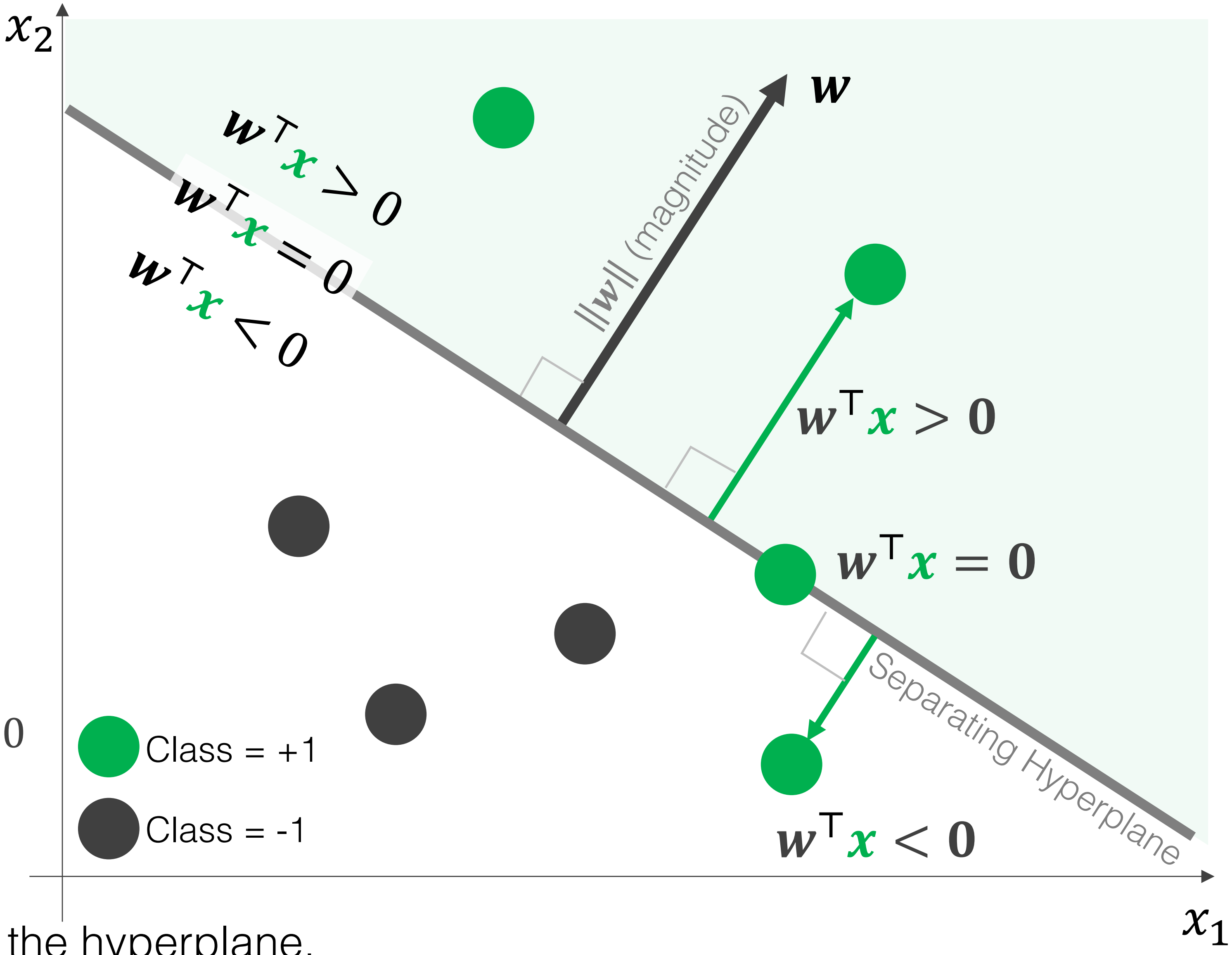
$\boldsymbol{w}$ defines and is orthogonal to the separating hyperplane

$$\hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^\top \boldsymbol{x})$$

Decision rule based on $sign(\boldsymbol{w}^T \boldsymbol{x})$:

if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$
if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$



Interpretation: if a point is on one side of the hyperplane, assign one class, if it's on the other, assign the other class

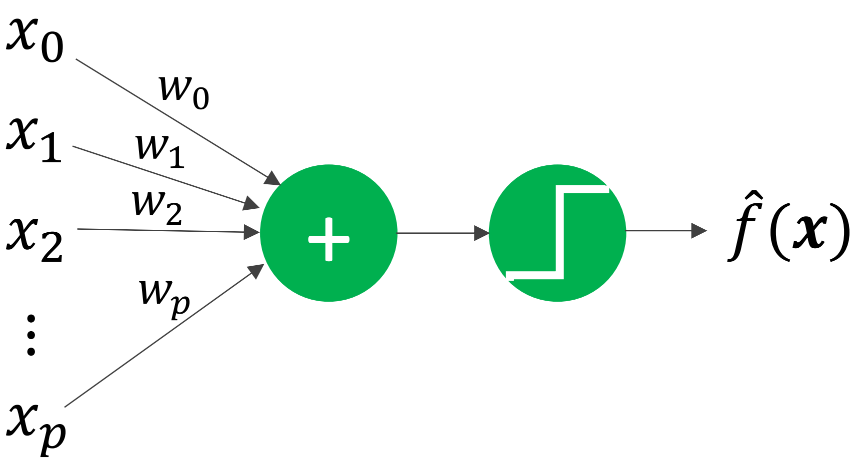
We constrain $\|\boldsymbol{w}\| = 1$

Perceptron classifier

Linear Classification
(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$= sign(\boldsymbol{w}^\top \boldsymbol{x})$$



Training data: $(\boldsymbol{x}_i, y_i), \quad i = 1, \dots, N$
with binary $y_i = \{-1,1\}$

Decision rule based on $sign(\boldsymbol{w}^\top \boldsymbol{x})$:
if $\boldsymbol{w}^\top \boldsymbol{x}_i > 0$, then $\hat{y}_i = +1$
if $\boldsymbol{w}^\top \boldsymbol{x}_i < 0$, then $\hat{y}_i = -1$

For correctly classified points: $y_i \boldsymbol{w}^\top \boldsymbol{x}_i > 0$

Our cost (error) function to minimize:

$$C = -\sum_{\substack{i \in \{\text{mistakes}\} \\ \hat{y}_i \neq y_i}} y_i \boldsymbol{w}^\top \boldsymbol{x}_i$$

Source: Abu-Mostafa, Learning from Data, Caltech

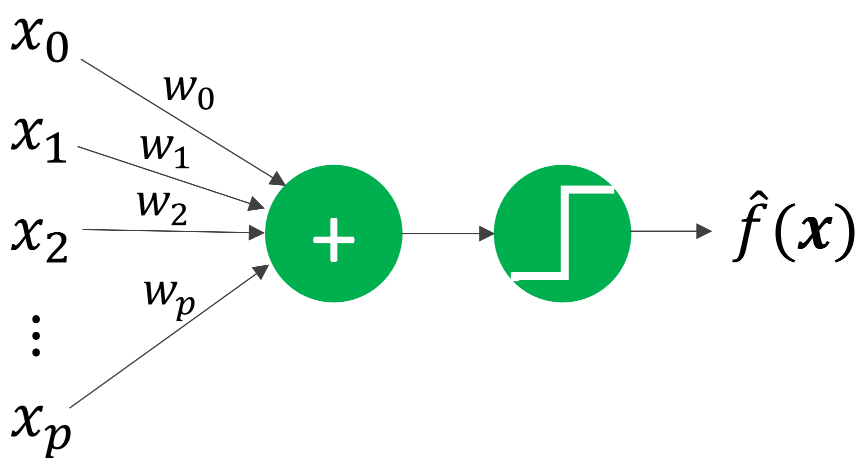
Perceptron classifier

Linear Classification

(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$= sign(\boldsymbol{w}^{\top}\boldsymbol{x})$$



Our cost (error) function to minimize:

$$C = -\sum_{i \in \{\text{mistakes}\}} y_i \boldsymbol{w}^{\top} \boldsymbol{x}_i$$

The gradient with respect to $\boldsymbol{w}$:

$$\frac{\partial C}{\partial \boldsymbol{w}} = -\sum_{i \in \{\text{mistakes}\}} y_i \boldsymbol{x}_i$$

Applying stochastic gradient:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial C}{\partial \boldsymbol{w}}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$$

process one mistake at a time and assume a learning rate of 1

Source: Abu-Mostafa, Learning from Data, Caltech

Perceptron Learning Algorithm

Note: this algorithm assumes the classes are linearly separable

1. Pick a misclassified point and use it to update the weights:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$$

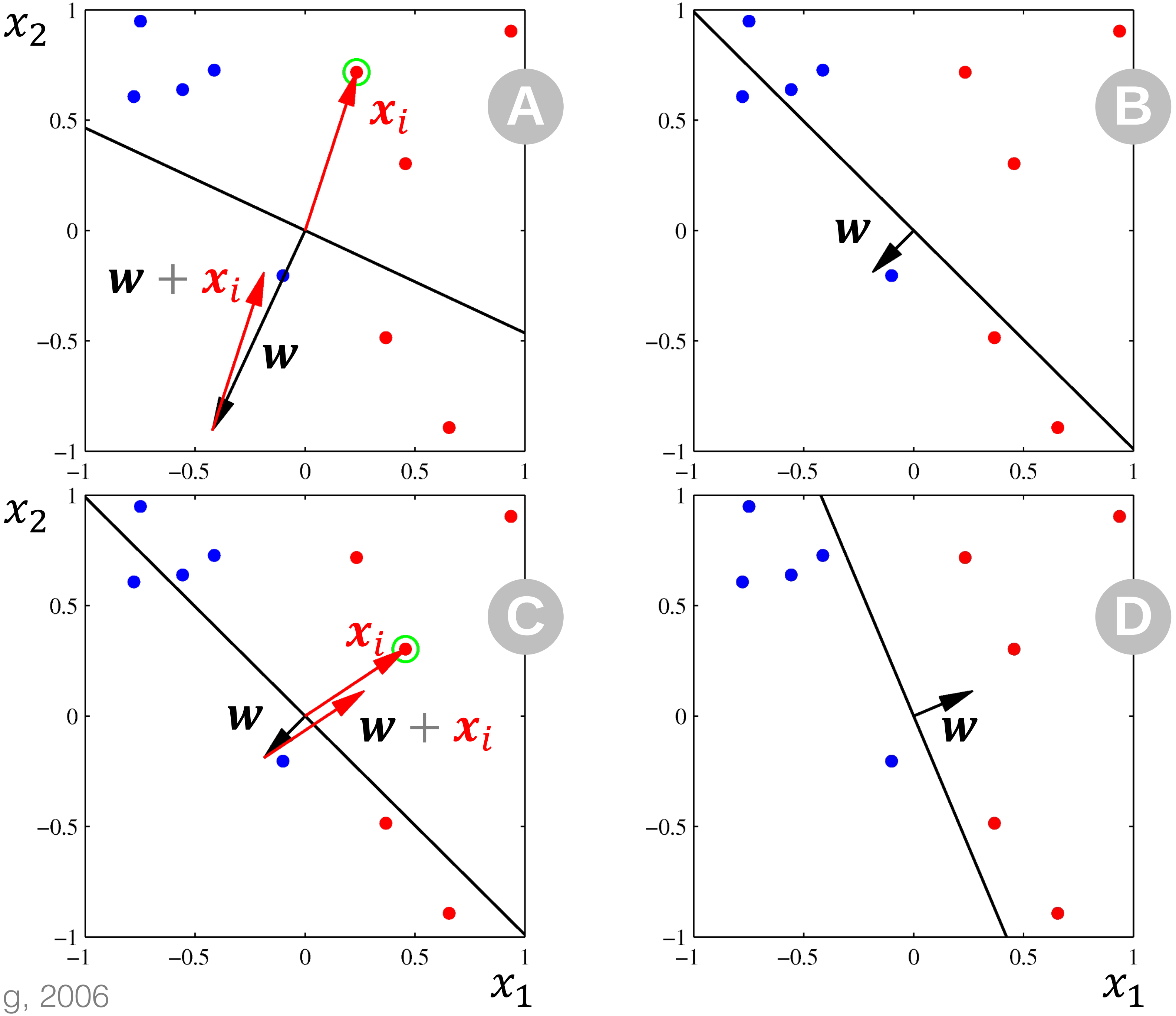
$$a_i \leftarrow a_i + 1$$

(mistake counter)

2. Reclassify all the data:

$$\hat{y}_i = sign(\boldsymbol{w}^\top \boldsymbol{x}_i)$$

3. Repeat until no mistakes



Bishop, Pattern Recognition and Machine Learning, 2006

Perceptron Learning Algorithm

Note: this algorithm assumes the classes are linearly separable

Update weights

$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$

$a_i \leftarrow a_i + 1$

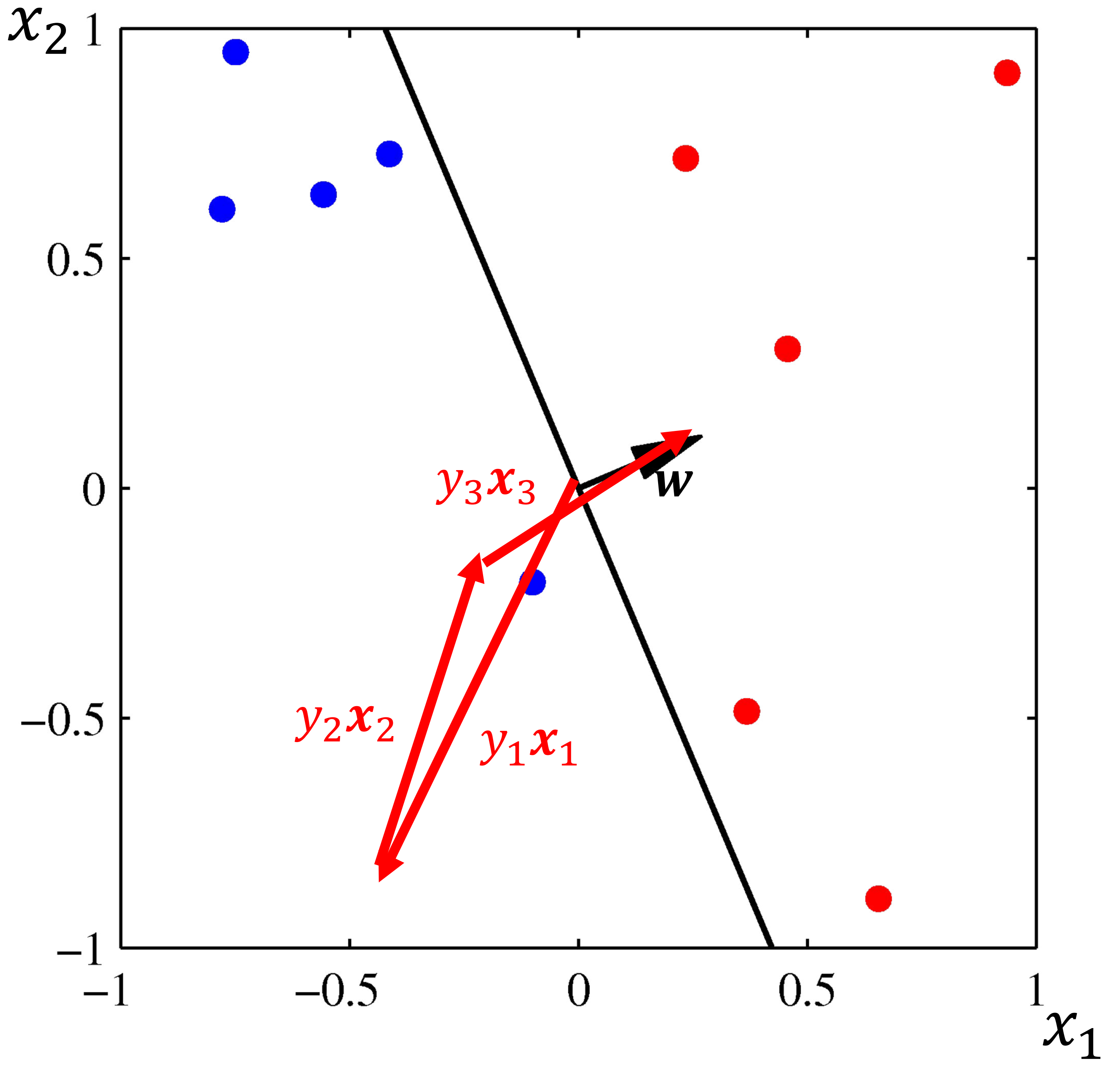
(mistake counter)

We can rewrite an expression for our weights:

$$\boldsymbol{w} = \sum_i a_i y_i \boldsymbol{x}_i$$

If we store our mistake counter, we can update our weights as a sum over all observations, but only the mistakes that were considered will have a nonzero value for a_i

x_2



Bishop, Pattern Recognition and Machine Learning, 2006

Perceptron Learning Algorithm (towards kernels)

Update weights

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$$

$$a_i \leftarrow a_i + 1$$

(mistake counter)

We can rewrite an expression for our weights:

$$\boldsymbol{w} = \sum_i a_i y_i \boldsymbol{x}_i$$

If we store our mistake counter, we can update our weights as a sum over all observations, but only the mistakes that were considered will have a nonzero value for a_i

Let's plug this new expression into our classifier:

$$\hat{y} = \hat{f}(\boldsymbol{x}) = sign(\boldsymbol{w}^{\top}\boldsymbol{x})$$

$$= sign\left(\left(\sum_i a_i y_i \boldsymbol{x}_i\right)^{\top} \boldsymbol{x}\right)$$

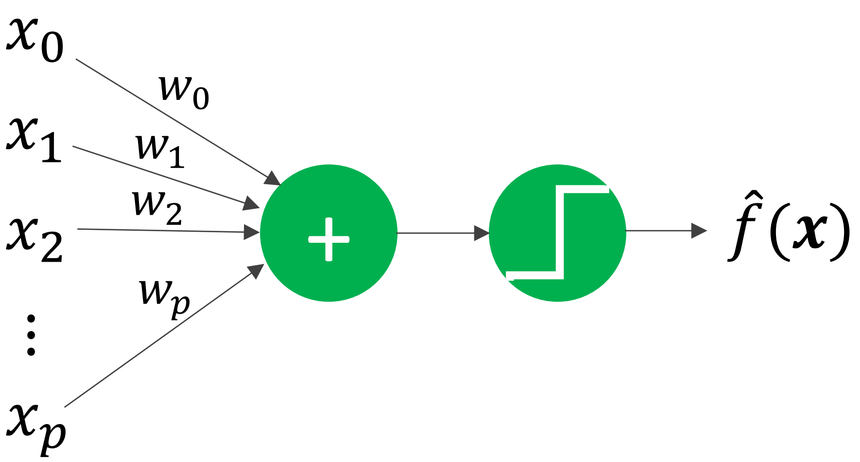
$$= sign\left(\sum_i a_i y_i \underbrace{\boldsymbol{x}_i^{\top} \boldsymbol{x}}_{\text{inner product}}\right)$$

Our classifier **stores training data**, but it only depends on **inner products**

Kernel perceptron classifier

Linear Classification
(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_i a_i y_i \boldsymbol{x}_i^\top \boldsymbol{x}\right)$$



Our classifier **stores training data**, but it only depends on an **inner product**

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_i a_i y_i \boldsymbol{x}_i^\top \boldsymbol{x}\right)$$

We can write this inner product as a **kernel function**, $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_i a_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})\right)$$

We can replace this with **any valid kernel**

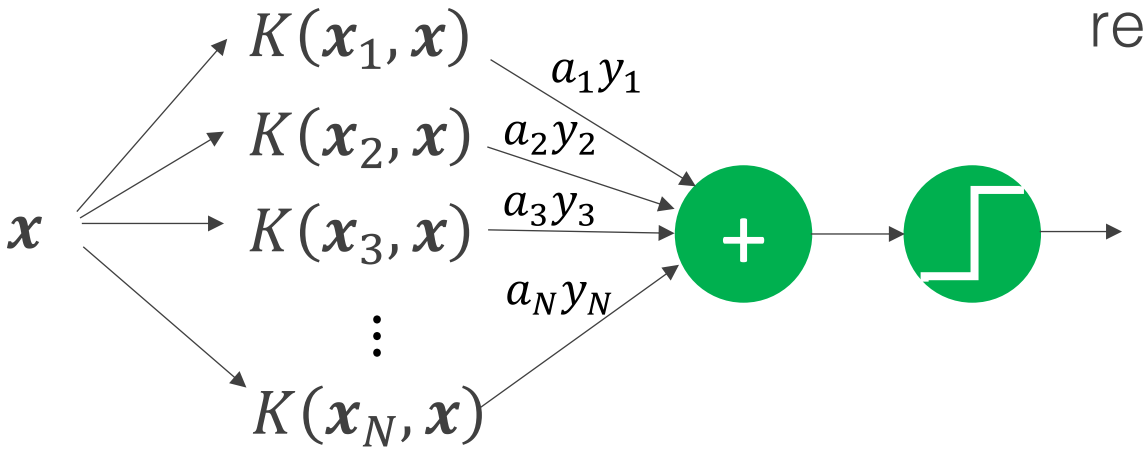
Source: Abu-Mostafa, Learning from Data, Caltech

Kernel perceptron classifier

No need to explicitly transform the feature space

$$\cancel{\boldsymbol{z} = \Phi(\boldsymbol{x})}$$

We only need the kernel function



$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_i a_i y_i K(\boldsymbol{x}_i, \boldsymbol{x})\right)$$

Now we need to store our training data

We have to use lots of training data in each prediction BUT if we use this with the SVC we get the SVM which only relies on the support vectors

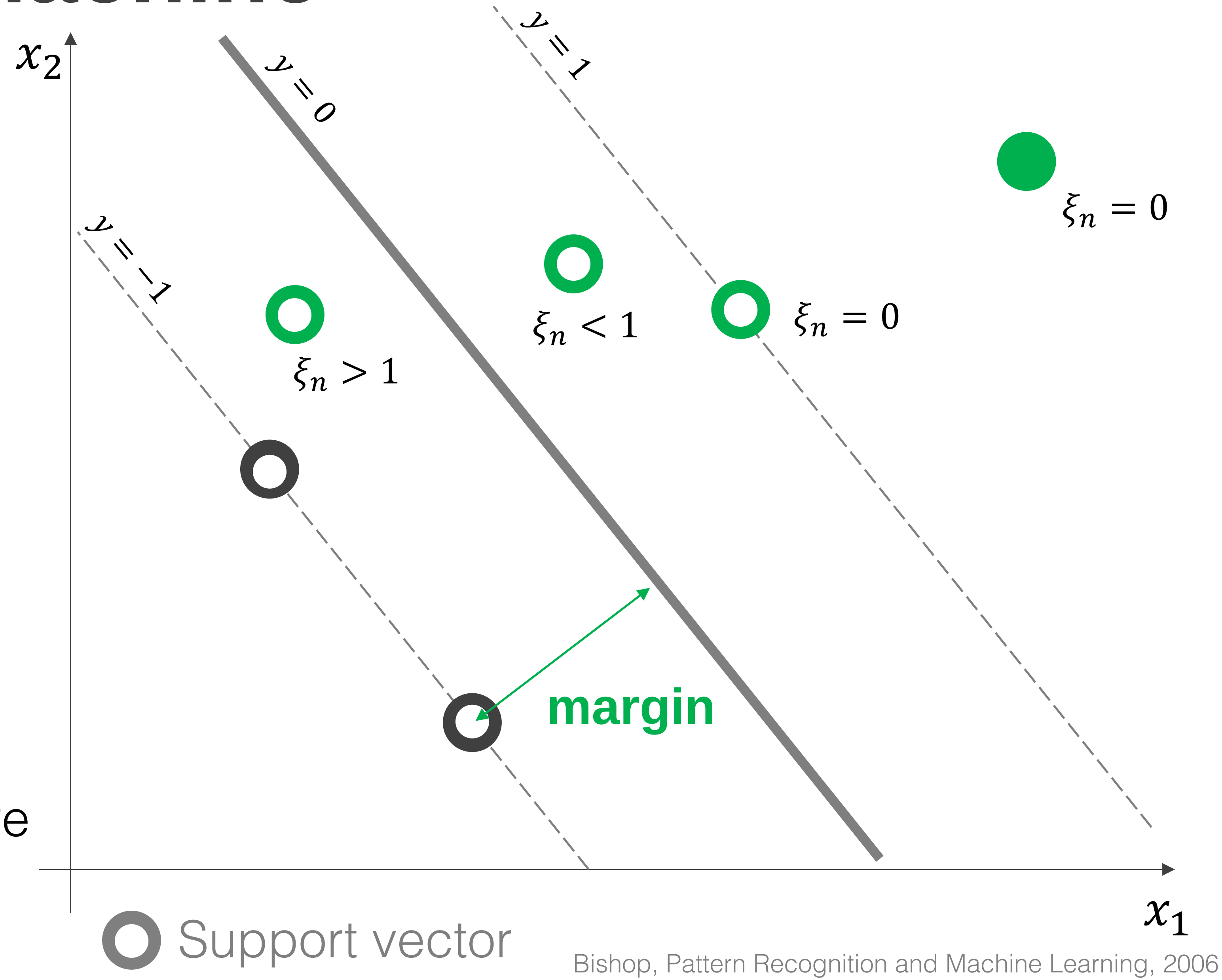
Support vector machine

Penalty term for violating the margin: ξ_n

The SVM is an SVC that uses a kernel function to implicitly transform the feature space

Pick $\boldsymbol{w}$ to define a decision boundary (hyperplane) and maximize the margin in the implicit feature space (the one provided by the **kernel trick**)

Does not assume linear separability in the original feature space



Bishop, Pattern Recognition and Machine Learning, 2006

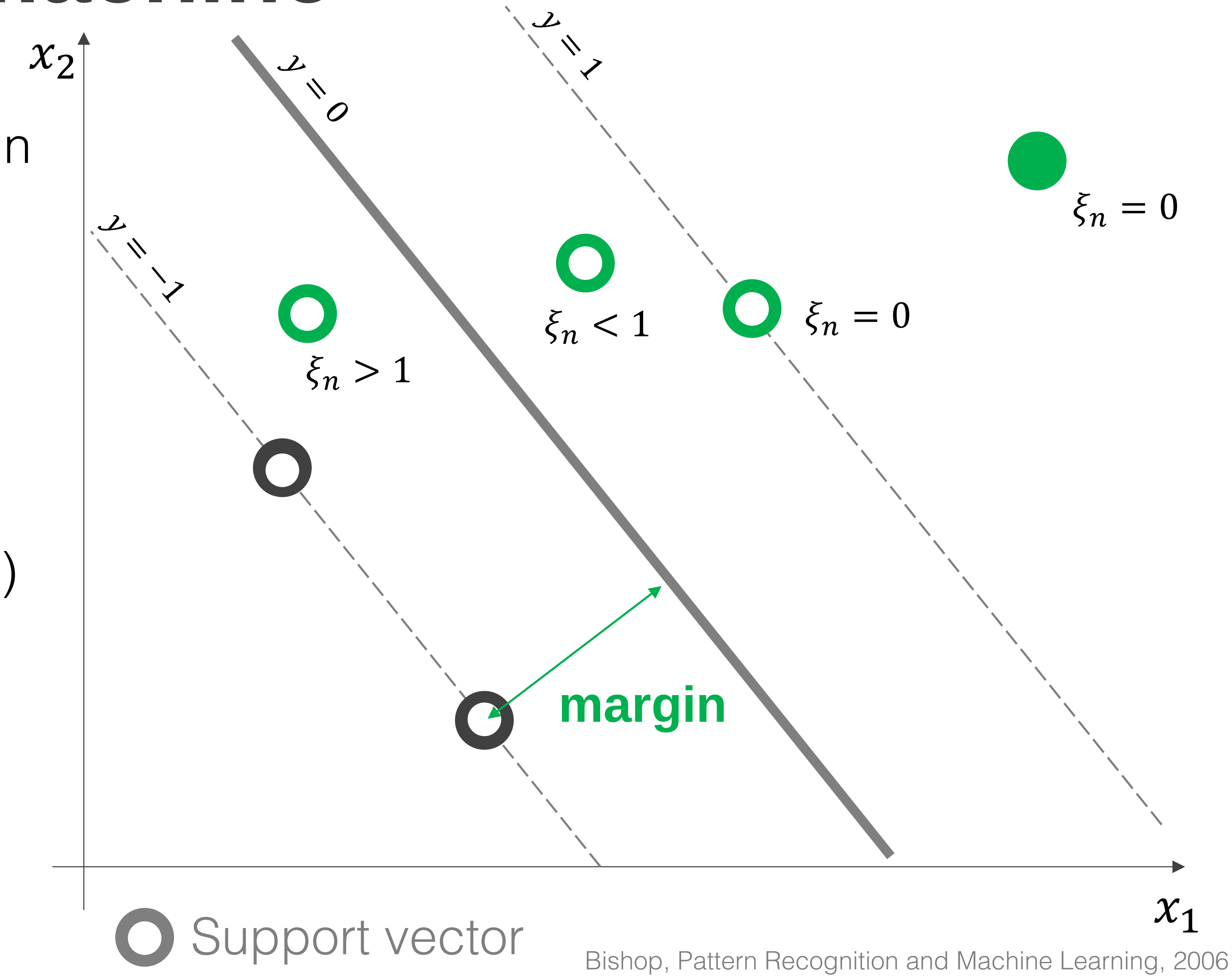
Support vector machine

Penalty term for violating the margin: ξ_n

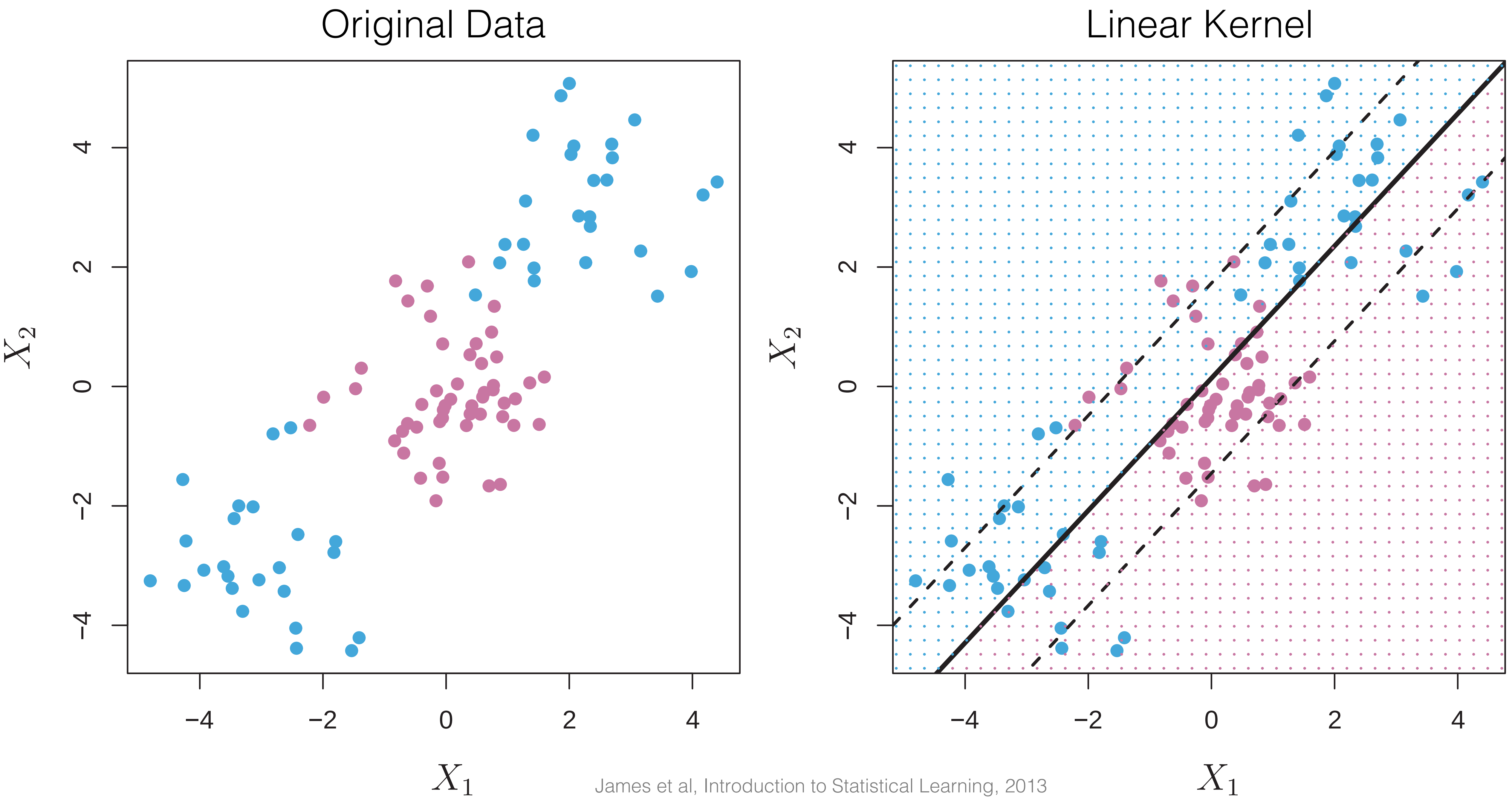
Use the **kernel trick** to classify in other feature spaces

Sparse kernel machine

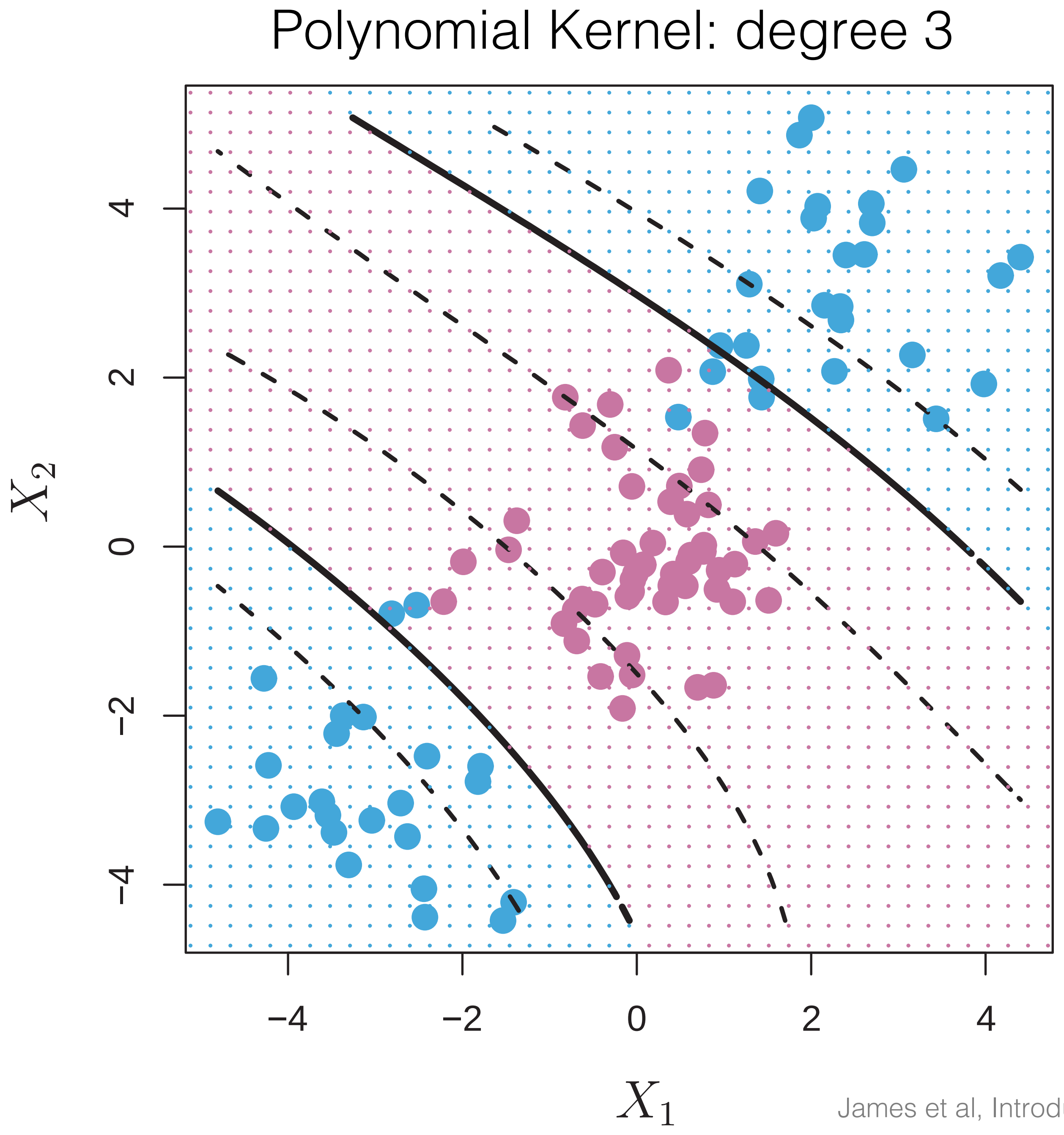
Prediction: kernel comparisons with weighted support vectors (similar to the kernel perceptron)

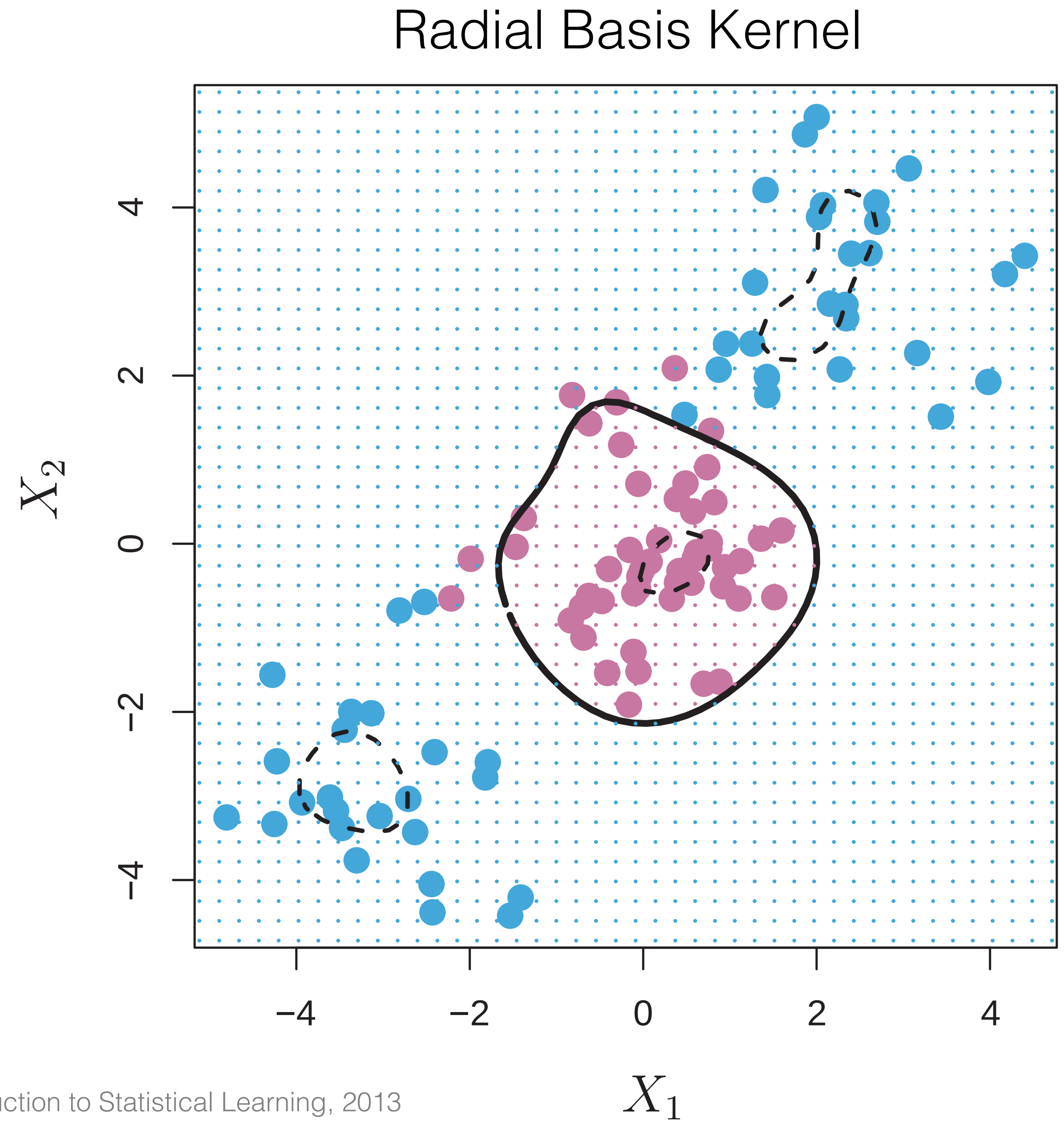


Bishop, Pattern Recognition and Machine Learning, 2006



James et al, Introduction to Statistical Learning, 2013





James et al, Introduction to Statistical Learning, 2013

Produces “sparse” models

Kernel trick allows otherwise impossible computation in higher dimensional feature spaces in tractable ways

Need to select a “good” kernel for the method to work

Large datasets require significant training time

Model interpretability is low

Support Vector Machine

Bases the decision boundary on a subset of its **training examples** and produces sparse models

Can operate in **implicit** **alternative feature spaces** without explicitly transforming the data into that space

Relies on a similarity measure, the **kernel function**, to compare test points to the training data

Supervised Learning Techniques

- Linear Regression
- K-Nearest Neighbors
- Perceptron
- Logistic Regression
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- Naïve Bayes
- Decision Trees and Random Forests
- Ensemble methods (bagging, boosting, stacking)
- Support Vector Machines

Appropriate for:
- Classification
- Regression

Can be used with many machine learning techniques