Support Vector Machines

Mark Craven and David Page
Computer Sciences 760
Spring 2019

Goals for the lecture

you should understand the following concepts

- the margin
- slack variables
- the linear support vector machine
- nonlinear SVMs
- the kernel trick
- the primal and dual formulations of SVM learning
- support vectors
- the kernel matrix
- valid kernels
- polynomial kernel
- Gausian kernel
- string kernels
- support vector regression

Algorithm You Should Know

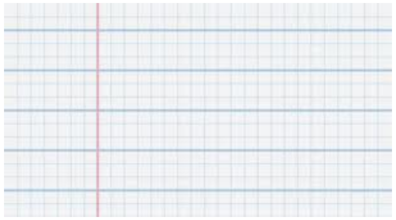
- SVM as constrained optimization
  - Fit training data and maximize margin
  - Primal and dual formulations
  - Kernel trick
  - Slack variables to allow imperfect fit
  - For now, assuming optimizer is black box

- Next lecture will look inside black box: sequential minimal optimization (SMO)

Burr Settles, UW CS PhD

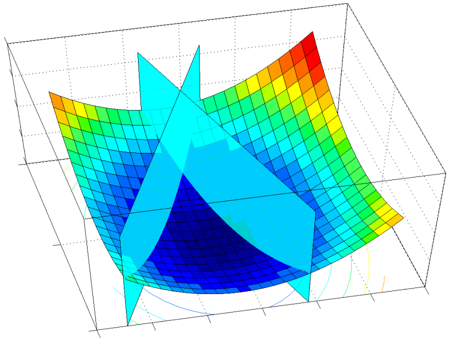
Four key SVM ideas

- Maximize the margin
  - don't choose just *any* separating hyperplane



- Penalize misclassified examples
  - use soft constraints and slack variables

- Use optimization methods to find model
  - linear programming
  - quadratic programming



- Use kernels to represent nonlinear functions and handle complex instances (sequences, trees, graphs, etc.)

Some key vector concepts

the *dot product* between two vectors $\boldsymbol{w}$ and $\boldsymbol{x}$ is defined as:

$$\boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x} = \sum_i w_i x_i$$

for example

$$\begin{bmatrix} 1 \\ 3 \\ -5 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -2 \\ -1 \end{bmatrix} = (1)(4) + (3)(-2) + (-5)(-1) = 3$$

the 2-norm (Euclidean length) of a vector $\boldsymbol{x}$ is defined as:

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_i |x_i|^2}$$

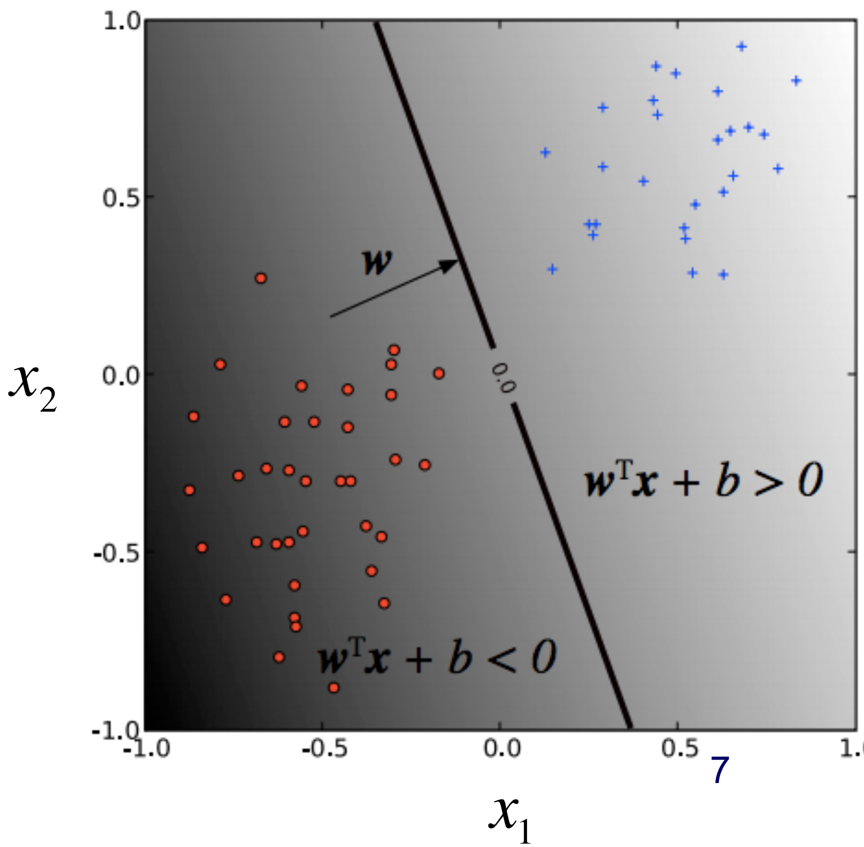
Linear separator learning revisited

suppose we encode our classes as $\{-1, +1\}$ and consider a linear classifier

$$h(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \left(\sum_{i=1}^{n} w_i x_i \right) + b > 0 \\ -1 \text{ otherwise} \end{cases}$$

an instance $\langle \boldsymbol{x}, y \rangle$ will be classified correctly if

$$y(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x} + b) > 0$$



Large margin classification

- Given a training set that is linearly separable, there are infinitely many hyperplanes that could separate the positive/negative instances.
  - Which one should we choose?
- In SVM learning, we find the hyperplane that maximizes the margin.

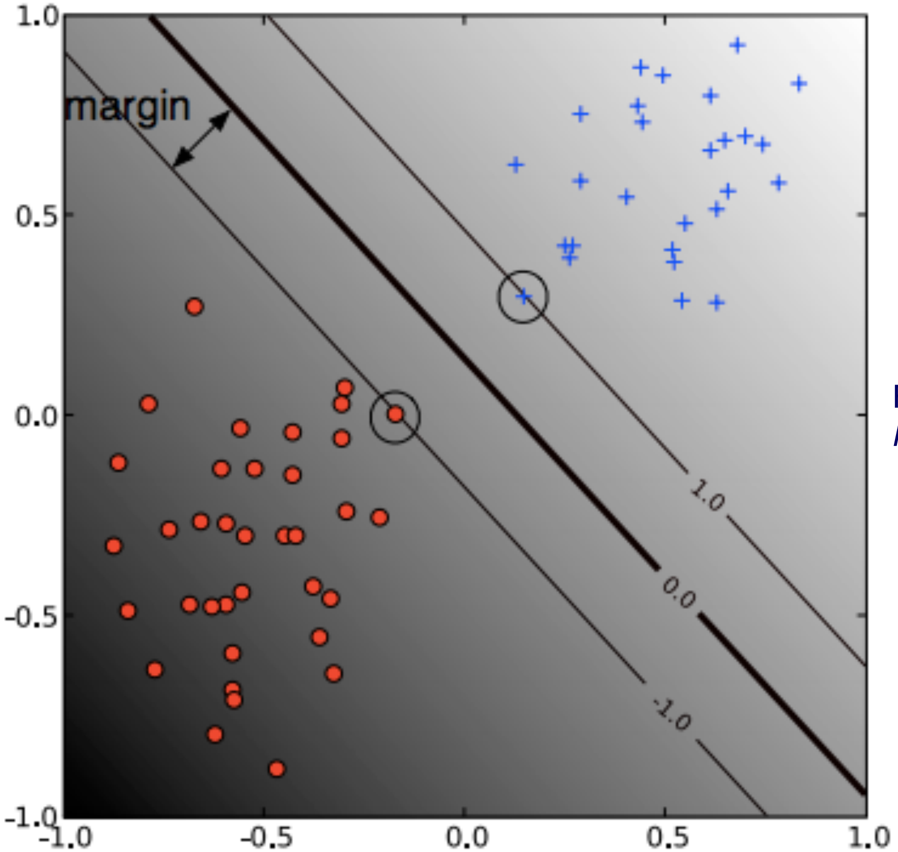


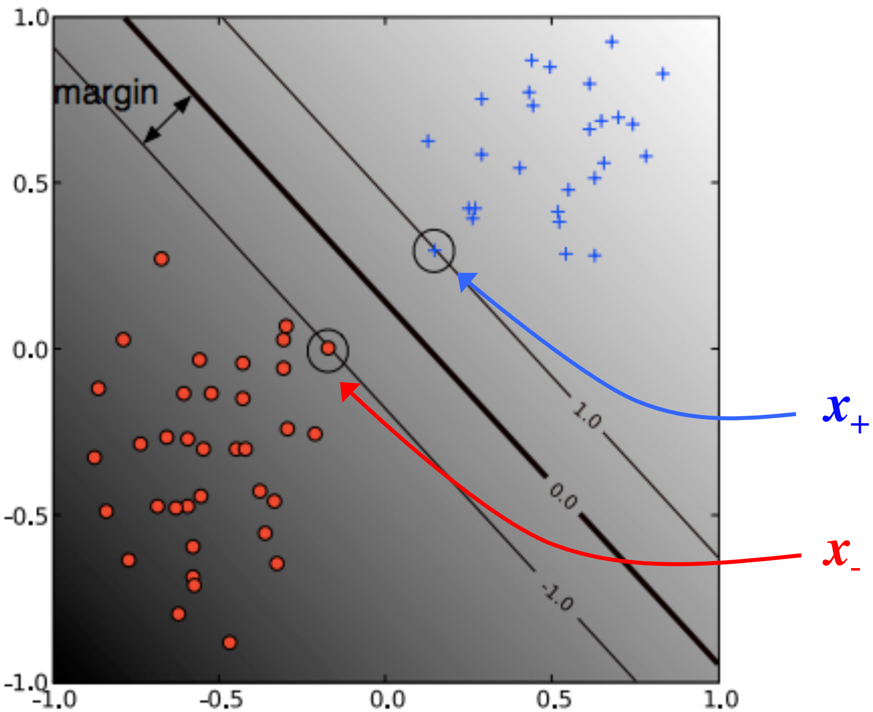
Figure from Ben-Hur & Weston, *Methods in Molecular Biology* 2010

Large margin classification

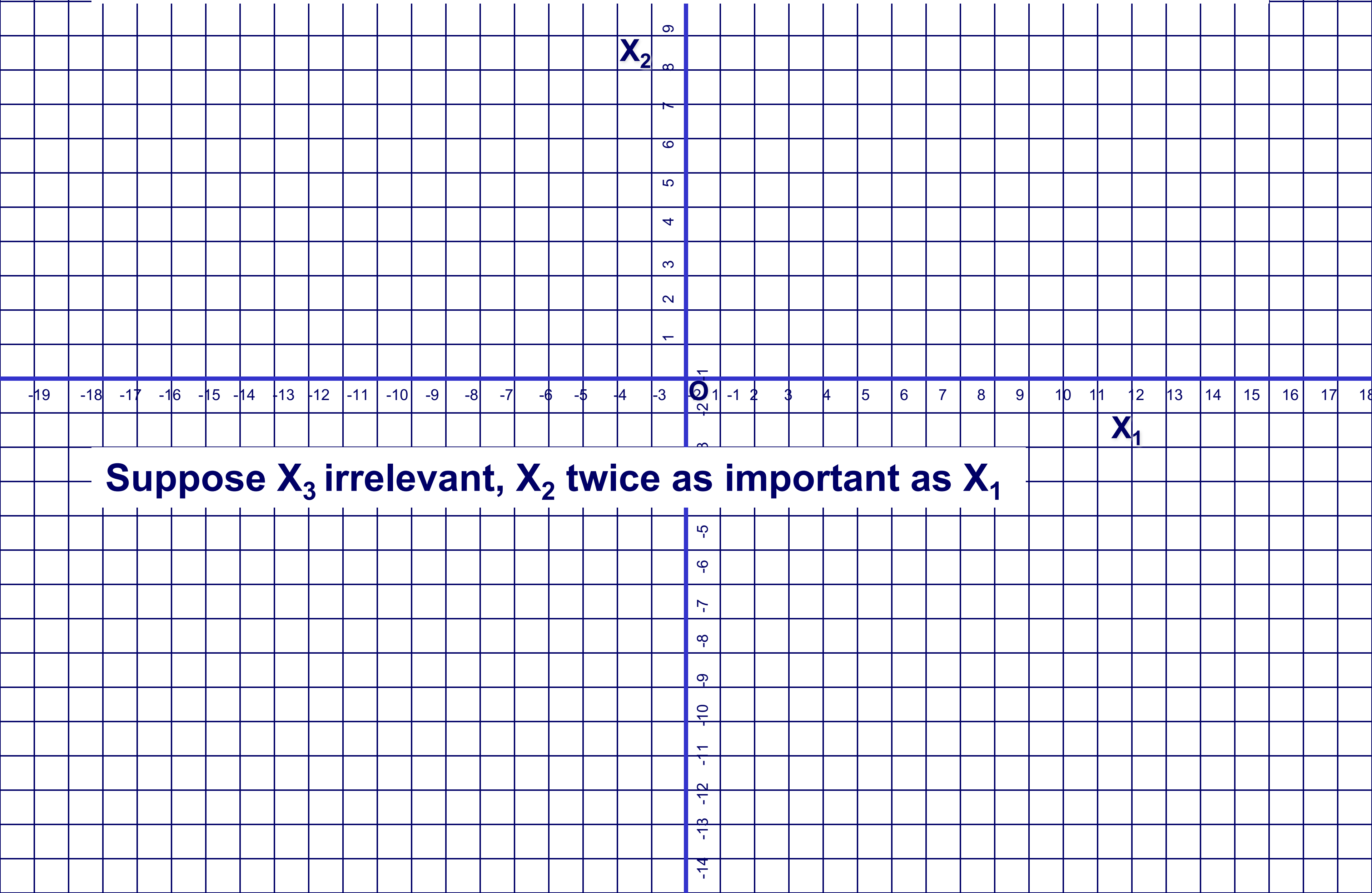
- suppose we learn a hyperplane h given a training set *D*
- let $\boldsymbol{x}_+$ denote the closest instance to the hyperplane among positive instances, and similarly for $\boldsymbol{x}_-$ and negative instances
- the margin is given by

$$\mathrm{margin}_D(h) = \frac{1}{2}\hat{\boldsymbol{w}}^{\mathsf{T}}\left(\boldsymbol{x}_+ - \boldsymbol{x}_-\right) = \frac{1}{\|\boldsymbol{w}\|_2}$$

length 1 vector in same direction as $\boldsymbol{w}$

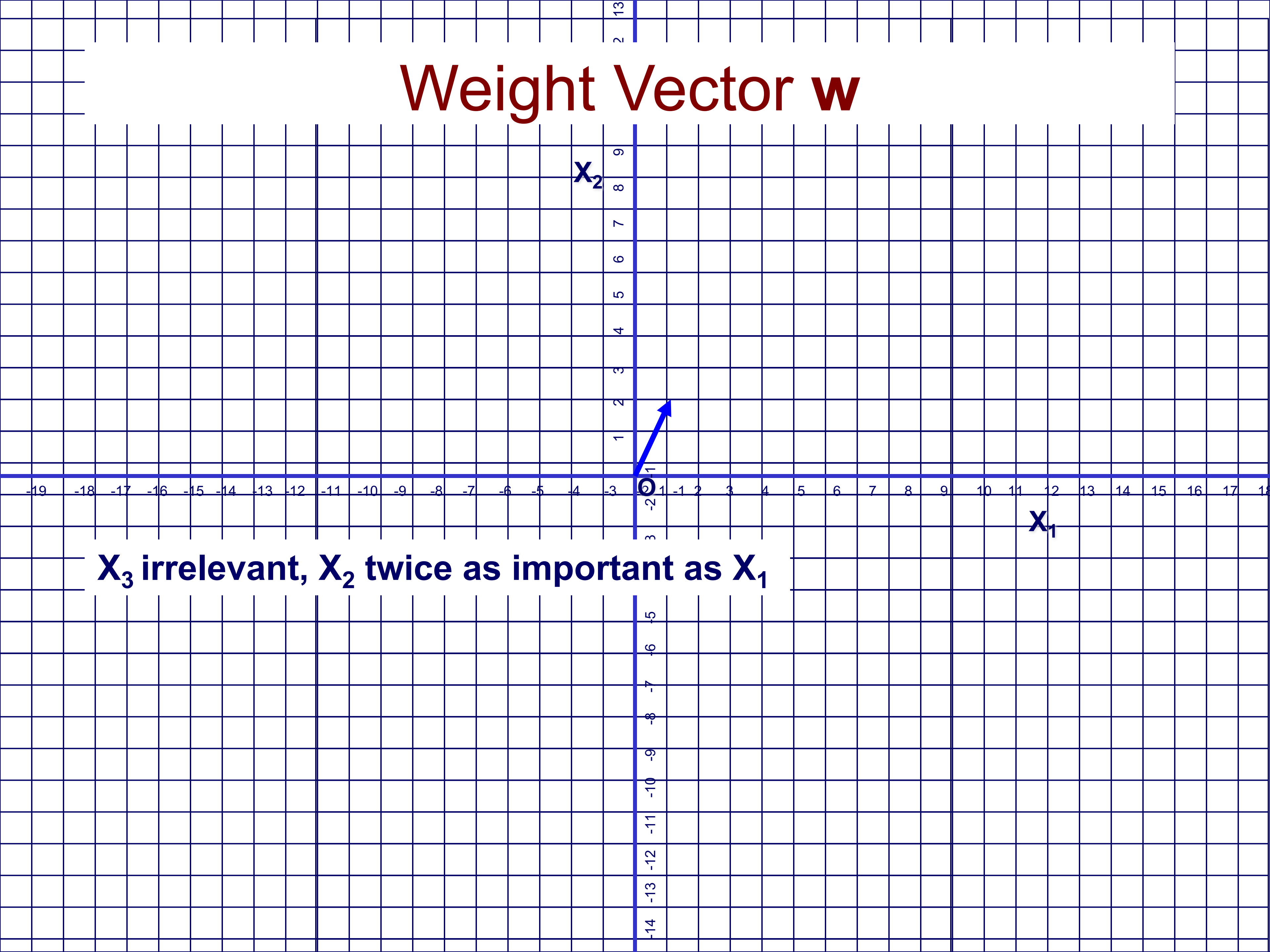


Example



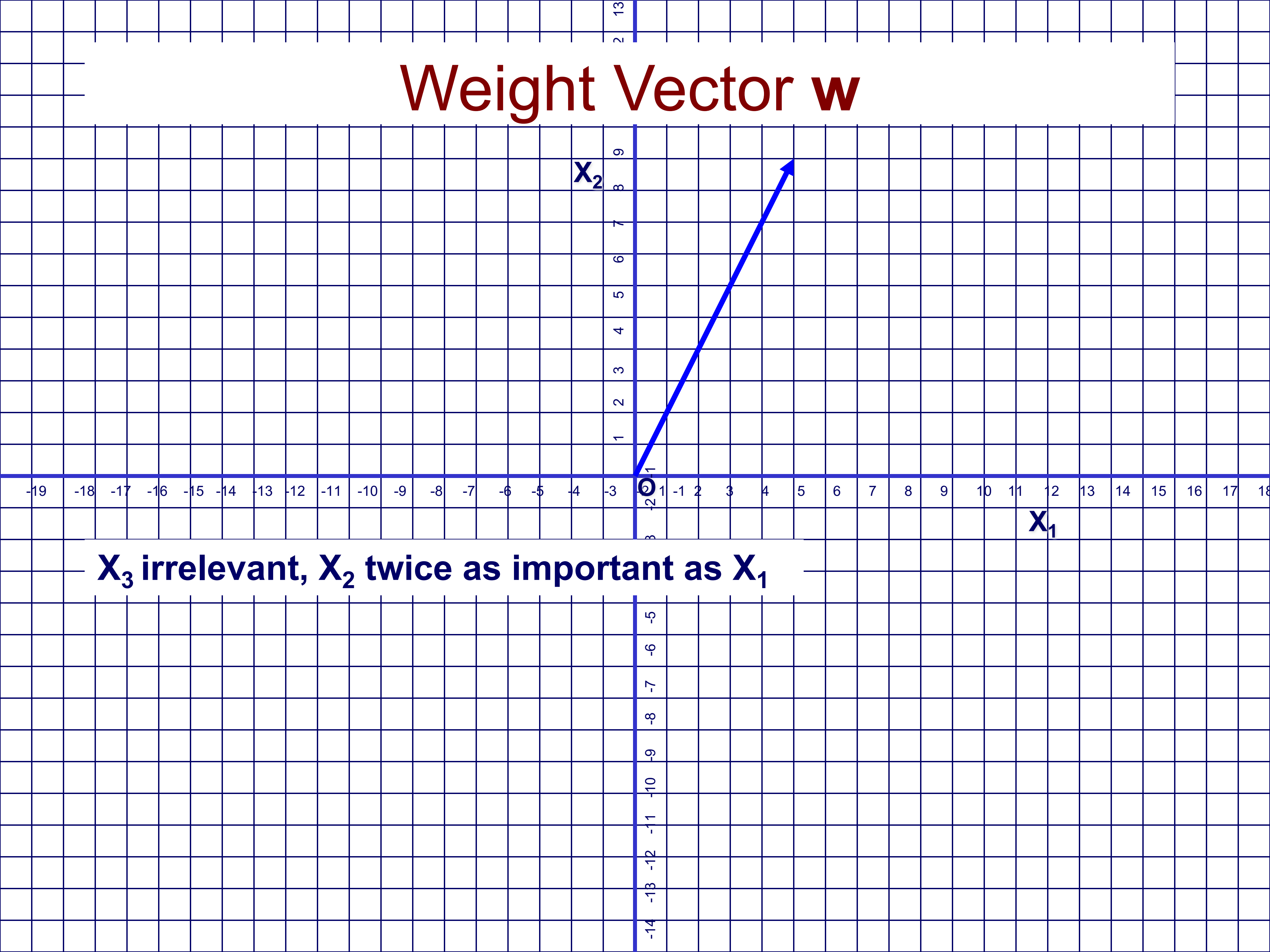
Suppose X_3 irrelevant, X_2 twice as important as X_1

Weight Vector **w**



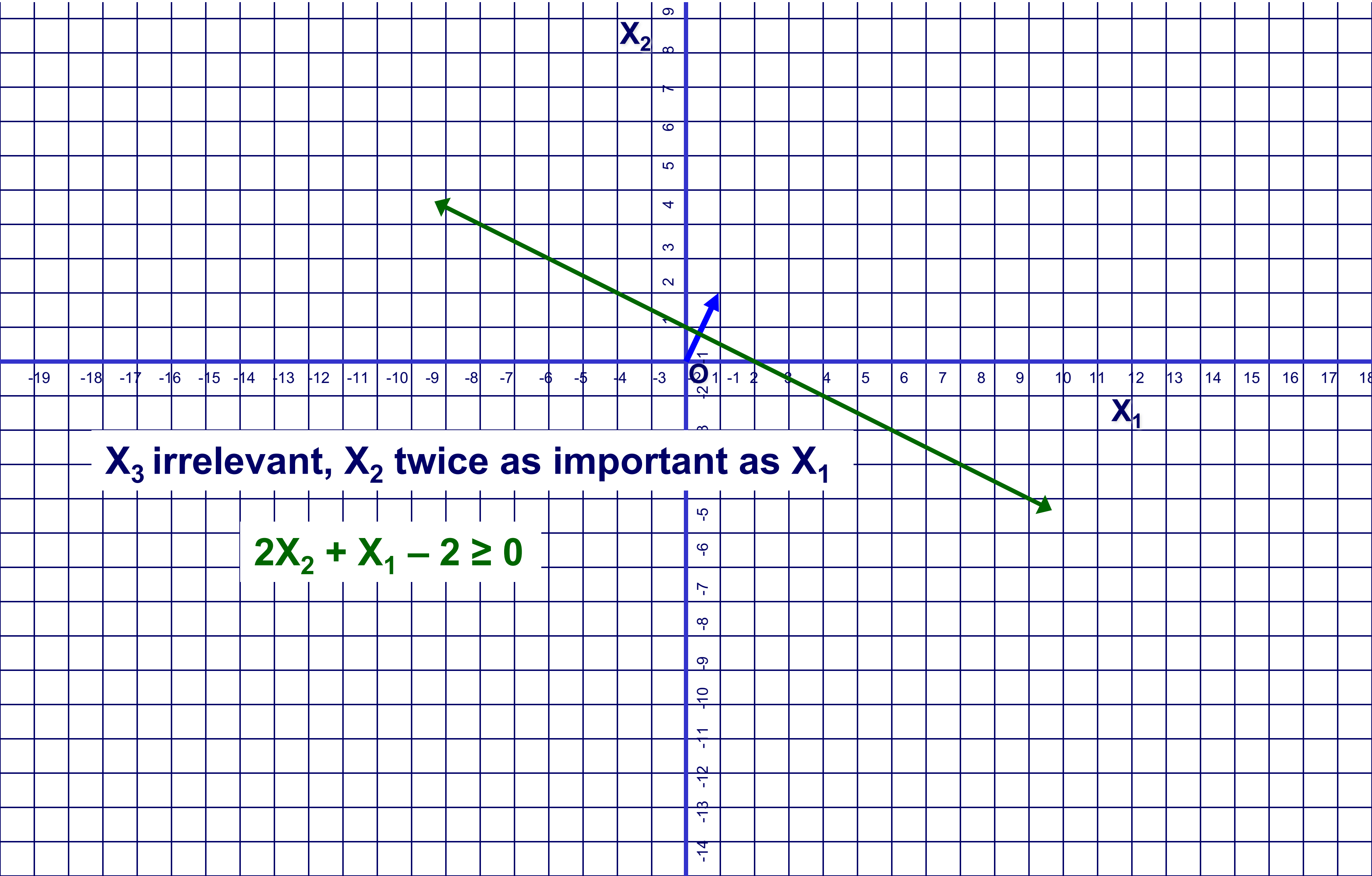
X_3 irrelevant, X_2 twice as important as X_1

Weight Vector **w**



X_3 irrelevant, X_2 twice as important as X_1

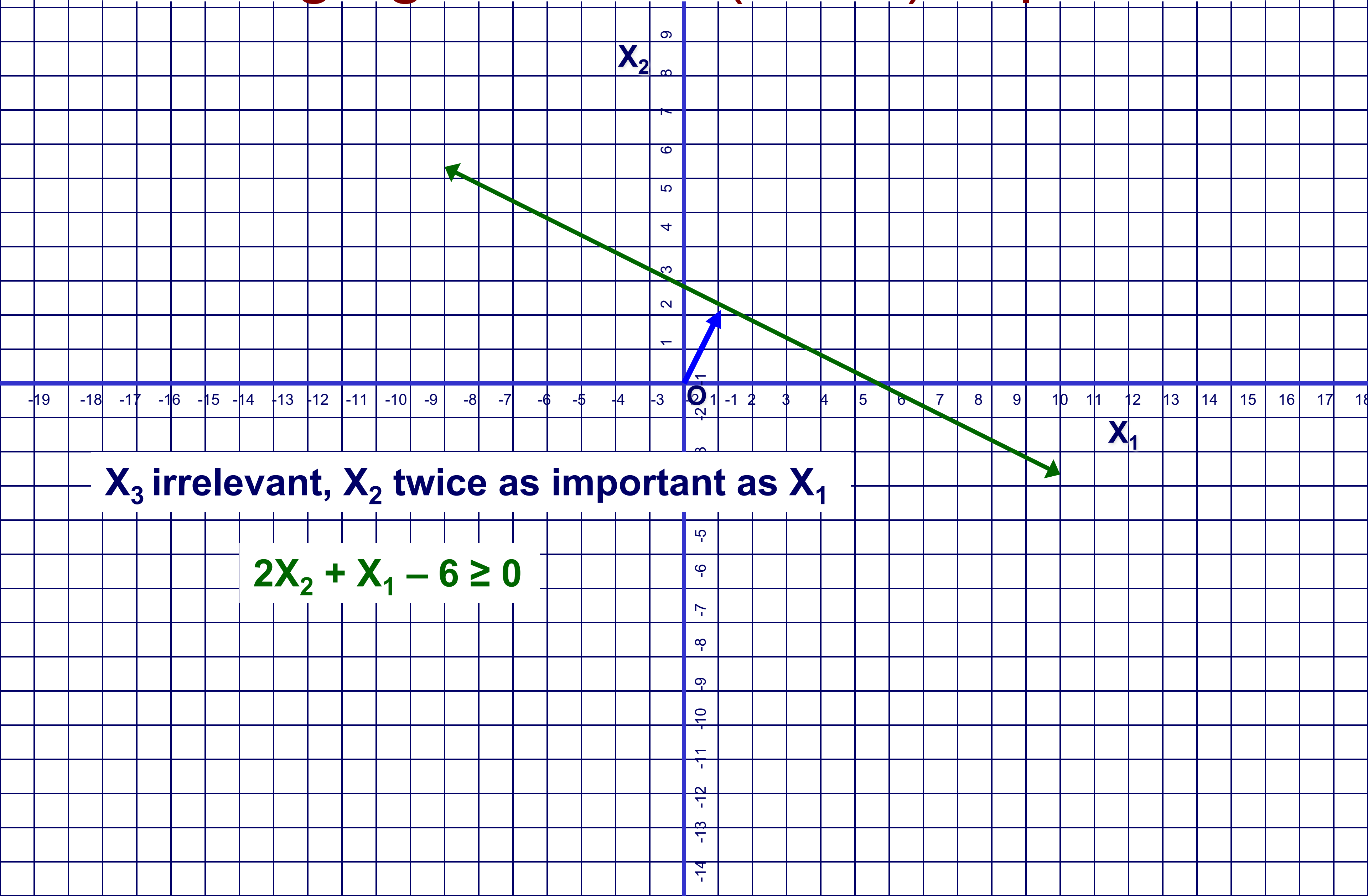
Separator is perpendicular to weight vector



X_3 irrelevant, X_2 twice as important as X_1

$$2X_2 + X_1 - 2 \geq 0$$

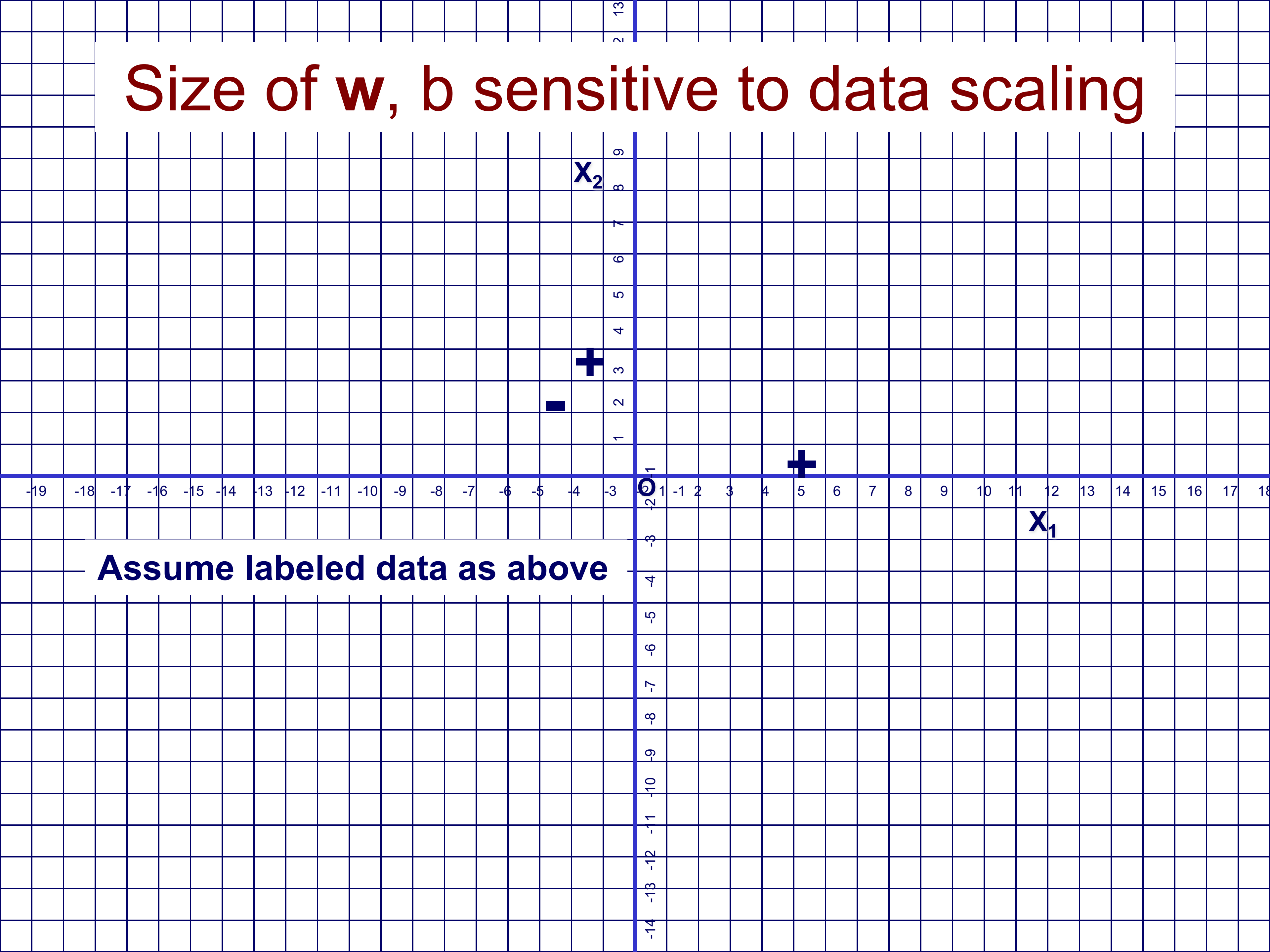
Changing b moves (shifts) separator



X_3 irrelevant, X_2 twice as important as X_1

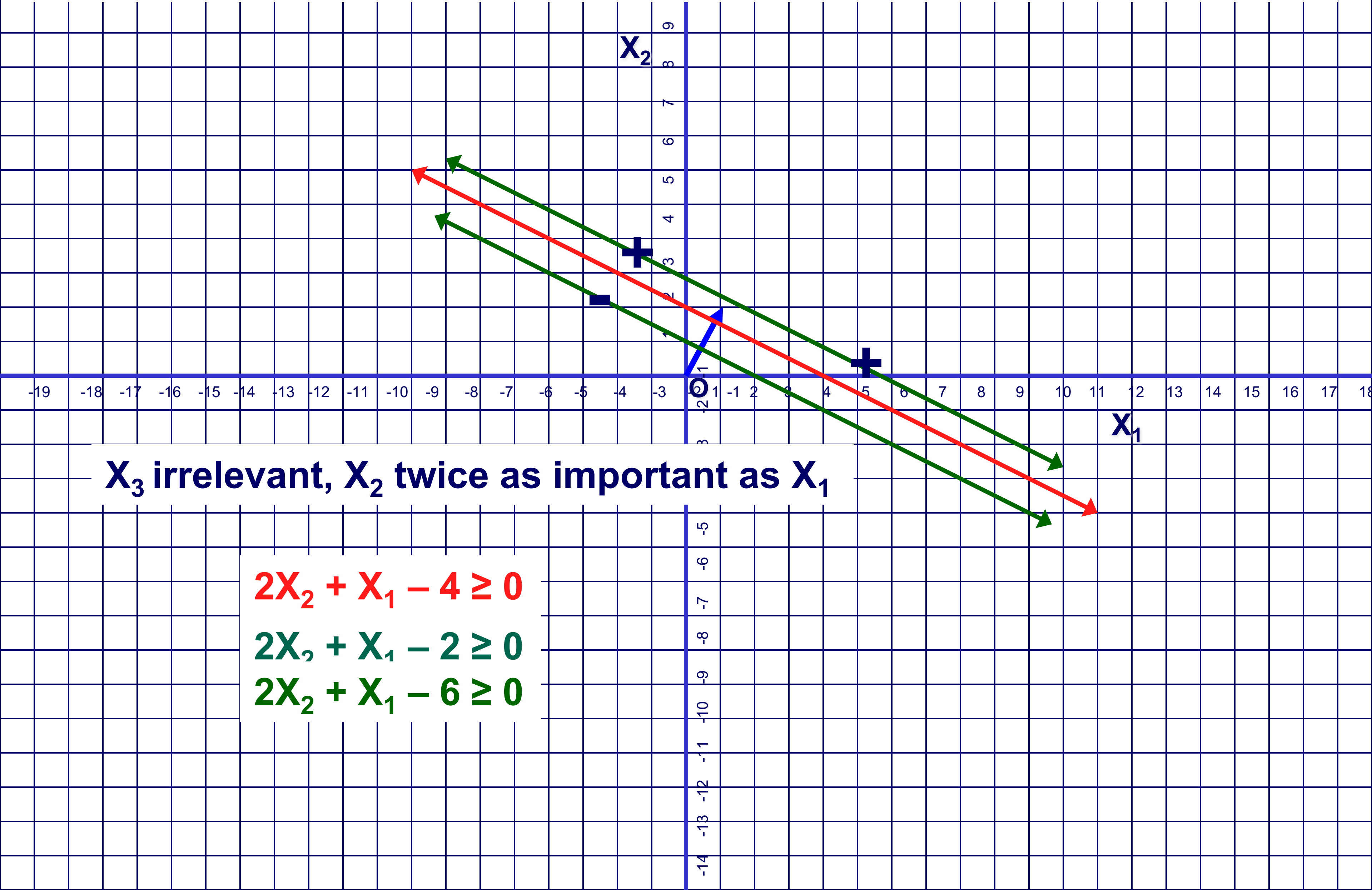
$$2X_2 + X_1 - 6 \geq 0$$

Size of **w**, b sensitive to data scaling



Assume labeled data as above

Margin is width between our earlier lines



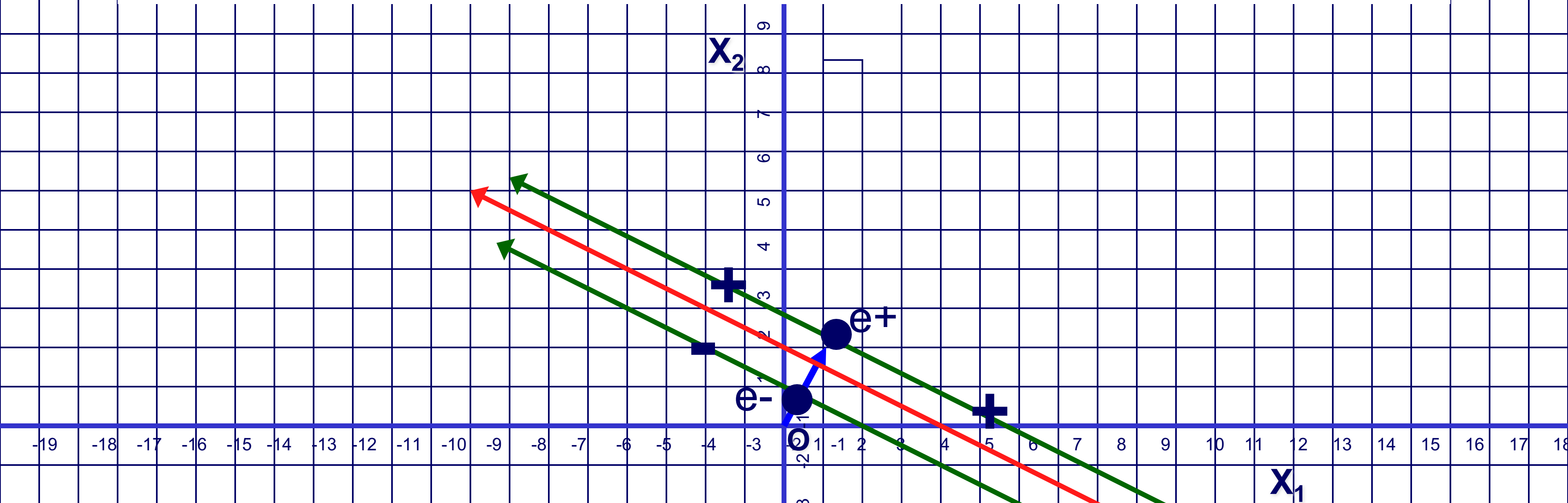
X_3 irrelevant, X_2 twice as important as X_1

$2X_2 + X_1 - 4 \geq 0$

$2X_2 + X_1 - 2 \geq 0$

$2X_2 + X_1 - 6 \geq 0$

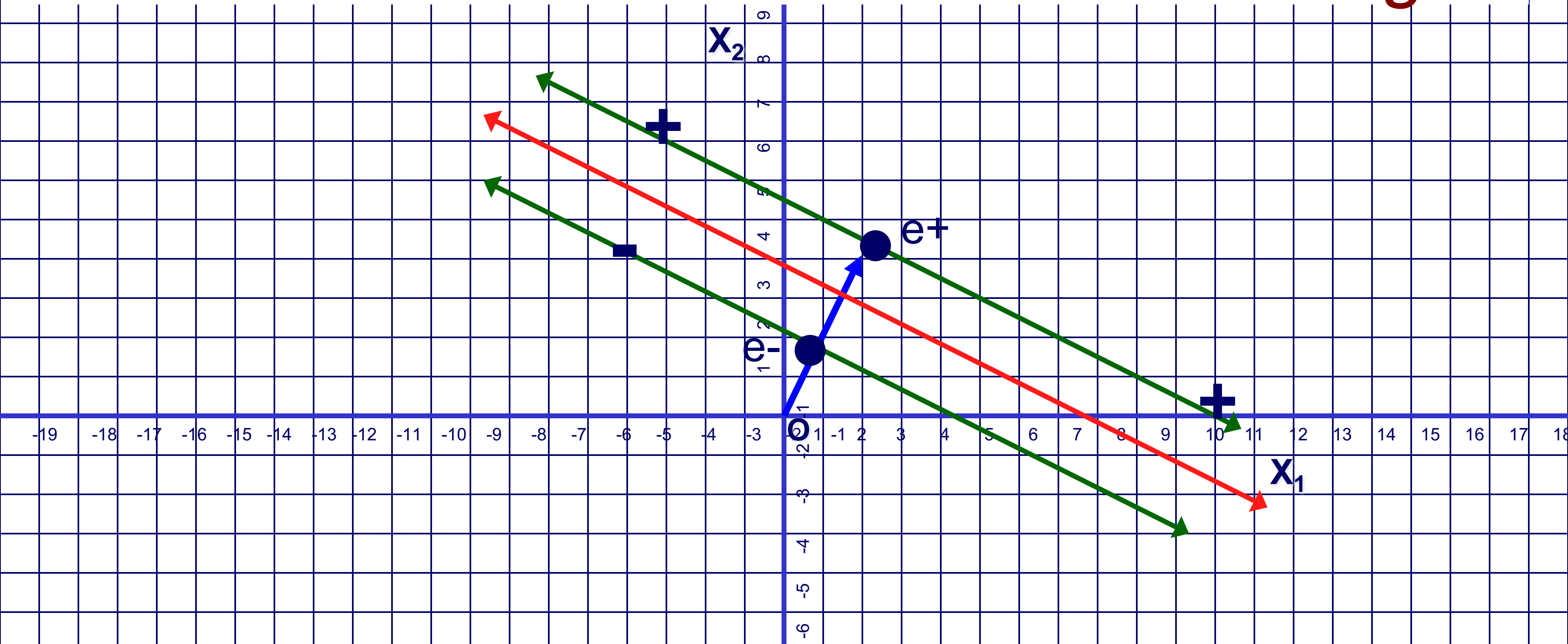
Specifically margin here is $||\mathbf{e}_+ - \mathbf{e}_-||_2$



Let **e+** and **e-**, respectively, be the points where **w** intersects the hyperplanes:

$$2X_2 + X_1 - 2 \geq 0$$
$$2X_2 + X_1 - 6 \geq 0$$

Margin doubled artificially: so neither ||w|| nor difference in b's alone defines *margin*



$2X_2 + X_1 - 8 \geq 0$

$2X_2 + X_1 - 4 = 0$

$2X_2 + X_1 - 12 = 0$

Keep **w** fixed, double b

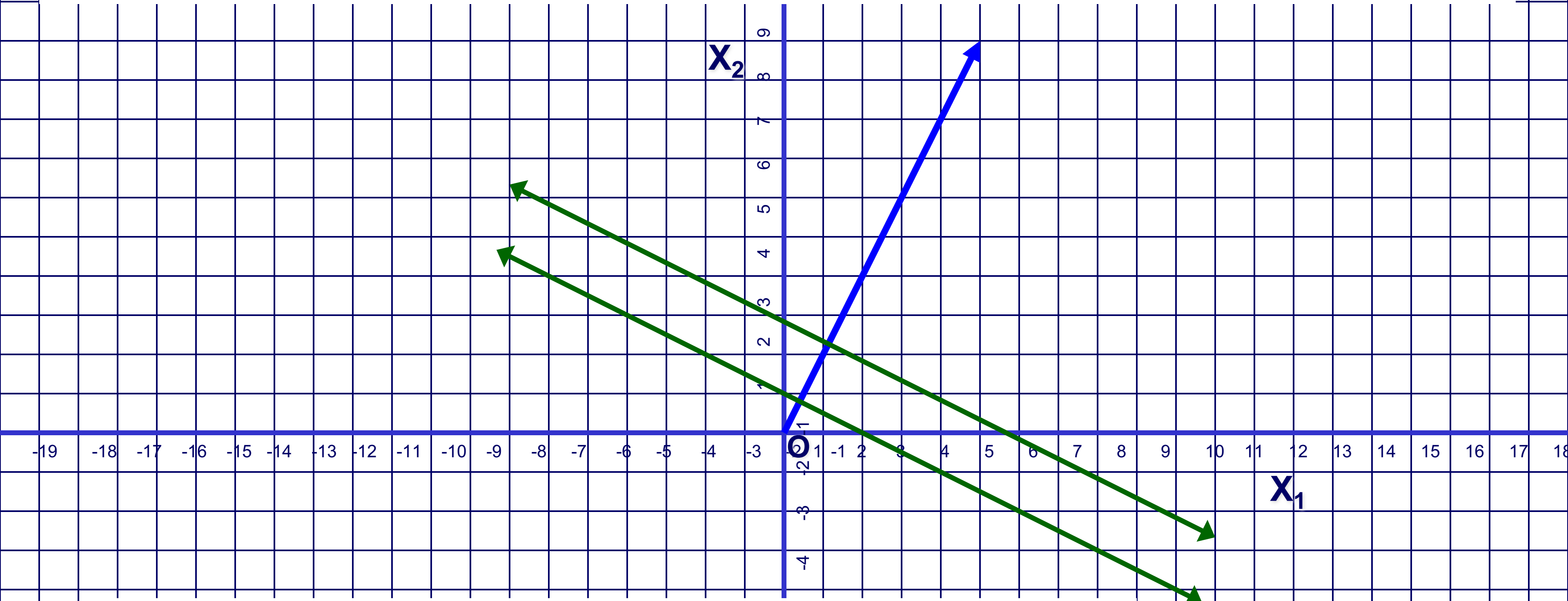
OR… $4X_2 + 2X_1 - 4 \geq 0$

$4X_2 + 2X_1 - 2 = 0$

$4X_2 + 2X_1 - 6 = 0$

Double **w**, keep b fixed

Should normalize margin with norm of **w**



But normalized margin remains unchanged:
$||\mathbf{e}_+ - \mathbf{e}_-||_2 \,/\, ||\mathbf{w}||_2$

Can keep $||\mathbf{w}||_2$ fixed and try to maximize $||\mathbf{e}_+ - \mathbf{e}_-||_2$

Or fix $||\mathbf{e}_+ - \mathbf{e}_-||_2$ (e.g., to 2) and minimize $||\mathbf{w}||_2$

The hard-margin SVM

- given a training set $D = \{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle \; , \ldots , \; \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \; \}$
- we can frame the goal of maximizing the margin as a constrained optimization problem

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \frac{1}{2} \| \boldsymbol{w} \|_2^2$$

adjust these parameters

to minimize this

$$\text{subject to constraints: } \quad y^{(i)} (\boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}^{(i)} + b) \geq 1$$

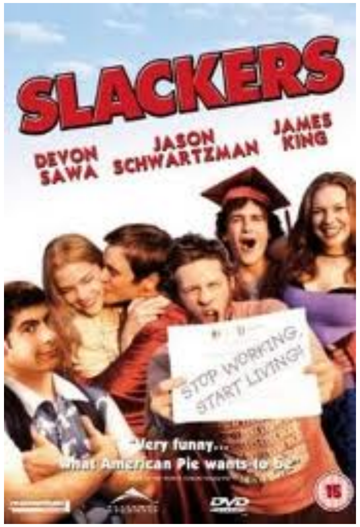
$$\text{for } i = 1, \ldots, m$$

correctly classify $\boldsymbol{x}^{(i)}$ with room to spare

- and use standard algorithms to find an optimal solution to this problem

The soft-margin SVM

[Cortes & Vapnik, *Machine Learning* 1995]



- if the training instances are not linearly separable, the previous formulation will fail
- we can adjust our approach by using *slack variables* (denoted by ξ) to tolerate errors

$$\underset{\boldsymbol{w}, b, \xi^{(1)} \ldots \xi^{(m)}}{\text{minimize}} \quad \frac{1}{2} \| \boldsymbol{w} \|_2^2 + C \sum_{i=1}^{m} \xi^{(i)}$$

$$\text{subject to constraints:} \quad y^{(i)} (\boldsymbol{w}^{\top} \boldsymbol{x}^{(i)} + b) \geq 1 - \xi^{(i)}$$

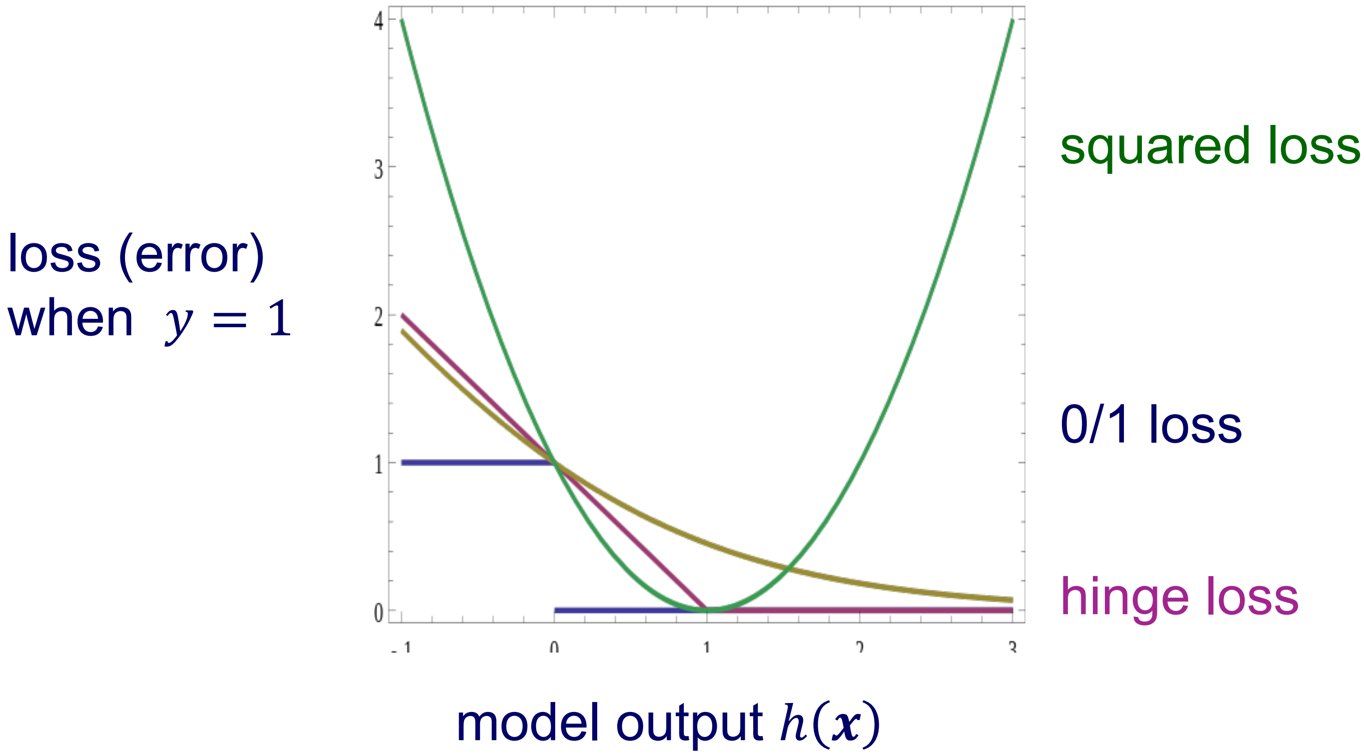
$$\xi^{(i)} \geq 0$$

for $i = 1, \ldots, m$

- C determines the relative importance of maximizing margin vs. minimizing slack

Hinge loss

- when we covered neural nets, we talked about minimizing squared loss and cross-entropy loss
- SVMs minimize *hinge loss*



The effect of C in a soft-margin SVM

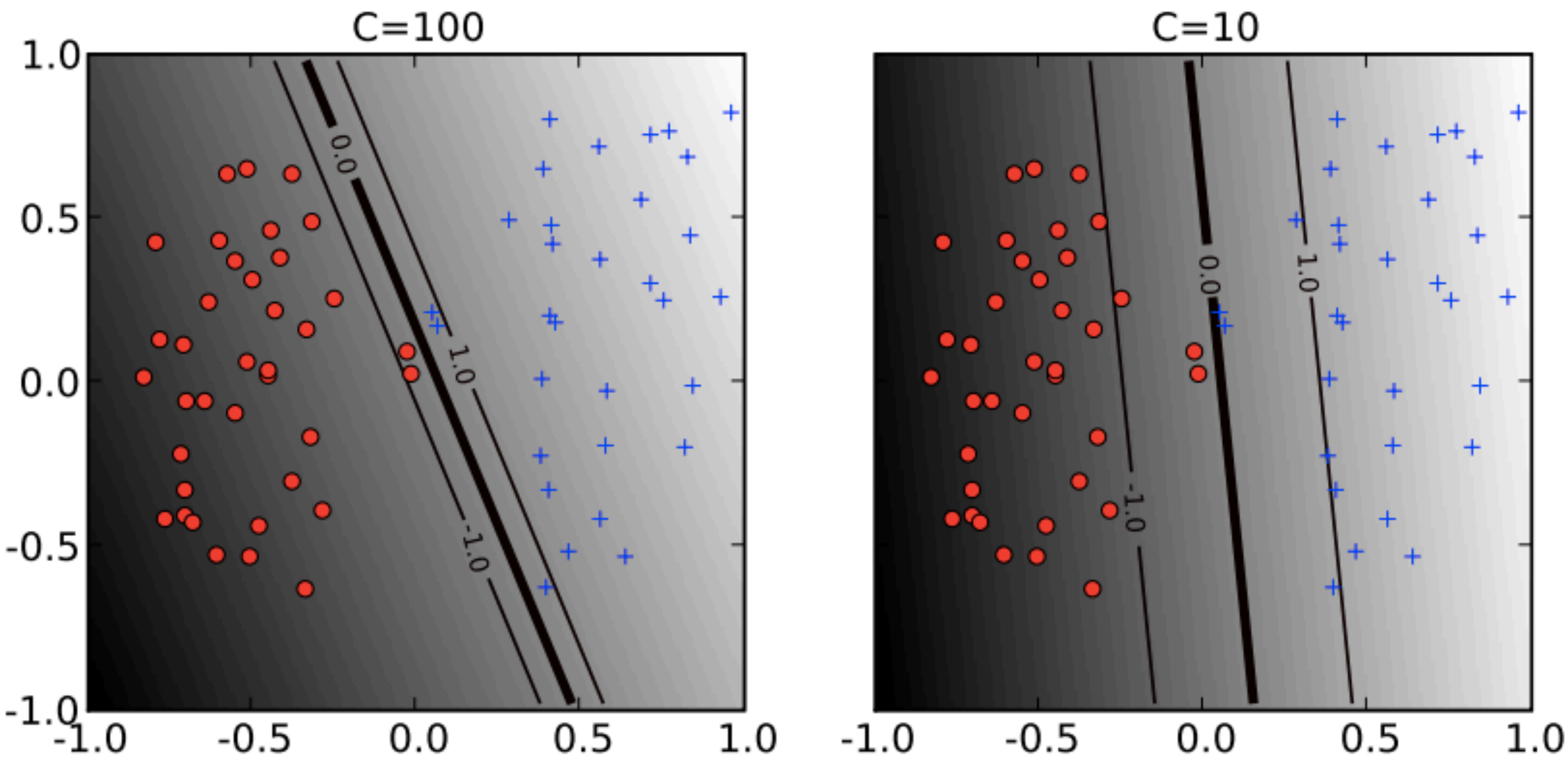
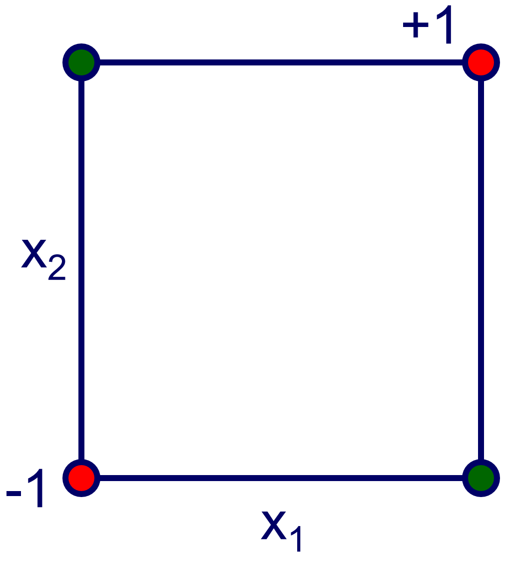
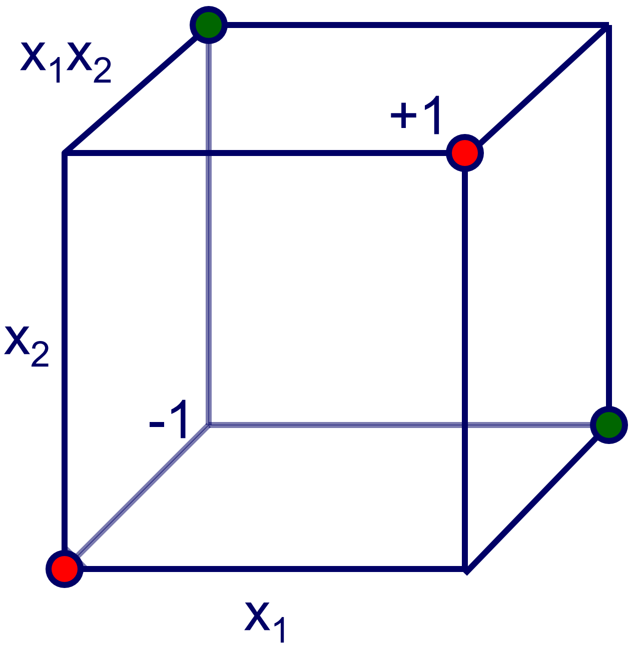


Figure from Ben-Hur & Weston,
Methods in Molecular Biology 2010

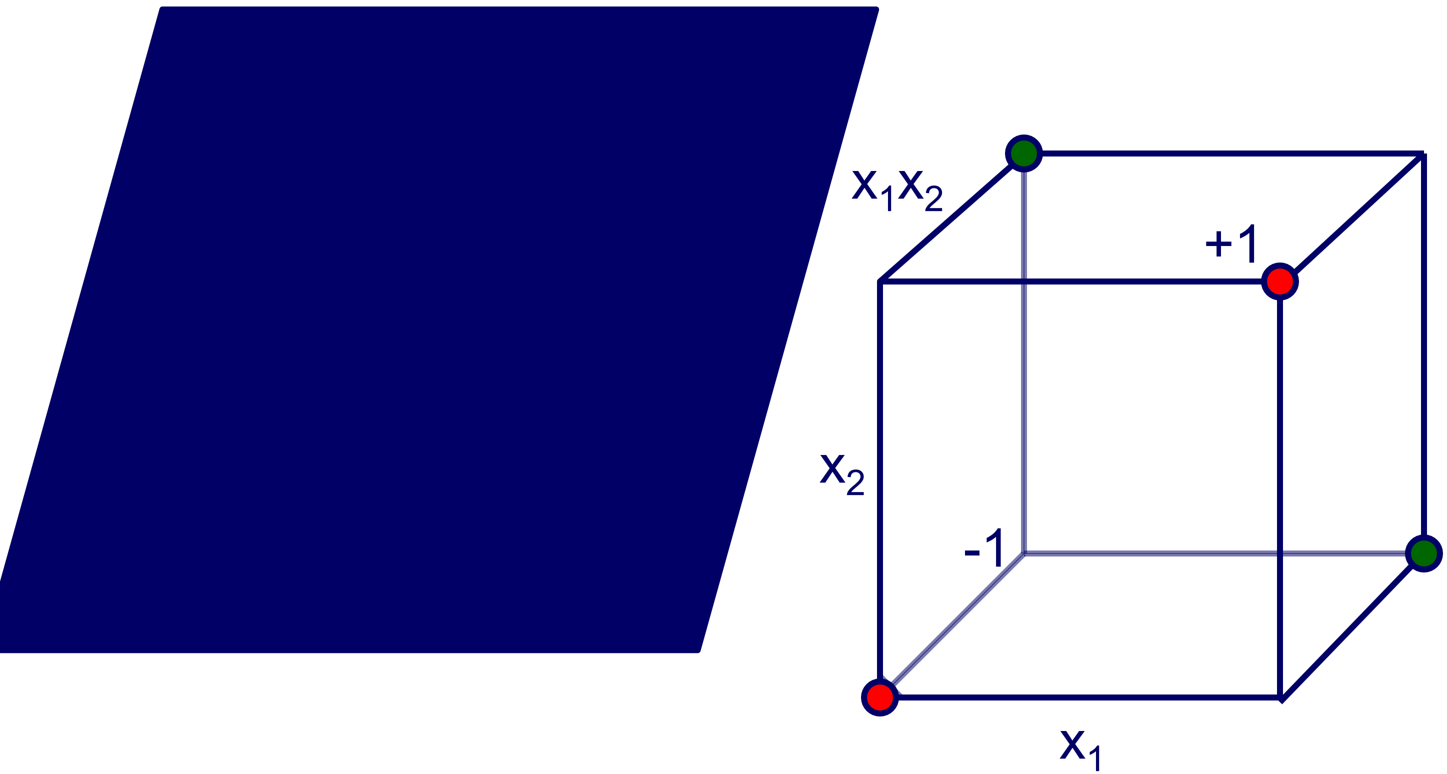
Nonlinear classifiers



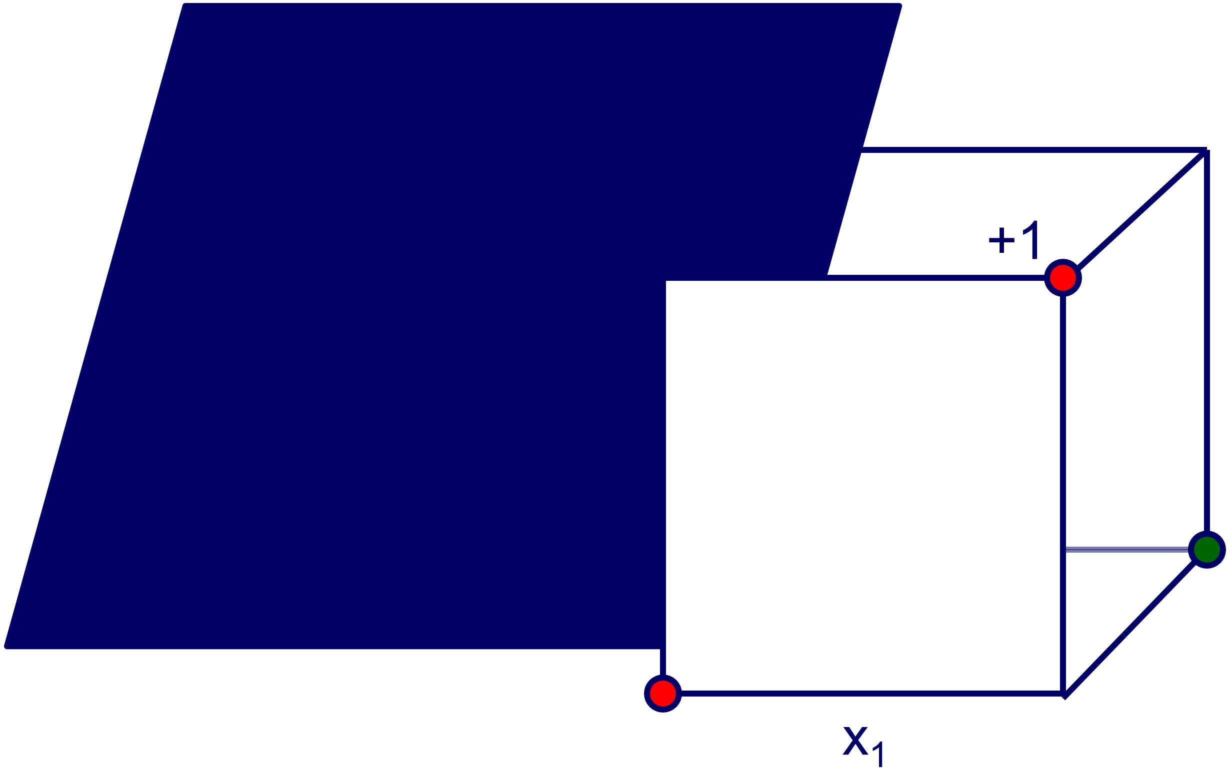
Nonlinear classifiers



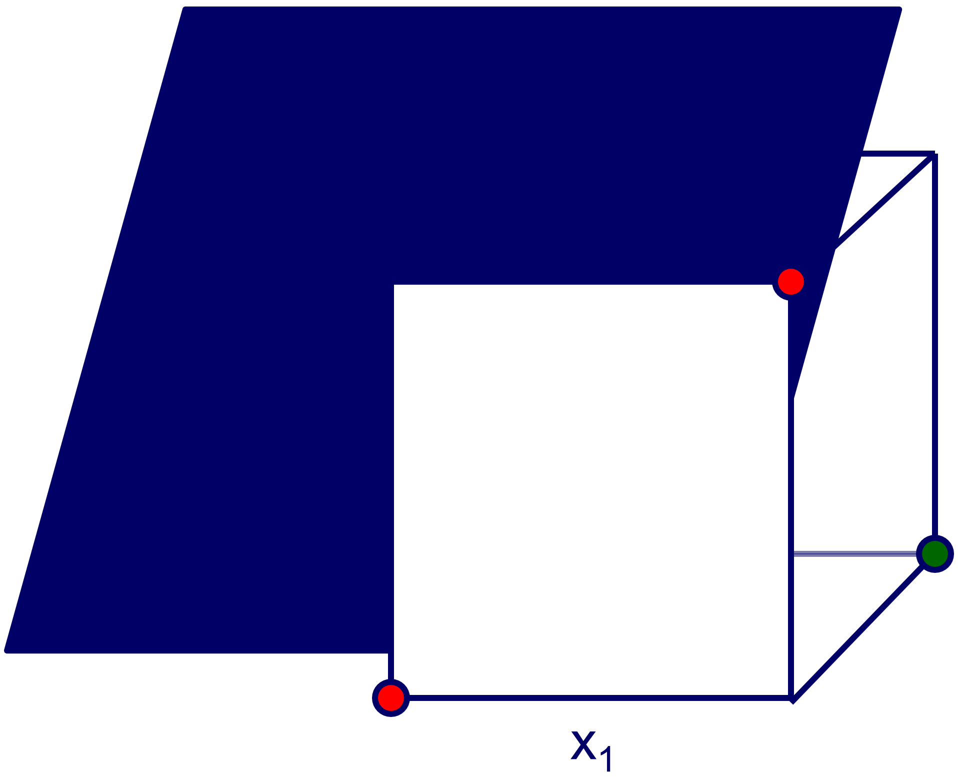
Nonlinear classifiers



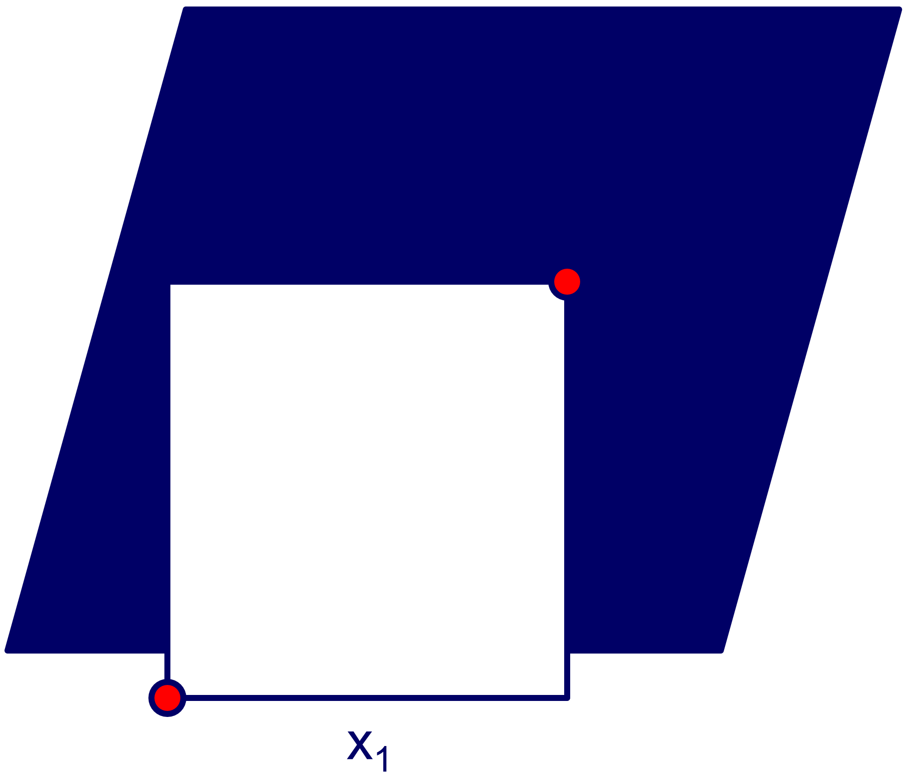
Nonlinear classifiers



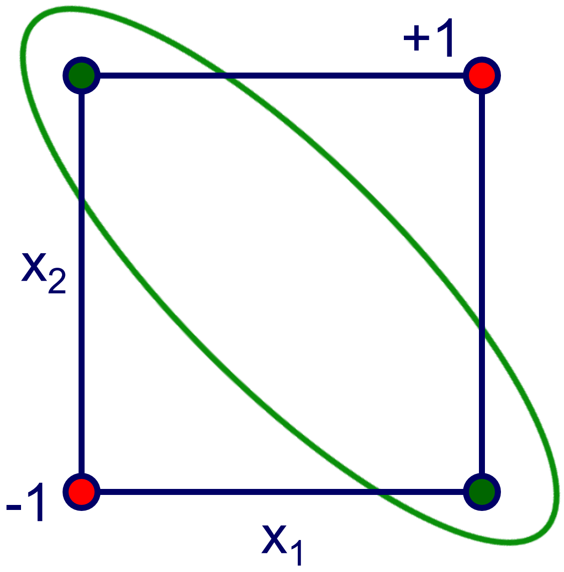
Nonlinear classifiers



Nonlinear classifiers



Nonlinear classifiers



Nonlinear classifiers

- What if a linear separator is not an appropriate decision boundary for a given task?
- For any (consistent) data set, there exists a mapping ϕ to a higher-dimensional space such that the data is linearly separable

$$\phi(\boldsymbol{x}) = \left(\phi_1(\boldsymbol{x}),\ \phi_2(\boldsymbol{x}),\ \ldots,\ \phi_k(\boldsymbol{x})\right)$$

- Example: mapping to quadratic space

$$\boldsymbol{x} = \left\langle x_1,\ x_2 \right\rangle$$ suppose $\boldsymbol{x}$ is represented by 2 features

$$\phi(\boldsymbol{x}) = \left(x_1^2,\ \sqrt{2}x_{\mathbf{1}}x_2,\ x_2^2,\ \sqrt{2}x_{\mathbf{1}},\ \sqrt{2}x_2,\ 1\right)$$

- now try to find a linear separator in this space

Nonlinear classifiers

- for the linear case, our discriminant function was given by

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

- for the nonlinear case, it can be expressed as

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \phi(\boldsymbol{x}) + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\boldsymbol{w}$ is a higher dimensional vector

SVMs with polynomial kernels

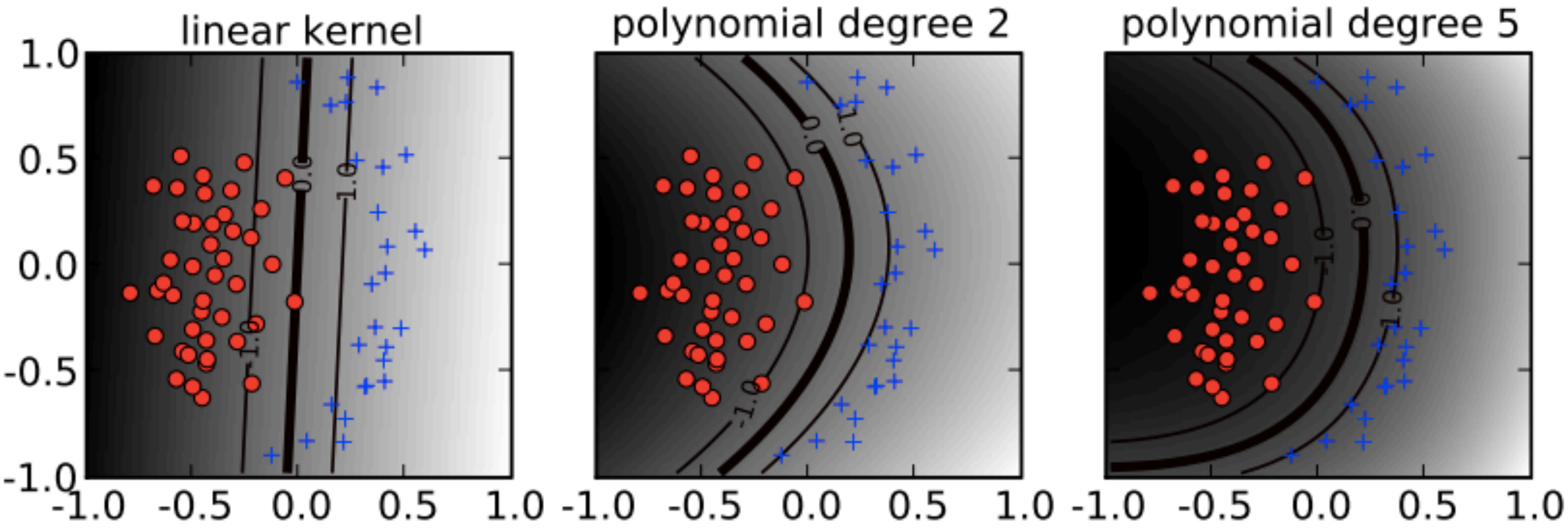


Figure from Ben-Hur & Weston,
Methods in Molecular Biology 2010

The kernel trick

- explicitly computing this nonlinear mapping does not scale well
- a dot product between two higher-dimensional mappings can sometimes be implemented by a *kernel function*
- example: quadratic kernel

$$
\begin{aligned}
k(\boldsymbol{x},\boldsymbol{z}) &= \left(\boldsymbol{x}\cdot\boldsymbol{z}+1\right)^2 \\
&= \left(x_1 z_1 + x_2 z_2 + 1\right)^2 \\
&= x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\
&= \left(x_1^2,\ \sqrt{2}x_1 x_2,\ x_2^2,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ 1\right)\bullet \\
&\quad\ \left(z_1^2,\ \sqrt{2}z_1 z_2,\ z_2^2,\ \sqrt{2}z_1,\ \sqrt{2}z_2,\ 1\right) \\
&= \phi(\boldsymbol{x})\cdot\phi(\boldsymbol{z})
\end{aligned}
$$

The kernel trick

- thus we can use a kernel to compute the dot product without explicitly mapping the instances to a higher-dimensional space

$$k(\boldsymbol{x}, z) = \left(\boldsymbol{x} \cdot z + 1\right)^2 = \phi(\boldsymbol{x}) \cdot \phi(z)$$

But why is the kernel trick helpful?

Using the kernel trick

- given a training set $D = \{ \langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle, \ldots, \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle \}$
- suppose the weight vector can be represented as a linear combination of the training instances

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^{(i)}$$

- then we can represent a linear SVM as

$$\sum_{i=1}^{m} \alpha_i \boldsymbol{x}^{(i)} \bullet \boldsymbol{x} + b$$

- and a nonlinear SVM as

$$\sum_{i=1}^{m} \alpha_i \phi\left(\boldsymbol{x}^{(i)}\right) \bullet \phi\left(\boldsymbol{x}\right) + b$$

$$= \sum_{i=1}^{m} \alpha_i k\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}\right) + b$$

Can we represent a weight vector as a linear combination of training instances?

- consider perceptron learning, where each weight can be represented as

$$w_j = \sum_{i=1}^{m} \alpha_i x_j^{(i)}$$

- proof: each weight update has the form $w_j(t) = w_j(t-1) + \eta \delta_t^{(i)} y^{(i)} x_j^{(i)}$

$$\delta_t^{(i)} = \begin{cases} 1 \text{ if } \boldsymbol{x}^{(i)} \text{ misclassified in epoch } t \\ 0 \text{ otherwise} \end{cases}$$

$$w_j = \sum_t \sum_{i=1}^{m} \eta \delta_t^{(i)} y^{(i)} x_j^{(i)}$$

$$w_j = \sum_{i=1}^{m} \left(\sum_t \eta \delta_t^{(i)} y^{(i)} \right) x_j^{(i)}$$

$$w_j = \sum_{i=1}^{m} \alpha_i x_j^{(i)}$$

The *primal* and *dual* formulations of the hard-margin SVM

primal

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

subject to constraints: $y^{(i)}(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b) \geq 1$

for $i = 1, \ldots, m$

dual

$$\underset{\alpha_1,\ldots,\alpha_m}{\text{maximize}} \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{j=1}^{m} \sum_{k=1}^{m} \alpha_j \alpha_k y^{(j)} y^{(k)} \left(\boldsymbol{x}^{(j)} \bullet \boldsymbol{x}^{(k)} \right)$$

subject to constraints: $\alpha_i \geq 0 \quad$ for $i = 1, \ldots, m$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

How did we get this?
Review: Lagrange Multipliers

Given: a constrained optimization problem such as Lasso regression:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2 \text{ s.t. } |\beta| \leq c$$

Solve: unconstrained problem with *Langrange multiplier* λ:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2 + \lambda|\beta|$$

How did we get this?
Review: Lagrange Multipliers

Given: a constrained optimization problem such as Lasso regression:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2 \text{ s.t. } |\beta| \leq c$$

Solve: unconstrained problem with *Langrange multiplier* λ:

$$\min_{\beta} \|\mathbf{X}\beta - \mathbf{y}\|_2^2 + \lambda|\beta|$$

Using $\alpha_1, ..., \alpha_m$ for Langrange multipliers, Lagrangian of SVM primal is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_i \alpha_i y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)$$

Set to 0 the partial derivatives wrt **w** and b

Using $\alpha_1, ..., \alpha_m$ for Langrange multipliers, Lagrangian of SVM primal is:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_i \alpha_i y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)$$

Setting derivative with respect to $\mathbf{w}$ to 0 yields:

$$\mathbf{w} = \sum_i \alpha_i y^{(i)}\mathbf{x}^{(i)}$$

Setting derivative with respect to b to 0 yields:

$$\sum_i \alpha_i y^{(i)} = 0$$

Substituting these back in yields the dual.

The *dual* formulation with a kernel (hard margin version)

primal

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

subject to constraints: $y^{(i)}(\boldsymbol{w}^{\mathsf{T}}\phi\left(\boldsymbol{x}^{(i)}\right)+b)\geq 1$

for $i=1,\ldots,m$

dual

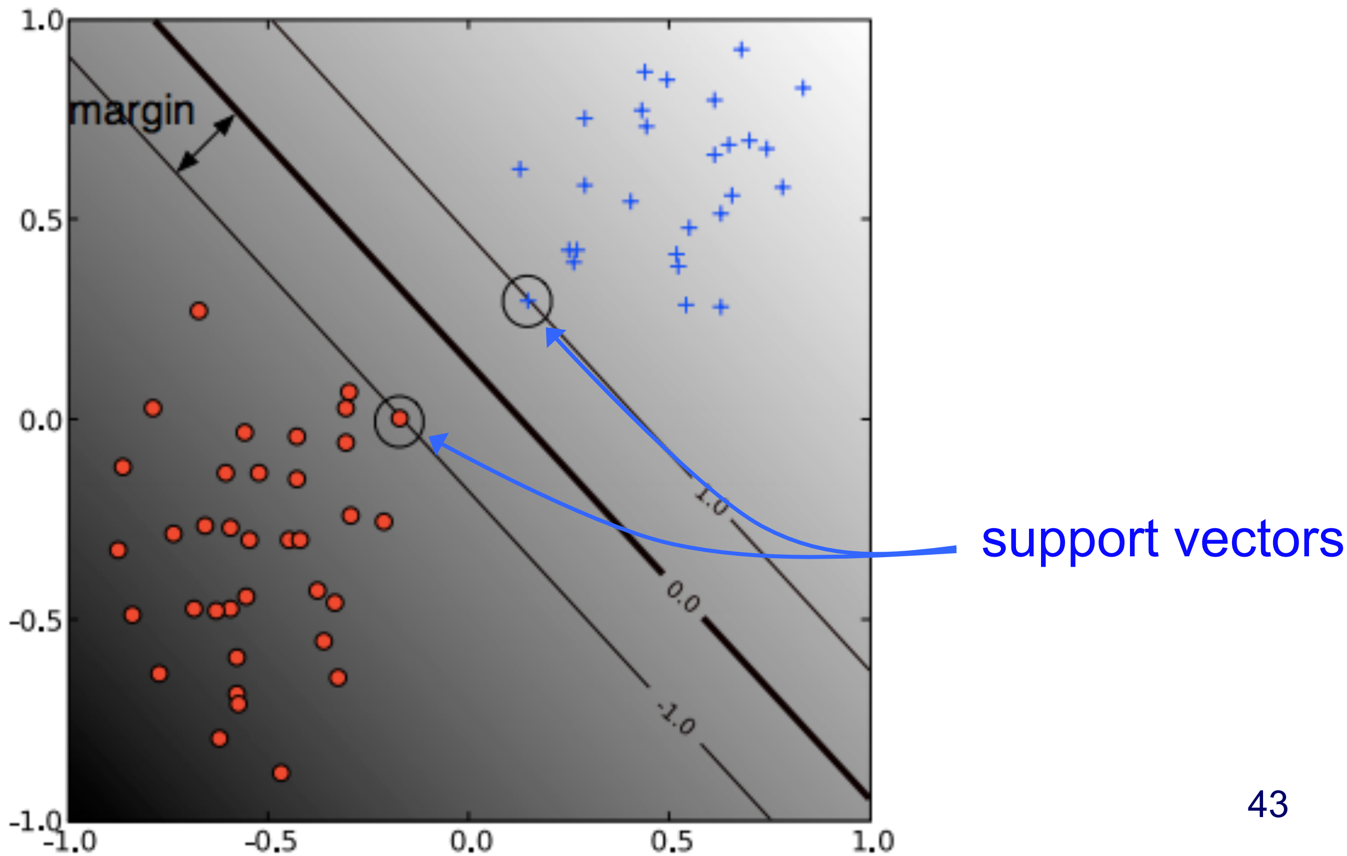
$$\underset{\alpha_1,\ldots,\alpha_m}{\text{maximize}} \quad \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{j=1}^{m}\sum_{k=1}^{m}\alpha_j\alpha_k y^{(j)}y^{(k)}k\left(\boldsymbol{x}^{(j)},\boldsymbol{x}^{(k)}\right)$$

subject to constraints: $\alpha_i \geq 0$ for $i=1,\ldots,m$

$$\sum_{i=1}^{m}\alpha_i y^{(i)} = 0$$

Support vectors

- the final solution is a sparse linear combination of the training instances
- those instances having $\alpha_i > 0$ are called *support vectors* – they lie on the margin boundary
- the solution wouldn't change if all the instances with $\alpha_i = 0$ were deleted



The kernel matrix

- the kernel matrix (a.k.a. Gram matrix) represents pairwise similarities for instances in the training set

$$\begin{bmatrix} k(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(1)}) & k(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(2)}) & \cdots & k(\boldsymbol{x}^{(1)},\boldsymbol{x}^{(m)}) \\ k(\boldsymbol{x}^{(2)},\boldsymbol{x}^{(1)}) & \ddots & & \\ \vdots & & & \\ k(\boldsymbol{x}^{(m)},\boldsymbol{x}^{(1)}) & & & k(\boldsymbol{x}^{(m)},\boldsymbol{x}^{(m)}) \end{bmatrix}$$

- it represents the information about the training set that is provided as input to the optimization process

Some common kernels

- polynomial of degree d

$$k(\boldsymbol{x},\boldsymbol{z}) = \left(\boldsymbol{x}\cdot\boldsymbol{z}\right)^d$$

- polynomial of degree up to d

$$k(\boldsymbol{x},\boldsymbol{z}) = \left(\boldsymbol{x}\cdot\boldsymbol{z}+1\right)^d$$

- radial basis function (RBF) (a.k.a. Gaussian)

$$k(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-\gamma\,\|\boldsymbol{x}-\boldsymbol{z}\|^2\right)$$

The RBF kernel

- the feature mapping ϕ for the RBF kernel is infinite dimensional!
- recall that $k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \bullet \phi(\boldsymbol{z})$

$$k(\boldsymbol{x},\boldsymbol{z}) = \exp\left(-\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2\right) \quad \text{for } \gamma = \frac{1}{2}$$

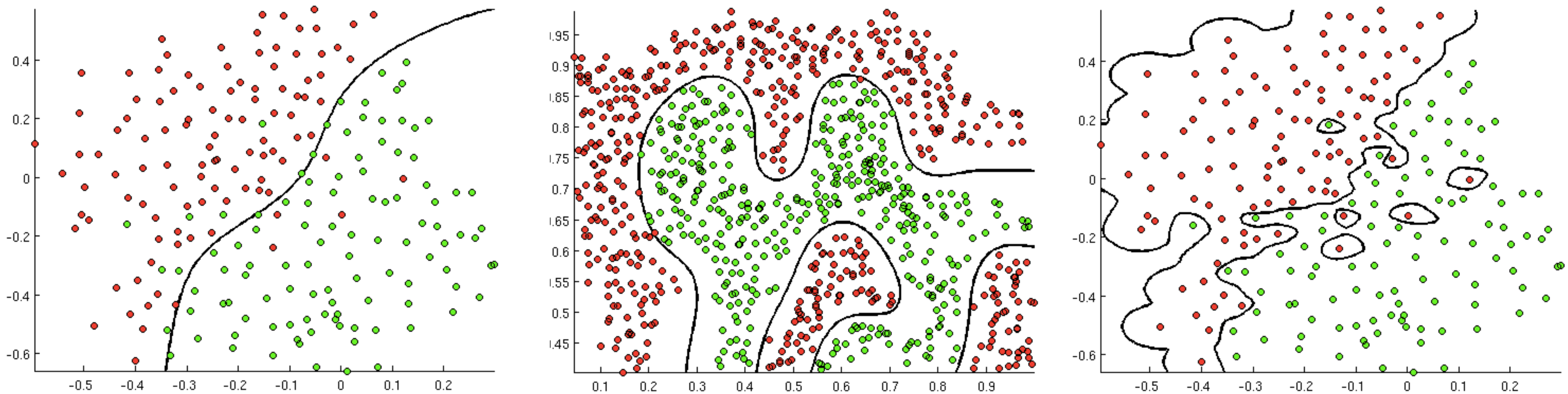
$$= \exp\left(-\frac{1}{2}\|\boldsymbol{x}\|^2\right)\exp\left(-\frac{1}{2}\|\boldsymbol{z}\|^2\right)\exp(\boldsymbol{x}\cdot\boldsymbol{z})$$

$$= \exp\left(-\frac{1}{2}\|\boldsymbol{x}\|^2\right)\exp\left(-\frac{1}{2}\|\boldsymbol{z}\|^2\right)\left(\sum_{n=0}^{\infty}\frac{(\boldsymbol{x}\cdot\boldsymbol{z})^n}{n!}\right)$$

from the Taylor series expansion of $\exp(\boldsymbol{x} \bullet \boldsymbol{z})$

The RBF kernel illustrated

$\gamma = -10$ $\gamma = -100$ $\gamma = -1000$



Figures from openclassroom.stanford.edu (Andrew Ng)

What makes a valid kernel?

- $k(\boldsymbol{x}, \boldsymbol{z})$ is a valid kernel if there is some ϕ such that

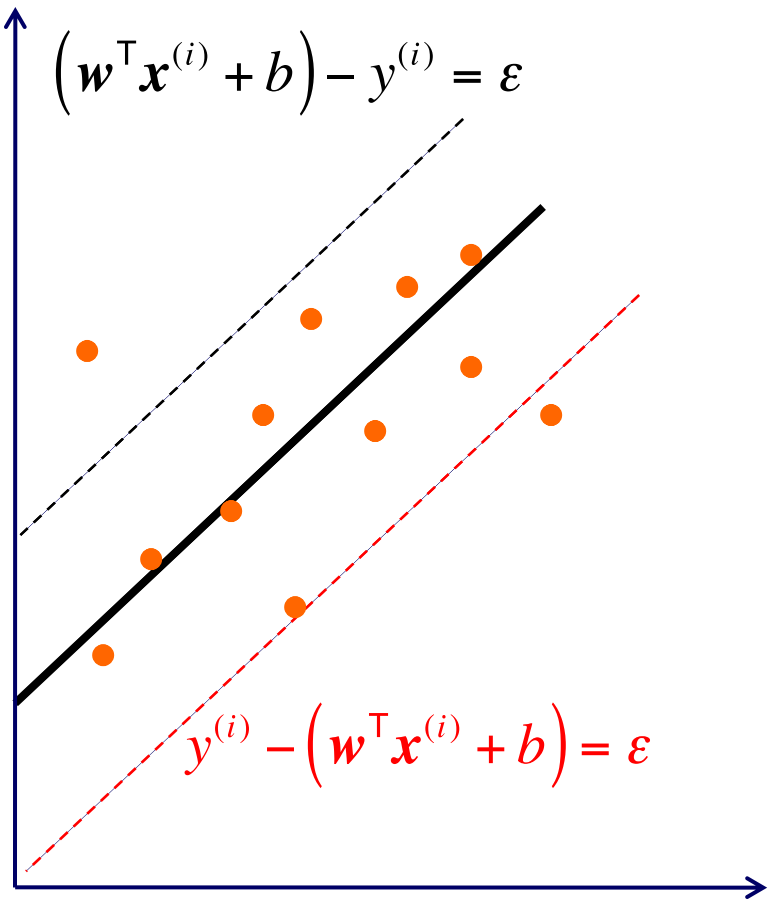
$$k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$$

- this holds for a symmetric function $k(\boldsymbol{x}, \boldsymbol{z})$ if and only if the kernel matrix $\boldsymbol{K}$ is positive semidefinite for any training set (Mercer's theorem)

definition of positive semidefinite (p.s.d): $\forall \boldsymbol{v} : \boldsymbol{v}^{\top} \boldsymbol{K} \boldsymbol{v} \geq \boldsymbol{0}$

Support vector regression

- the SVM idea can also be applied in regression tasks
- an ε-insensitive error function specifies that a training instance is well explained if the model's prediction is within ε of $y^{(i)}$



Support vector regression

$$\underset{\boldsymbol{w},b,\xi^{(1)}\ldots\xi^{(m)},\hat{\xi}^{(1)}\ldots\hat{\xi}^{(m)}}{\text{minimize}} \quad \frac{1}{2}\|\,\boldsymbol{w}\,\|_2^2 + C\sum_{i=1}^{m}\left(\xi^{(i)} + \hat{\xi}^{(i)}\right)$$

subject to constraints:

$$\left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}^{(i)} + b\right) - y^{(i)} \leq \varepsilon + \xi^{(i)}$$

$$y^{(i)} - \left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}^{(i)} + b\right) \leq \varepsilon + \hat{\xi}^{(i)}$$

$$\xi^{(i)}, \hat{\xi}^{(i)} \geq 0$$

for $i = 1,\ldots, m$

slack variables allow predictions for some training instances to be off by more than ε

Learning theory justification for maximizing the margin

$$error_{\mathrm{D}}(h) \leq error_{\mathrm{D}}(h) + \sqrt{\frac{VC\left(\log\frac{2m}{VC}+1\right)+\log\frac{4}{\delta}}{m}}$$

error on true distribution

training set error

VC-dimension of hypothesis class

Vapnik showed there is a connection between the margin and VC dimension

$$VC \leq \frac{4R^2}{\mathrm{margin}_D(h)^2}$$

constant dependent on training data

thus to minimize the VC dimension (and to improve the error bound) ➔ maximize the margin

The power of kernel functions

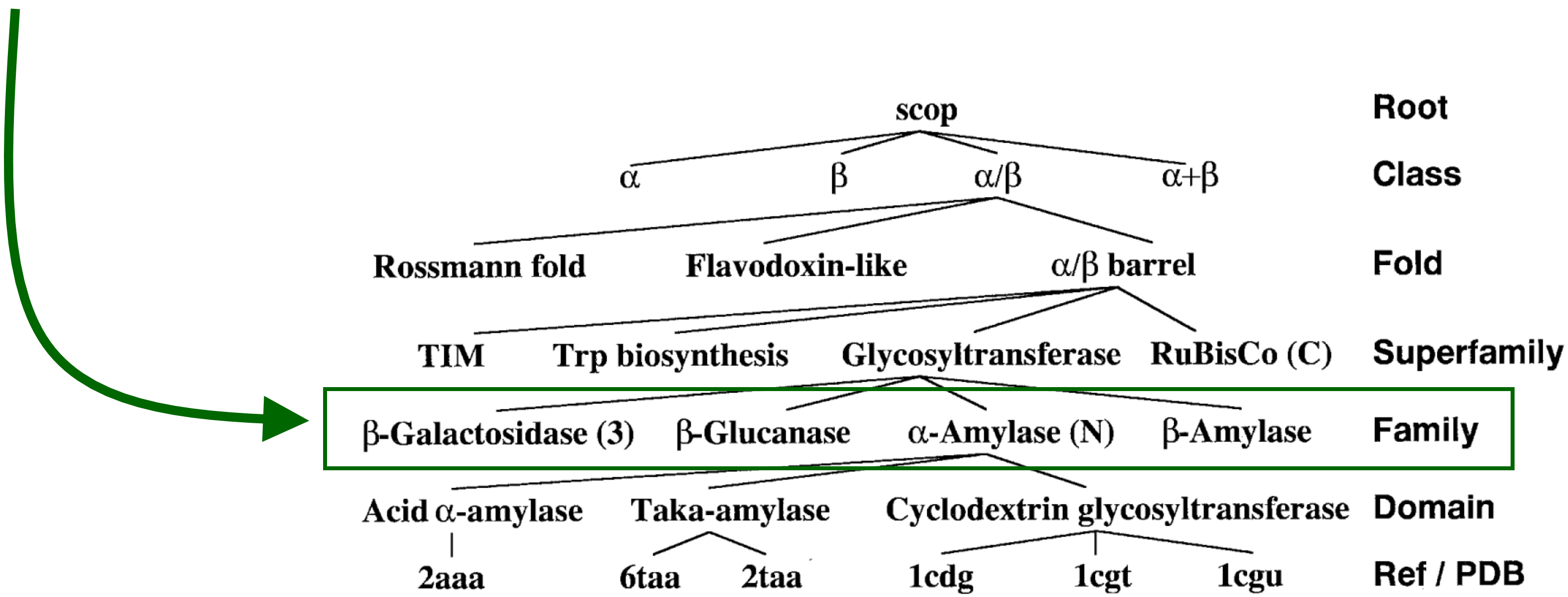
- kernels can be designed and used to represent complex data types such as
  - strings
  - trees
  - graphs
  - etc.
- let’s consider a specific example




The protein classification task

Given: amino-acid sequence of a protein
Do: predict the *family* to which it belongs

GDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVCVLAHHFGKEFTPPVQAAYAKVVAGVANALAHKYH



The k-spectrum feature map

- we can represent sequences by counts of all of their k-mers

$$\boldsymbol{x} = \texttt{AKQDYYYYEI}$$

$$\Big\downarrow \; k = 3$$

$$\begin{array}{lcccccccccc} \phi(\boldsymbol{x}) = (& 0, & 0, & \ldots, & 1, & \ldots, & 1, & \ldots, & 2 &) \\ & \texttt{AAA} & \texttt{AAC} & \ldots & \texttt{AKQ} & \ldots & \texttt{DYY} & \ldots & \texttt{YYY} & \end{array}$$

- the dimension of $\phi(\boldsymbol{x}) = |A|^k$ where $|A|$ is the size of the alphabet
  - using 6-mers for protein sequences, $|20|^6 = 64$ million
  - almost all of the elements in $\phi(\boldsymbol{x})$ are 0 since a sequence of length l has at most l-k+1 k-mers

The k-spectrum kernel

- consider the k-spectrum kernel applied to $\boldsymbol{x}$ and $\boldsymbol{z}$ with $k = 3$

$\boldsymbol{x}$ = AKQDYYYYEI

$\boldsymbol{z}$ = AKQIAKQYEI

$$\phi(\boldsymbol{x}) = (\underset{\texttt{AAA}}{0}, \underset{\ldots}{\ldots}, \underset{\texttt{AKQ}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{YEI}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{YYY}}{0})$$

$$\phi(\boldsymbol{z}) = (\underset{\texttt{AAA}}{0}, \underset{\ldots}{\ldots}, \underset{\texttt{AKQ}}{2}, \underset{\ldots}{\ldots}, \underset{\texttt{YEI}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{YYY}}{0})$$

$$\phi(\boldsymbol{x}) \bullet \phi(\boldsymbol{z}) = 2 + 1 = 3$$

(k, m)-mismatch feature map

- closely related protein sequences may have few exact matches, but many near matches
- the (k, m)-mismatch feature map uses the k-spectrum representation, but allows up to m mismatches

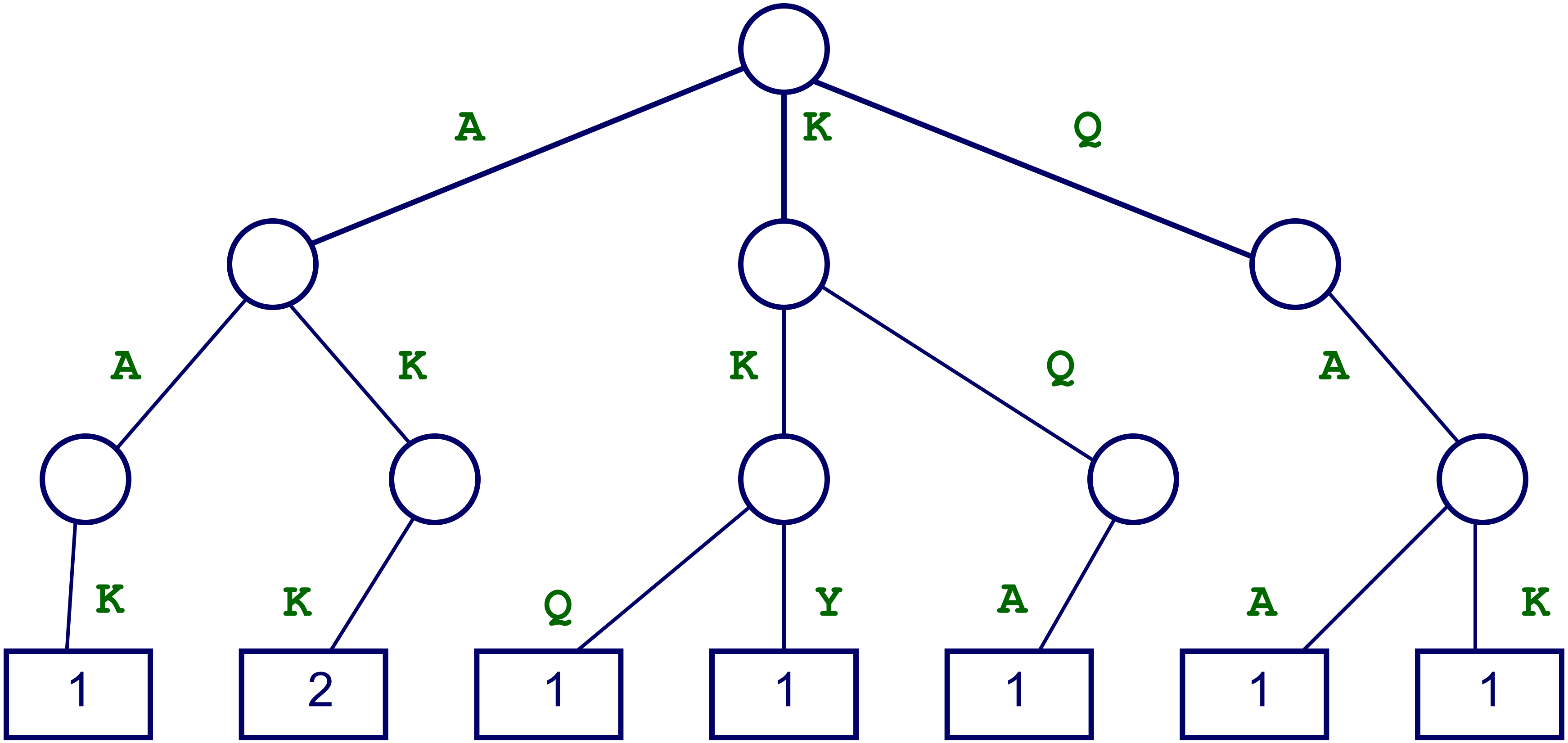
$$\boldsymbol{x} = \texttt{AKQ}$$

$$\Big\Downarrow \; k = 3, m = 1$$

$$\phi(\boldsymbol{x}) = (\underset{\texttt{AAA}}{0}, \ldots, \underset{\texttt{AAQ}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{AKQ}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{DKQ}}{1}, \underset{\ldots}{\ldots}, \underset{\texttt{YYY}}{0})$$

Using a trie to represent 3-mers

- example: representing all 3-mers of the sequence **QAAKKQAKKY**



Computing the kernels efficiently with tries [Leslie et al., *NIPS* 2002]

***k*-spectrum kernel**

- for each sequence
  - build a trie representing its k-mers
- compute kernel $\phi(\boldsymbol{x}) \bullet \phi(\boldsymbol{z})$ by traversing trie for $\boldsymbol{x}$ using k-mers from $\boldsymbol{z}$
  - update kernel function when reaching a leaf

(*k*, *m*)-mismatch kernel

- for each sequence
  - build a trie representing its k-mers and also k-mers with at most m mismatches
- compute kernel $\phi(\boldsymbol{x}) \bullet \phi(\boldsymbol{z})$ by traversing trie for $\boldsymbol{x}$ using k-mers from $\boldsymbol{z}$
  - update kernel function when reaching a leaf

scales linearly with sequence length: $O\left(k^{m+1}|A|^m\left(|\boldsymbol{x}|+|\boldsymbol{z}|\right)\right)$

Kernel algebra

- given a valid kernel, we can make new valid kernels using a variety of operators

kernel composition	mapping composition
$k(\boldsymbol{x},\boldsymbol{v}) = k_a(\boldsymbol{x},\boldsymbol{v}) + k_b(\boldsymbol{x},\boldsymbol{v})$	$\phi(\boldsymbol{x}) = \left(\phi_a(\boldsymbol{x}),\ \phi_b(\boldsymbol{x})\right)$
$k(\boldsymbol{x},\boldsymbol{v}) = \gamma\ k_a(\boldsymbol{x},\boldsymbol{v}),\ \gamma > 0$	$\phi(\boldsymbol{x}) = \sqrt{\gamma}\ \phi_a(\boldsymbol{x})$
$k(\boldsymbol{x},\boldsymbol{v}) = k_a(\boldsymbol{x},\boldsymbol{v})k_b(\boldsymbol{x},\boldsymbol{v})$	$\phi_l(\boldsymbol{x}) = \phi_{ai}(\boldsymbol{x})\phi_{bj}(\boldsymbol{x})$
$k(\boldsymbol{x},\boldsymbol{v}) = \boldsymbol{x}^{\mathsf{T}} A\boldsymbol{v},\quad A \text{ is p.s.d.}$	$\phi(\boldsymbol{x}) = L^{\mathsf{T}}\boldsymbol{x},\ \text{where } A = LL^{\mathsf{T}}$
$k(\boldsymbol{x},\boldsymbol{v}) = f(\boldsymbol{x})f(\boldsymbol{v})k_a(\boldsymbol{x},\boldsymbol{v})$	$\phi(\boldsymbol{x}) = f(\boldsymbol{x})\phi_a(\boldsymbol{x})$

Comments on SVMs

- we can find solutions that are globally optimal (maximize the margin)
  - because the learning task is framed as a convex optimization problem
  - no local minima, in contrast to multi-layer neural nets
- there are two formulations of the optimization: *primal* and *dual*
  - dual represents classifier decision in terms of support vectors
  - dual enables the use of kernel functions
- we can use a wide range of optimization methods to learn SVM
  - standard quadratic programming solvers
  - SMO [Platt, 1999]
  - linear programming solvers for some formulations
  - etc.

Comments on SVMs

- kernels provide a powerful way to
  - allow nonlinear decision boundaries
  - represent/compare complex objects such as strings and trees
  - incorporate domain knowledge into the learning task
- using the kernel trick, we can implicitly use high-dimensional mappings without explicitly computing them
- one SVM can represent only a binary classification task; multi-class problems handled using multiple SVMs and some encoding
  - one class vs. rest
  - ECOC
  - etc.
- empirically, SVMs have shown state-of-the art accuracy for many tasks
- the kernel idea can be extended to other tasks (anomaly detection, regression, etc.)