Linear Discriminant Functions

- General form:

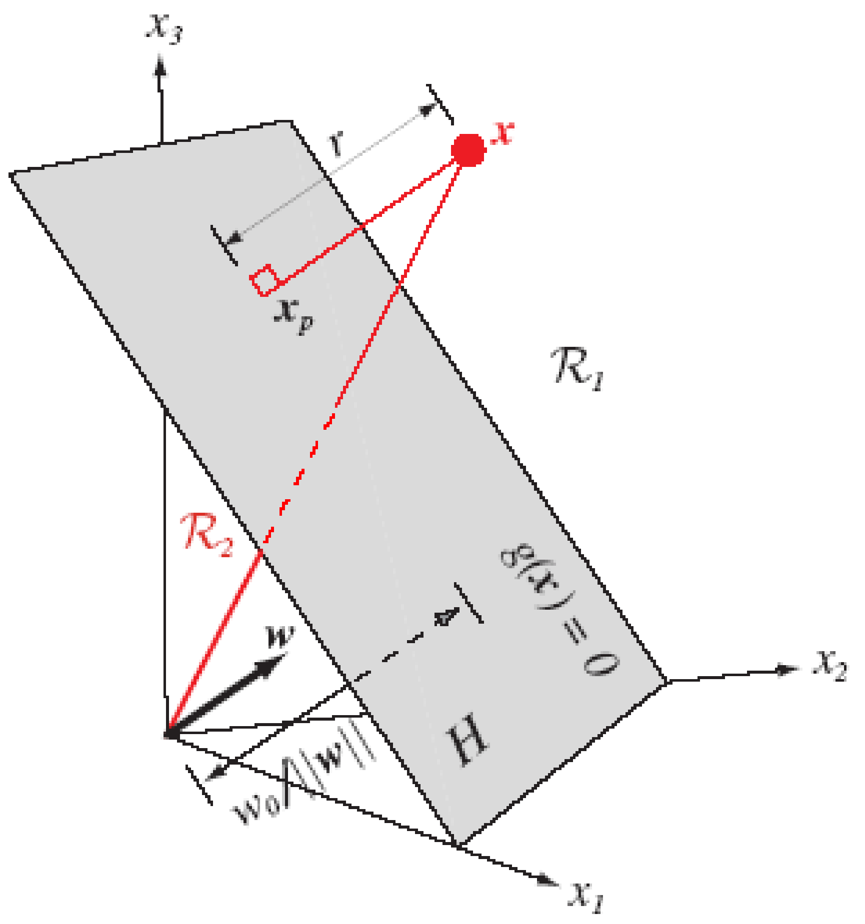
$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

w is the *weight vector*, and

w_0 is the *bias* or *threshold weight*

- 2 classes:
  - One function, g(**x**), and then:
  - decide ω_1 if $g(\mathbf{x}) > 0$, and
    ω_2 otherwise
  - if $g(\mathbf{x}) = 0$, decide arbitrarily

w gives the "orientation" of *H*, and
w_0 gives the "distance" from the origin to *H*



Fisher's Classifier

- Given a dataset with labeled samples:

$$D = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{1n_1}, \mathbf{x}_{21}, \mathbf{x}_{22}, \ldots, \mathbf{x}_{2n_2}\}$$

- Two data subsets (or datasets)

$$D_1 = \{\mathbf{x}_{11}, \mathbf{x}_{12}, \ldots, \mathbf{x}_{1n_1}\}$$

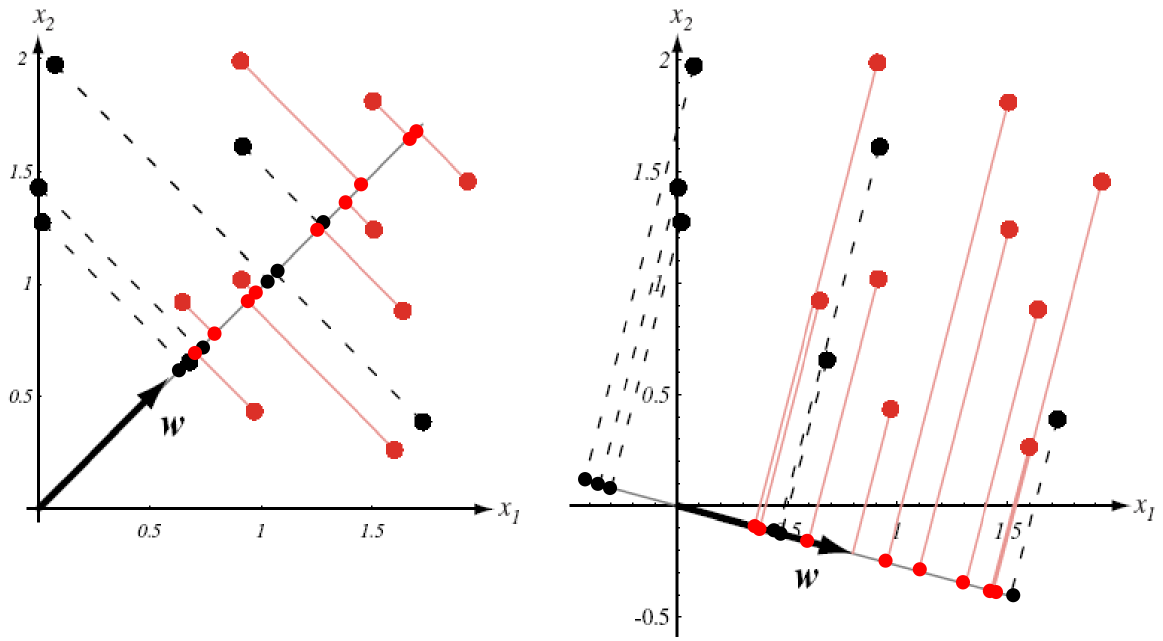
$$D_2 = \{\mathbf{x}_{21}, \mathbf{x}_{22}, \ldots, \mathbf{x}_{2n_2}\}$$

  samples $\in$ to ω_1 and ω_2 respectively, where $n = n_1 + n_2$
- Perform a linear transformation of each **x**: $y_{ij} = \mathbf{w}^t \mathbf{x}_{ij}$
- obtaining two new datasets

$$Y_1 = \{y_{11}, y_{12}, \ldots, y_{1n_1}\}$$ and

$$Y_2 = \{y_{21}, y_{22}, \ldots, y_{2n_2}\}$$

- ||**w**|| = 1 means that each y_{ij} is the projection of $\mathbf{x}_{ij}$ onto a line in the direction of **w**
- For now, don't care about the *magnitude* of **w**



Naïve approach

- What is the best direction of **w**?
- **Define:** *sample mean*, $\mathbf{m}_i$, as:

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in D_i} \mathbf{x}$$

- Then, sample mean in the *projected line* is:

$$\tilde{m}_i = \frac{1}{n_i} \sum_{y \in Y_i} y$$

$$= \frac{1}{n_i} \sum_{\mathbf{x} \in D_i} \mathbf{w}^t \mathbf{x} = \mathbf{w}^t \mathbf{m}_i$$

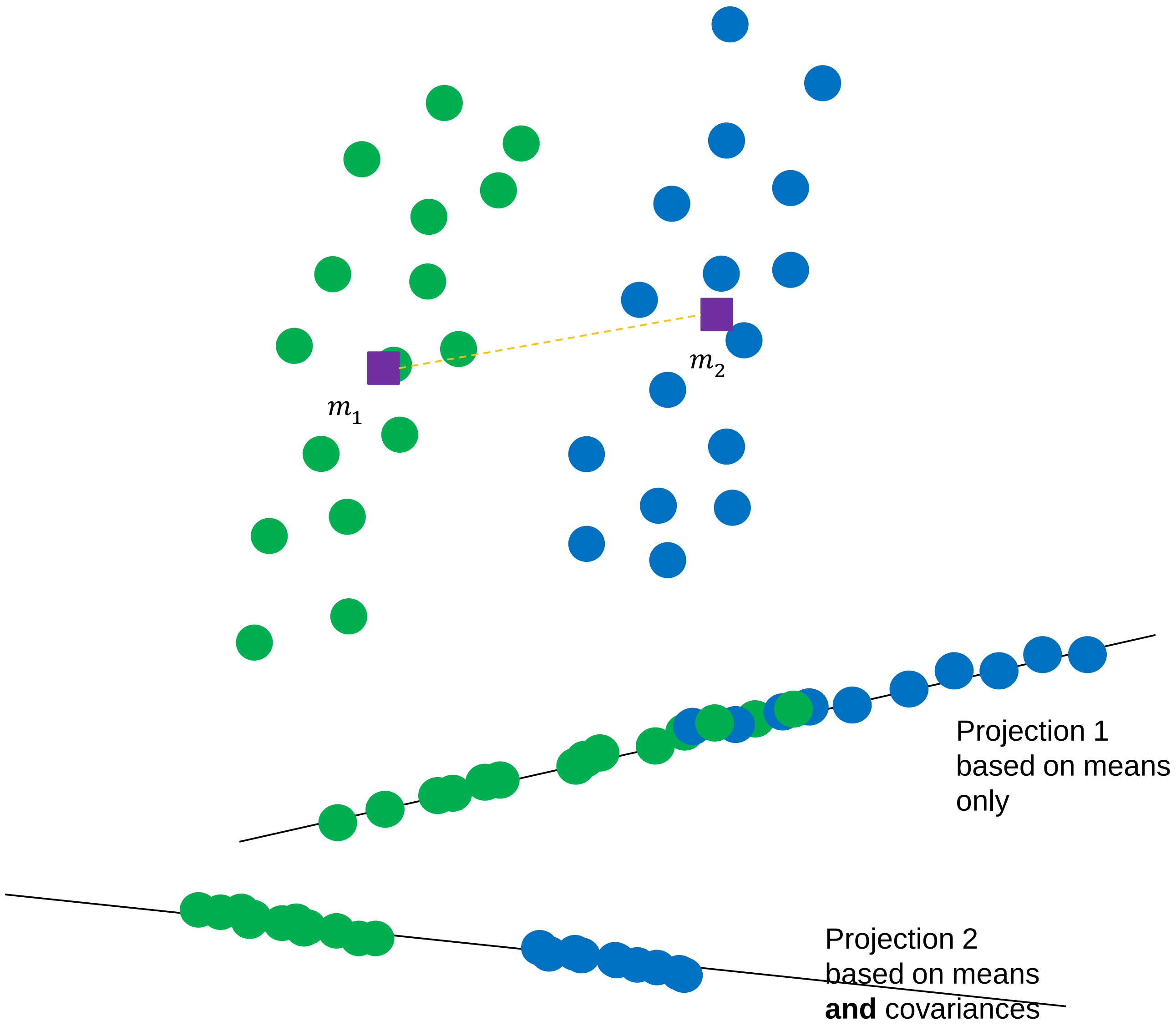
- Thus, $\tilde{m}_i$ is the projection of $\mathbf{m}_i$

- The distance between the projected means is given by:

$$\left| \tilde{m}_1 - \tilde{m}_2 \right| = \left| \mathbf{w}^t (\mathbf{m}_1 - \mathbf{m}_2) \right| \qquad (1)$$

- Then, different **w**'s will give us different "between-mean" values, and
- we want to maximize the class "separability"
- Find a vector **w** that maximizes (1)

Is this Naïve approach good enough?



- Use the variances $\tilde{s}_i^2 = \frac{1}{n_i} \sum_{y \in Y_i} (y - \tilde{m}_i)^2$
- *within-class variance*: $\tilde{s}_1^2 + \tilde{s}_2^2$
- Find a vector **w** that maximizes:

$$J(\mathbf{w}) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} \qquad (2)$$

- $J(.)$ has to be expressed in terms of **w**
- Sample *covariance matrix*:

$$\mathbf{S}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$$

and the *within-class covariance matrix*:

$$\mathbf{S}_W = \frac{1}{n}\left(n_1 \mathbf{S}_1 + n_2 \mathbf{S}_2\right) = p_1 \mathbf{S}_1 + p_2 \mathbf{S}_2$$

where $p_i = \frac{n_i}{n}$

- Express the sum of the covariances as:

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \mathbf{w}^t \mathbf{S}_W \mathbf{w}$$

- Separation of projected means as:

$$\left(\tilde{m}_1 - \tilde{m}_2\right)^2 = \mathbf{w}^t \mathbf{S}_B \mathbf{w}$$

where

$$\mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t$$

is the *between-class scatter matrix*.

- Thus, we can write $J(.)$ in terms of $\mathbf{S}_B$ and $\mathbf{S}_W$ as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^t \mathbf{S}_B \mathbf{w}}{\mathbf{w}^t \mathbf{S}_W \mathbf{w}} \qquad (3)$$

- The vector **w** that maximizes *J*(.) must satisfy: $\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w}$
- If $\mathbf{S}_W$ is *not singular*:

$$\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{w} = \lambda\mathbf{w}$$

- The solution **w** is an eigenvector of $\mathbf{S}_W^{-1}\mathbf{S}_B$
- Since $\mathbf{S}_B\mathbf{w}$ is in the direction of $\mathbf{m}_1 - \mathbf{m}_2$, and

  λ is a "scaling factor" of **w**,

  the solution is given by

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$$

- Note: there is only one non-zero eigenvalue of $\mathbf{S}_W^{-1}\mathbf{S}_B$

- Once obtained the direction of **w**,
- Complete linear classifier:

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

  i.e., find w_0
- A naïve approach is to assume that the *class conditional* probabilities in the projected line are *identical*, and thus:

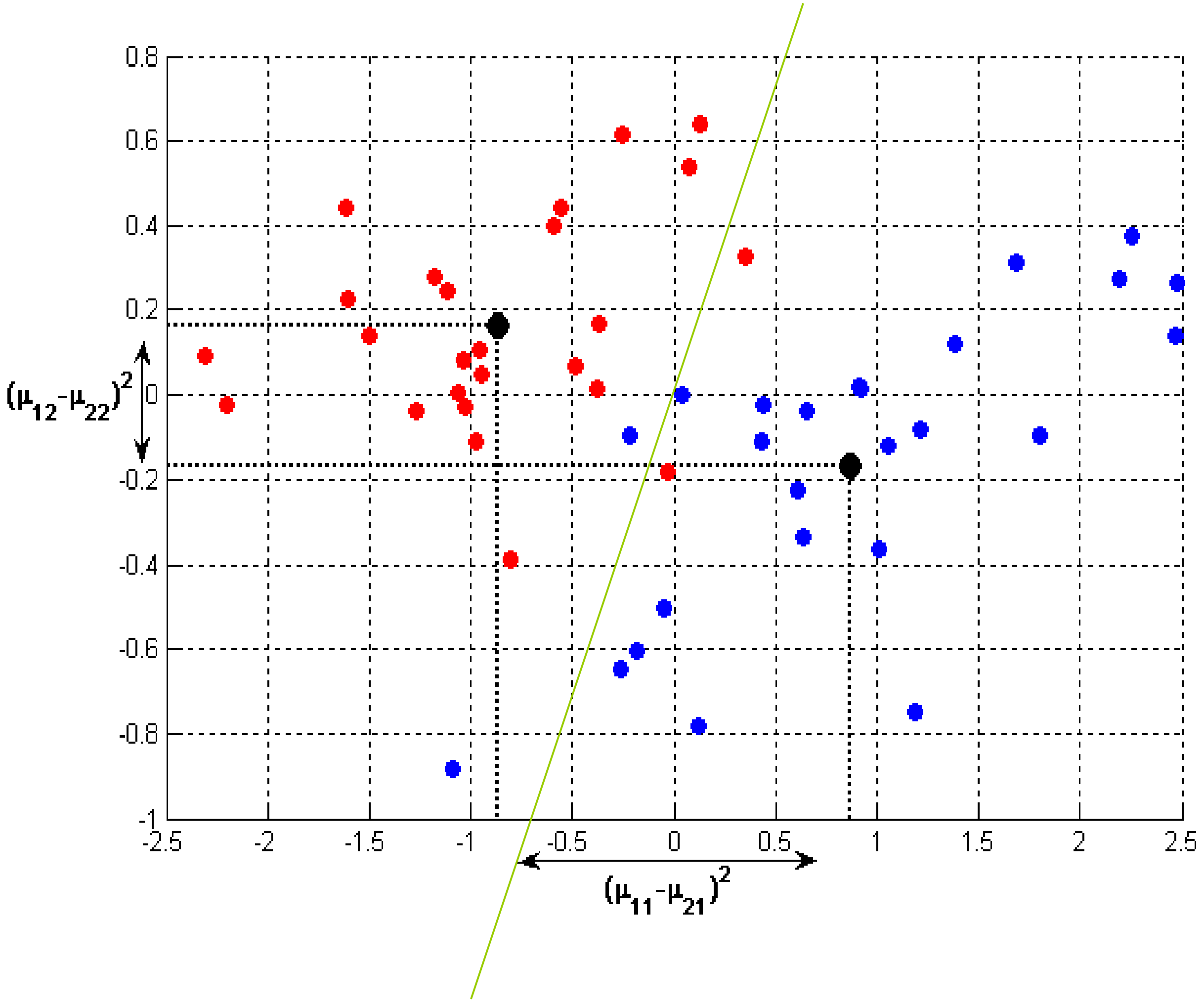
$$w_0 = -\frac{1}{2}(\tilde{m}_1 + \tilde{m}_2) - \ln\frac{P(\omega_1)}{P(\omega_2)}$$

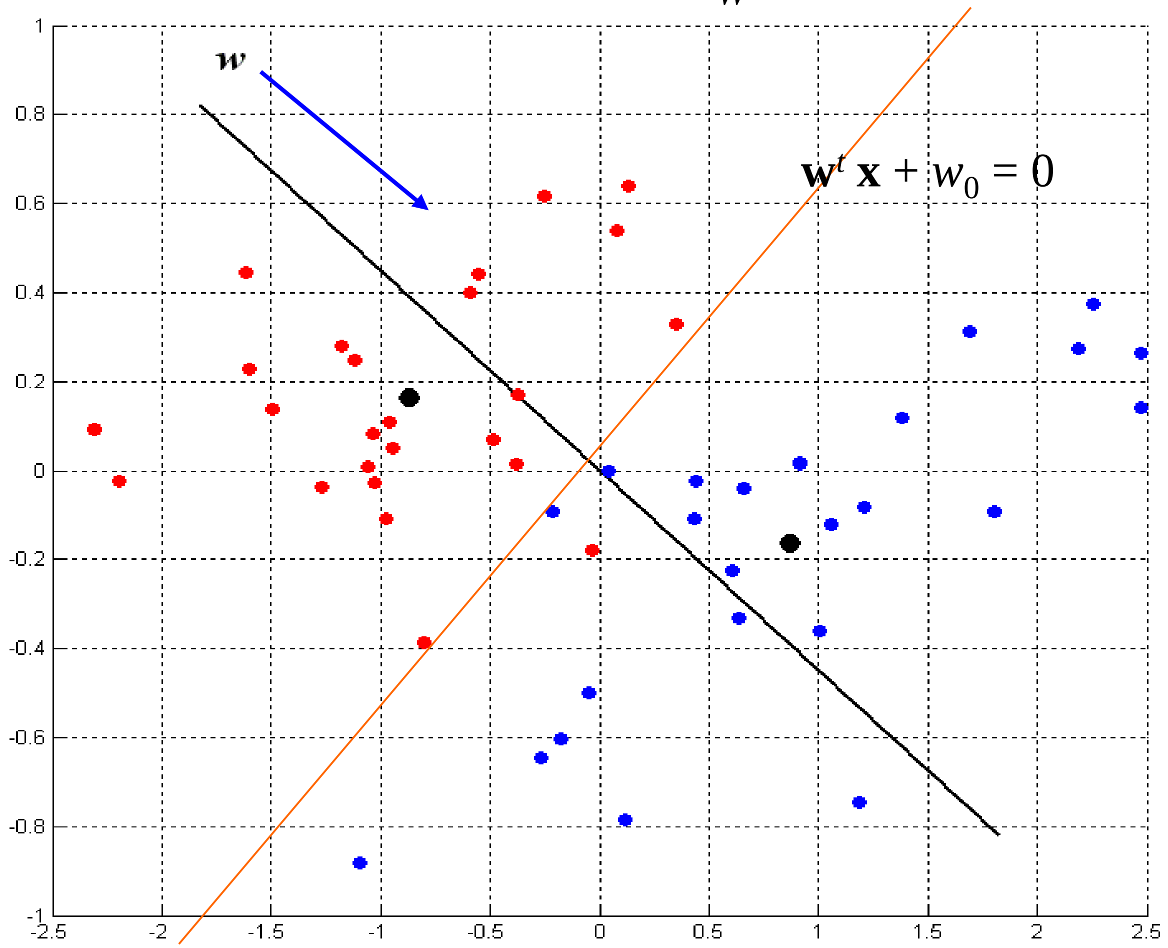
- w_0 is the middle point between the two sample means in the *projected* data

Example: means vs variances

$$J(\mathbf{w}) = |\tilde{m}_1 - \tilde{m}_2| = |\mathbf{w}^t(\mathbf{m}_1 - \mathbf{m}_2)|$$

$$J(\mathbf{w}) = \frac{\mathbf{w}^t \mathbf{S}_B \mathbf{w}}{\mathbf{w}^t \mathbf{S}_W \mathbf{w}}$$





Example 2

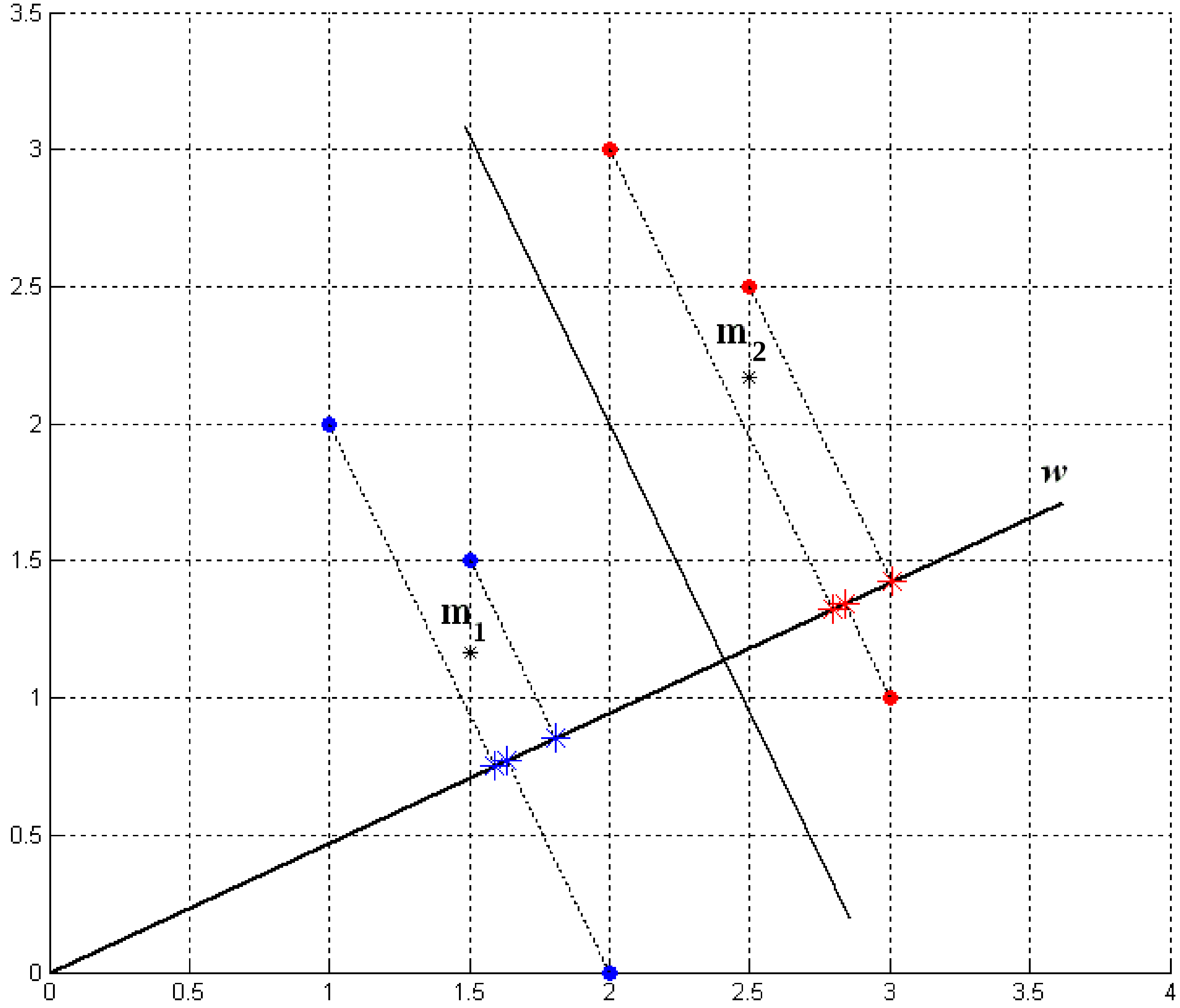
$$D_1 = \begin{bmatrix} 1.5 & 1.5 \\ 1 & 2 \\ 2 & 0 \end{bmatrix} \quad D_2 = \begin{bmatrix} 3 & 1 \\ 2 & 3 \\ 2.5 & 2.5 \end{bmatrix} \quad p_1 = p_2 = 0.5$$

$$\mathbf{m}_1 = \begin{bmatrix} 1.5 \\ 1.1667 \end{bmatrix} \mathbf{m}_2 = \begin{bmatrix} 2.5 \\ 2.1667 \end{bmatrix} \mathbf{S}_1 = \begin{bmatrix} 0.25 & -0.5 \\ -0.5 & 1.0833 \end{bmatrix} \mathbf{S}_2 = \begin{bmatrix} 0.25 & -0.5 \\ -0.5 & 1.0833 \end{bmatrix}$$

$$\mathbf{S}_W = p_1\mathbf{S}_1 + p_2\mathbf{S}_2 = \begin{bmatrix} 0.25 & -0.5 \\ -0.5 & 1.0833 \end{bmatrix} \qquad S_w^{-1} = \begin{bmatrix} 52 & 24 \\ 24 & 12 \end{bmatrix}$$

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2) = \begin{bmatrix} -76 \\ -36 \end{bmatrix}$$

Example 2 Graphically



Generalized Linear Classifiers

Augmented Feature Vector

- We've seen that the form of a linear classifier is:

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

- we can write this function as:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i$$

where $x_0 = 1$

- In NN, w_0 is called "bias"

- The *augmented feature vector*, **y**, is:

$$\mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

- Similarly, the *augmented weight vector*:

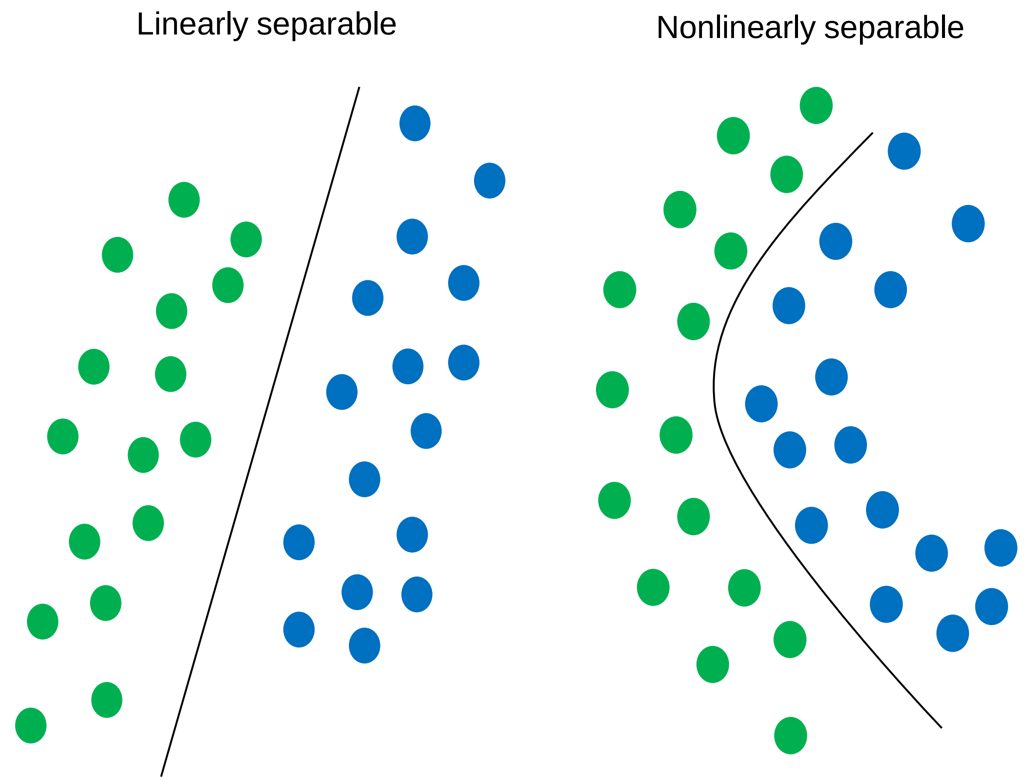
$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$$

- This implies a mapping from the *d*-dim. space onto the (*d*+1)-dim. space
- Classifier: $\mathbf{a}^t \mathbf{y} = 0$

Linearly Separable Classes

- A set of n samples $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$, labeled either ω_1 or ω_2
- Aim: find the weights of $\mathbf{a}$, used in the linear classifier $g(\mathbf{x}) = \mathbf{a}^t\,\mathbf{y}$
- A solution $\mathbf{a}$ that correctly classifies *all* the training samples, implies

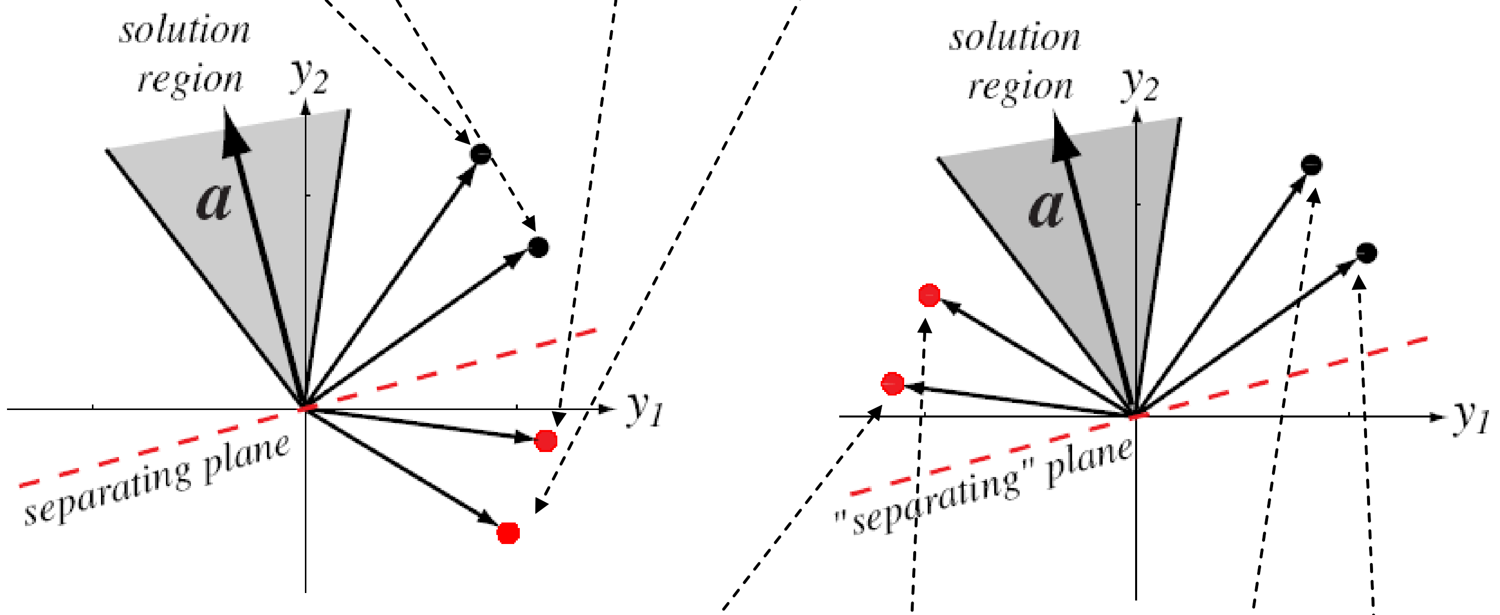
  $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n$ are *linearly separable*
  - $\mathbf{y}_i \in \omega_1$ **correctly** classified if $\mathbf{a}^t\,\mathbf{y}_i > 0$,
  - $\mathbf{y}_i \in \omega_2$ … …. if $\mathbf{a}^t\,\mathbf{y}_i < 0$



Normalization

- Replace all samples of ω_2 by their *negatives*

ω_1: $\{[2,3]^t,[3,2]^t\}$; ω_2: $\{[3,-0.5]^t,[2,-1.5]^t\}$



Normalized samples:

ω_2: $\{[-3, 0.5]^t,[-2,1.5]^t\}$; ω_1: $\{[2,3]^t,[3,2]^t\}$

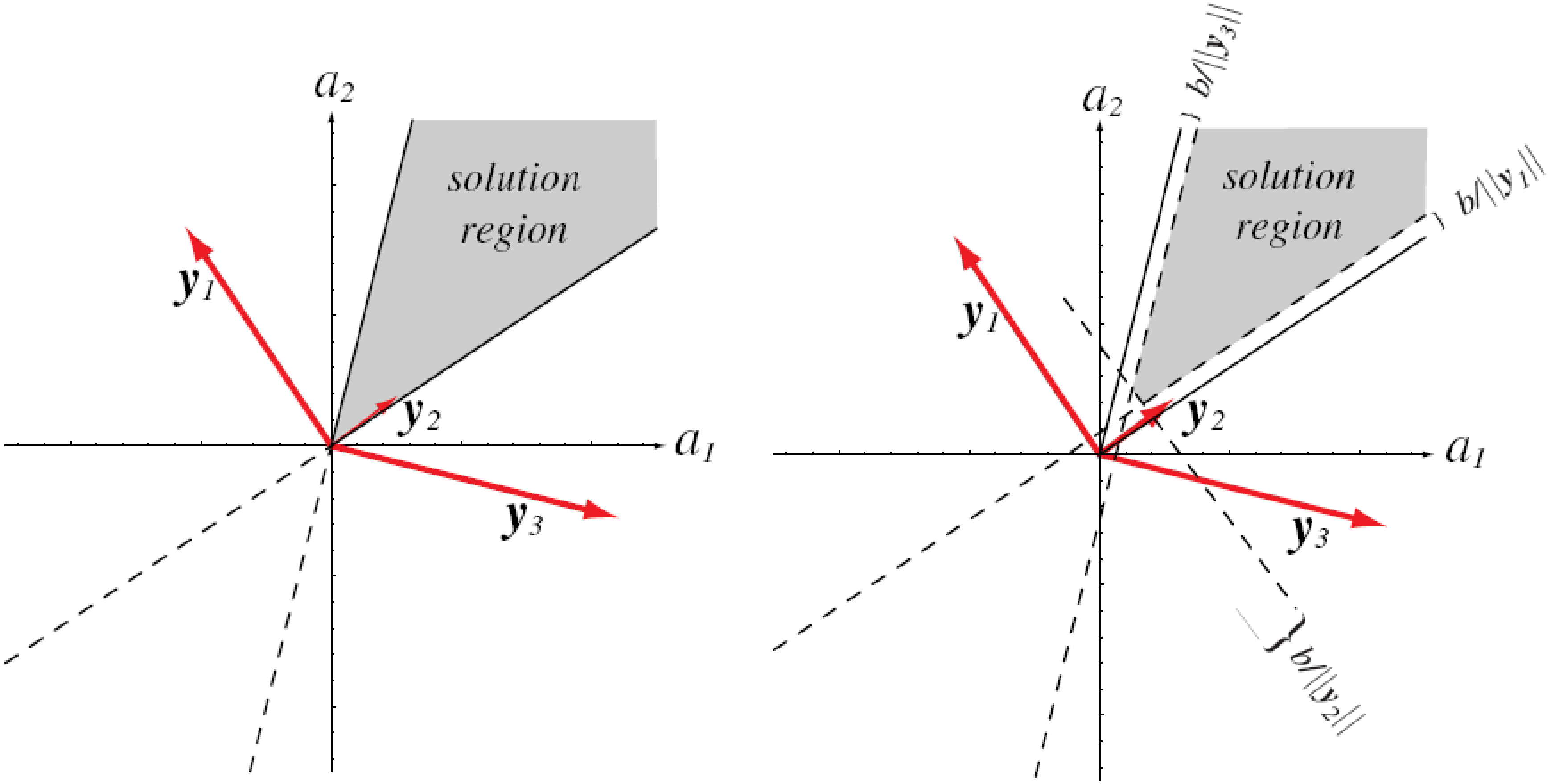
- It is then clear that if a solution vector **a** exists

  it may not be **unique**
- Thus, many approaches can be proposed
- **One way** is to look at a unit-length vector **a**

  that *maximizes* the *minimum* distance from the

  hyperplane to the samples
- **Another way:** Seek for the minimum-length vector **a** that satisfies $\mathbf{a}^t \mathbf{y}_i > b$
  - for all $\mathbf{y}_i$ and
  - for some positive value b.

- Say, we have $D = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$, samples in 2D space
- … three samples… already normalized…



Approaches for Linearly Separable Case

- Gradient descent procedures
- Newton's algorithm
- The perceptron criterion function/algorithm
- Relaxation procedures
  - Batch relaxation
  - Single-sample relaxation
- Minimum squared-error procedures
- Widrow-Hoff or LMS procedure
- Ho-Kashyap procedure
- Support vector machine (SVM)
- … many others

The Perceptron Criterion

- Find a criterion function to solve the set of inequalities $\mathbf{a}^t\ \mathbf{y}_i > 0$
- Solution: $J(.)$ is the number of samples "misclassified" by **a**.
- It gives a "piecewise" criterion function, for which the gradient descent algorithm can be used
- The *perceptron criterion function*:

$$J_p(\mathbf{a}) = \sum_{\mathbf{y}\in Y(\mathbf{a})} (-\mathbf{a}^t\mathbf{y})$$

where $Y(\mathbf{a})$ is the set of samples *misclassified* by **a**.

- Normalizing all samples
- A sample **y** is misclassified if $\mathbf{a}^t\ \mathbf{y} \leq 0$
- implies that for all $\mathbf{y} \in Y(\mathbf{a})$,

$$\mathbf{a}^t\ \mathbf{y} \leq 0 \ \Rightarrow\ -\mathbf{a}^t\ \mathbf{y} > 0$$

- $J_p(\mathbf{a})$ is **either** positive or zero, but **never** negative
- $J_p(\mathbf{a})$ is zero only when $Y(\mathbf{a})$ is *empty*
- Geometric interpretation of $J_p(\mathbf{a})$:
  - It is $\propto$ to the *sum* of distances from the *misclassified* samples to the discriminant function given by **a**

- Use $J_p(\mathbf{a})$ in the Gradient Descent algorithm
- *Gradient operator* ∇:

$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{y} \in Y(\mathbf{a})} (-\mathbf{y})$$

  - because the gradient operator of J_p is $\partial J_p / \partial a_j$

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{y}^t \mathbf{x} = \mathbf{y}$$

- Update rule:

$$\mathbf{a}(k+1) \leftarrow \mathbf{a}(k) + \eta(k) \sum_{\mathbf{y} \in Y(k)} \mathbf{y}$$

where $Y(k)$ is the set of samples misclassified by $\mathbf{a}(k)$

Algorithm **Batch Perceptron**

Input: A threshold θ

begin Initialize **a**, $\eta(1)$, $k \leftarrow 0$

    repeat

        $k \leftarrow k + 1$

        $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in Y(k)} \mathbf{y}$

    until $\left| \eta(k) \sum_{\mathbf{y} \in Y(k)} \mathbf{y} \right| < \theta$

end

Variants

- Algorithm called “batch perceptron”
  - use all samples in $J(.)$
- “single-sample” perceptron:

  if $\mathbf{y}^k$ is misclassified by $\mathbf{a}$

  $$\mathbf{a} \leftarrow \mathbf{a} + \mathbf{y}^k \qquad (4)$$
- Repeated until no $\mathbf{y}^k$ is misclassified
- Proved to converge if the samples are

  “linearly separable”
- Can still be used in nonlinearly separable
- (4) is also called “fixed-increment” rule,
- since $\eta(.)$ is constant for all iterations

Relaxation procedures

- Descent criterion:

$$J_q(\mathbf{a}) = \sum_{\mathbf{y} \in Y} \left(\mathbf{a}^t \mathbf{y}\right)^2$$

- Regularization/normalization:

$$J_r(\mathbf{a}) = \frac{1}{2} \sum_{\mathbf{y} \in Y} \frac{\left(\mathbf{a}^t \mathbf{y} - b\right)^2}{||\mathbf{y}||^2}$$

- aka “minimum square error”
- Lead to a “smoother” surface of the optimization problem

Nonlinearly Separable Case

- Perceptron and relaxation are usually called

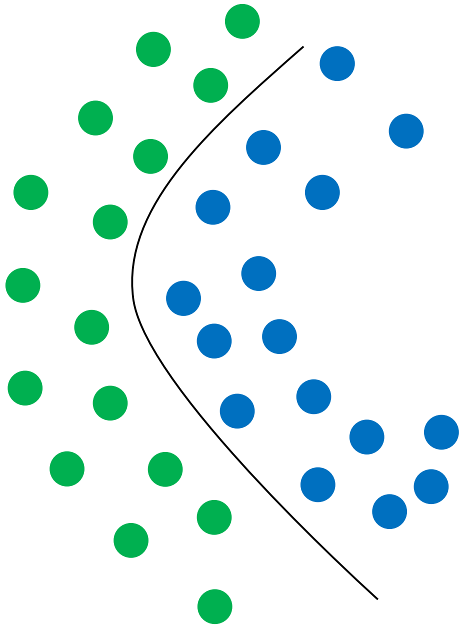
  error-correcting procedures
- It does work on nonlinearly separable
- In the nonlinearly separable case,

  relaxation procedures work better

$$J_r(\mathbf{a}) = \frac{1}{2} \sum_{\mathbf{y} \in Y} \frac{\left(\mathbf{a}^t \mathbf{y} - b\right)^2}{||\mathbf{y}||^2}$$

Solution using the

minimum squared-error procedure



Minimum Squared-error (MSE)

- *Normalizing* all samples:

  $\mathbf{a}^t\,\mathbf{y} > 0$ for all $\mathbf{y}$
- Consider $\mathbf{a}^t\,\mathbf{y}_i = b_i$

  where b_i is an *arbitrarily specified positive constant*
- Aim: find solution to

  $\mathbf{a}^t\,\mathbf{y}_i = b_i$

- Problem in matrix-like form: **Ya** = **b**

$$\begin{bmatrix} y_{10} & y_{11} & \cdots & y_{1d} \\ y_{20} & y_{21} & \cdots & y_{2d} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ y_{n0} & y_{n1} & \cdots & y_{nd} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

- **Y** is an $n \times (d+1)$ matrix
- The i^{th} row is vector $\mathbf{y}_i$ (normalized)
- **b** is a vector (of arbitrary constants)
- **a** is the weight vector

Aim: Find a solution vector **a** to **Ya** = **b**

- If **Y** is square, no problem…

  find **a** = $\mathbf{Y}^{-1}$ **b**, and we are done
- But in general, **Y** is *rectangular*
- Instead: minimize the error between **Ya** and **b**, as:

$$\mathbf{e} = \mathbf{Ya} - \mathbf{b}$$

or:
$$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

Gradient:
$$\nabla J_s = \sum_{i=1}^{n} 2(\mathbf{a}^t \mathbf{y}_i - b_i)\mathbf{y}_i = 2\mathbf{Y}^t(\mathbf{Ya} - \mathbf{b})$$

- **Solution:** $\mathbf{Y}^t$ **Y** is a square matrix, hopefully, nonsingular, we have:

$$\mathbf{a} = (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \mathbf{b} = \mathbf{Y}^\dagger \mathbf{b}$$

- We know what $\mathbf{Y}^\dagger$ is:
- the *pseudoinverse*
- The pseudoinverse, in general exists, but
- if it doesn't, we can find it as:

$$\mathbf{Y}^\dagger \equiv \lim_{\varepsilon \to 0} \left(\mathbf{Y}^t \mathbf{Y} + \varepsilon \mathbf{I} \right)^{-1} \mathbf{Y}^t$$

- It can be shown that the limit always exists, and **a** = $\mathbf{Y}^\dagger$ **b** is a solution to **Ya** = **b**
- Reasonable choice: **b** = $\mathbf{1}_n$

  a solution exists in most of the cases

Example

- Class 1: $[1,2]^t$ and $[2,0]^t$
- Class 2: $[3,1]^t$ and $[2,3]^t$
- Decision boundary:

$$\mathbf{a}^t \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = 0$$

- Lets find **a**
- Create

$$\mathbf{Y} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 0 \\ -1 & -3 & -1 \\ -1 & -2 & -3 \end{bmatrix}$$

- and find

$$\mathbf{Y}^+ = (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t = \begin{bmatrix} 5/4 & 13/12 & 3/4 & 7/12 \\ -1/2 & -1/6 & -1/2 & -1/6 \\ 0 & -1/3 & 0 & -1/3 \end{bmatrix}$$

- Set the margins $\mathbf{b} = [1,1,1,1]^t$
- Solution $\mathbf{a} = \mathbf{Y}^\dagger \mathbf{b} = [11/3,-4/3,-2/3]^t$

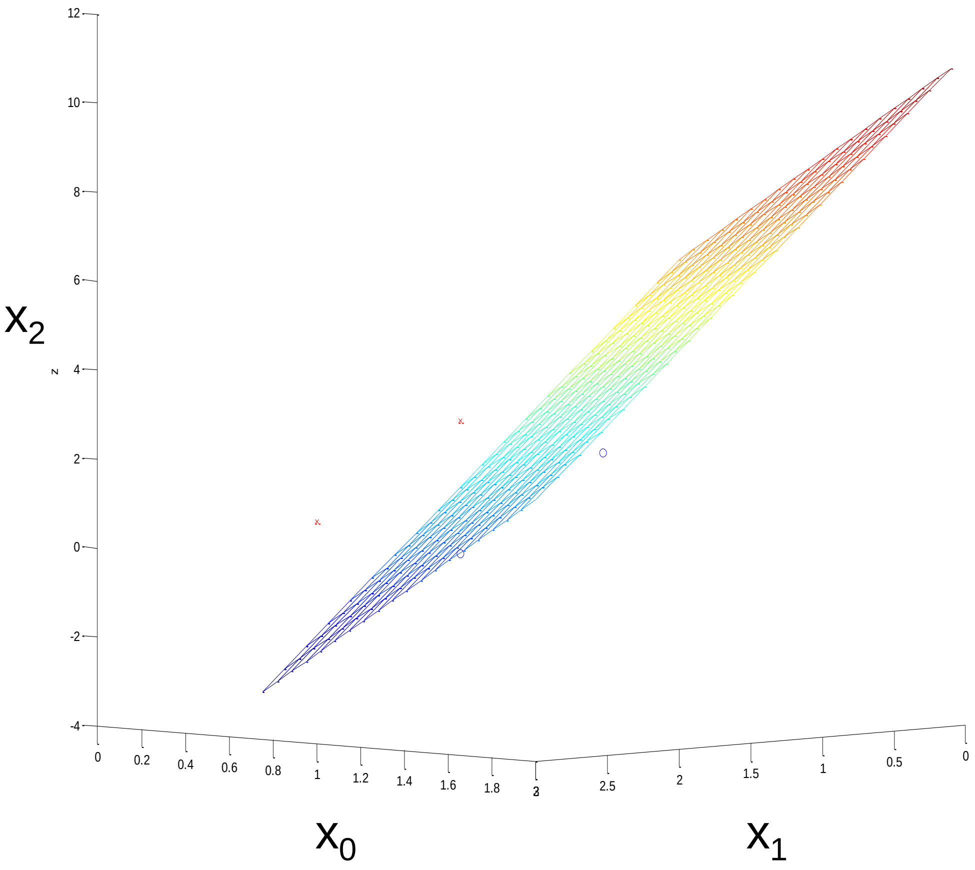
$$g(\mathbf{y}) = \frac{11}{3}x_0 - \frac{4}{3}x_1 - \frac{2}{3}x_2 = 0$$

Decide ω_1 if $g(\mathbf{y}) > 0$,

otherwise decide ω_2 when $g(\mathbf{y}) \le 0$

- Classify: $\mathbf{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

- Augmented vector: $\mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$

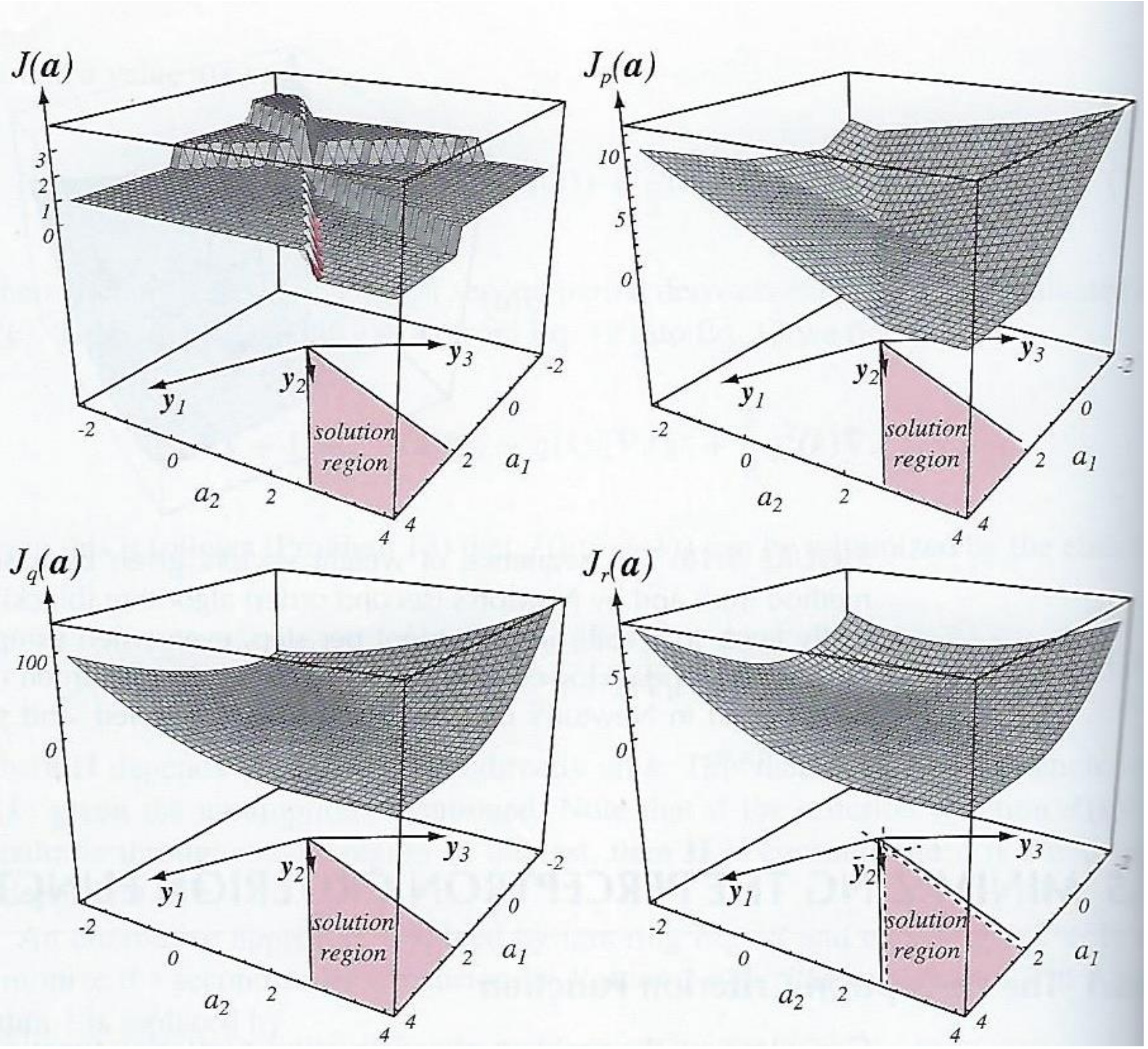


$$g(\mathbf{y}) = \frac{11}{3} - \frac{4}{3}2 - \frac{2}{3}2 = \frac{11}{3} - \frac{8}{3} - \frac{4}{3} = -\frac{1}{3} < 0$$

$$\Rightarrow \text{ decide } \mathbf{x} \in \omega_2$$

Optimization Landscape - Relaxation

Number of misclassified samples



$$J_p(\mathbf{a}) = \sum_{\mathbf{y}\in Y(\mathbf{a})} (-\mathbf{a}^t\mathbf{y})$$

$$J_q(\mathbf{a}) = \sum_{\mathbf{y}\in Y} \left(\mathbf{a}^t\mathbf{y}\right)^2$$

$$J_r(\mathbf{a}) = \frac{1}{2}\sum_{\mathbf{y}\in Y} \frac{\left(\mathbf{a}^t\mathbf{y} - b\right)^2}{\|\mathbf{y}\|^2}$$

Support Vector Machines (SVM)

- Most widely-used technique for classification
- Suitable for small training datasets
- Works well with a large number of features
- Excellent generalization power

Main idea:

- Derive a *linear* (could be nonlinear) classifier on the basis of a convex optimization problem
- Make use of "support vectors" to derive the classifier

- Consider a dataset that contains *labeled* samples

  drawn from two classes, ω_1 and ω_2 :

$$D = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$$

- Let z_k be the "identifier" for the class,
- where for each $\mathbf{y}_i$:

$$z_k = \begin{cases} +1 & \text{if } \mathbf{y}_i \in \omega_1 \\ -1 & \text{if } \mathbf{y}_i \in \omega_2 \end{cases}$$

Augmented space

- As before, the linear classifier in the *augmented* space is:

$$g(\mathbf{y}) = \mathbf{a}^t \mathbf{y}$$

- Recall:
  - $a_0 = w_0$ (the threshold or bias)
  - $y_0 = 1$ (the threshold is not changed)
- **Two cases:**
  - Linearly separable classes
  - Nonlinearly separable classes

Linearly separable case:

- A *separating* hyperplane satisfies:

for $k = 1, \ldots, n$

- Though the margin is *any* positive distance from the hyperplane to the samples

Aim of the SVM:

- Find the separating hyperplane with the *largest margin*
- But using only the *support vectors*
- The *larger* the margin, the better the classifier

Linearly separable case:

- A *separating* hyperplane satisfies:

$$z_k g(\mathbf{y}_k) \geq 1$$

for k = 1, …, n

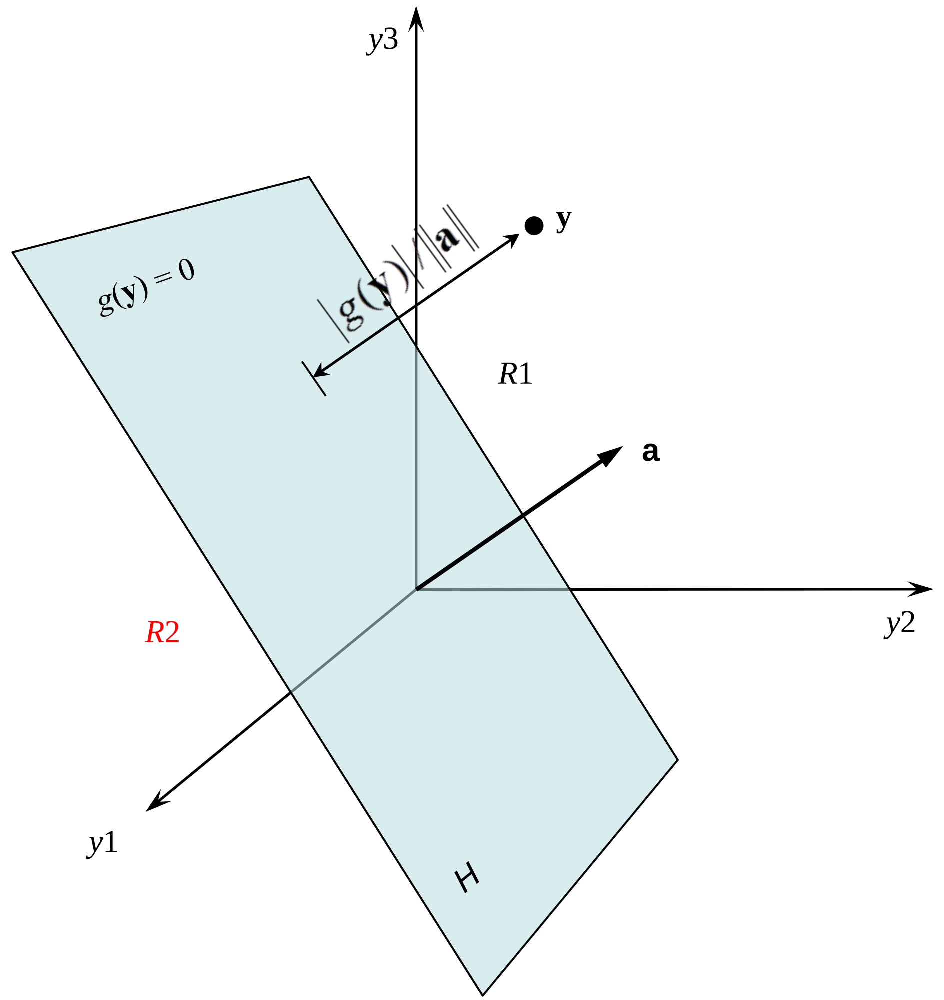
- Though the margin is *any* positive distance from the hyperplane to the samples

Aim of the SVM:

- Find the separating hyperplane with the *largest margin*
- But using only the *support vectors*
- The *larger* the margin, the better the classifier

Recall:

- The distance from any sample $\mathbf{y}$ to H: $|g(\mathbf{y})| / \|\mathbf{a}\|$
- Or equivalently $\frac{z_k g(\mathbf{y})}{\|\mathbf{a}\|}$



Support Vectors

Linearly separable case:

- A positive margin *b* exists,
- then…

Rewrite the problem:

- Find the vector(s) **a** that maximize(s) *b*, where:

$$\frac{z_k g(\mathbf{y}_k)}{\|\mathbf{a}\|} \geq b \qquad (2)$$

Note:

- The solution vector can be "scaled" and still preserve the *direction* of the hyperplane
- To *force* uniqueness of the solution, impose the constraint $b\,||\mathbf{a}|| = 1$
- In other words…

- Want to minimize $||\mathbf{a}||^2$
  while keeping the margin fixed.
- Why are they called *support vector machines*?
- They rely on the "support vectors" to derive the classifier.
- Which are the support vectors?
  … the ones whose margin is *minimal*

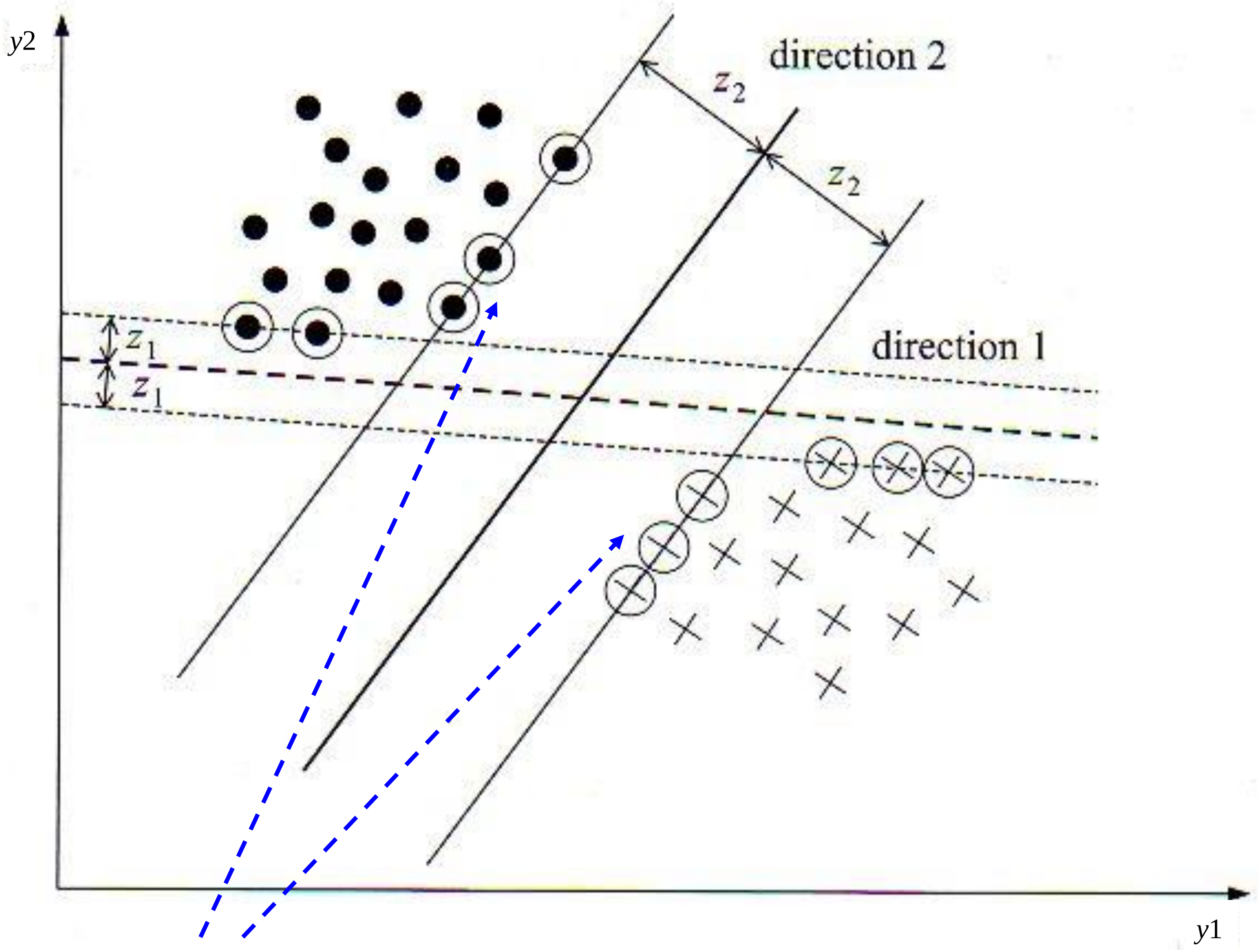
Or, put in other words:

- We intend to find samples that are the "hardest" to classify

Recall:

- We want to find the hyperplane that maximizes the margin

Graphically



support vectors

Reformulate the problem

- Lets keep the margin fixed, *say* b = 1

  and minimize $||\mathbf{a}||^2$
- Rewriting (2), yields a

  constrained optimization problem…
- Minimize:

$$\frac{1}{2}\|\mathbf{a}\|^2$$

- Subject to:

$$z_k \mathbf{a}^t \mathbf{y}_k = 1 \quad \text{or} \quad z_k \mathbf{a}^t \mathbf{y}_k - 1 = 0$$

  for all k = 1, .., n

Lagrange Multipliers

- Using *Lagrange undetermined multipliers*, transformed into an *unconstrained* optimization problem:
- Objective function:

$$L(\mathbf{a},\boldsymbol{\alpha}) = \underbrace{\frac{1}{2}\|\mathbf{a}\|^2}_{\text{Objective function}} - \sum_{k=1}^{n} \underbrace{\alpha_k}_{\text{Multipliers}} \underbrace{\left(z_k \mathbf{a}^t \mathbf{y}_k - 1\right)}_{\text{Constraints}} \qquad (3)$$

where $\alpha_k \geq 0$ are the undetermined multipliers

- Aim:
  - Minimize L with respect to **a**, and
  - Maximize it wrt to $\alpha_k \geq 0$

Dual Problem

Duality:

- Lagrangian treatment of optimization problems leads to an interesting dual representation, usually easier to solve than the “primal” problem

Besides:

- In the SVM, the so-called Karush-Kuhn-Tucker conditions allow us to use
  another representation of this problem,
  easier to solve, and
  involving fewer variables than the
  whole training set.

- Thus, (3) is now expressed in terms of α, as maximizing:

$$L(\boldsymbol{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j=1}^{n} \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k \tag{4}$$

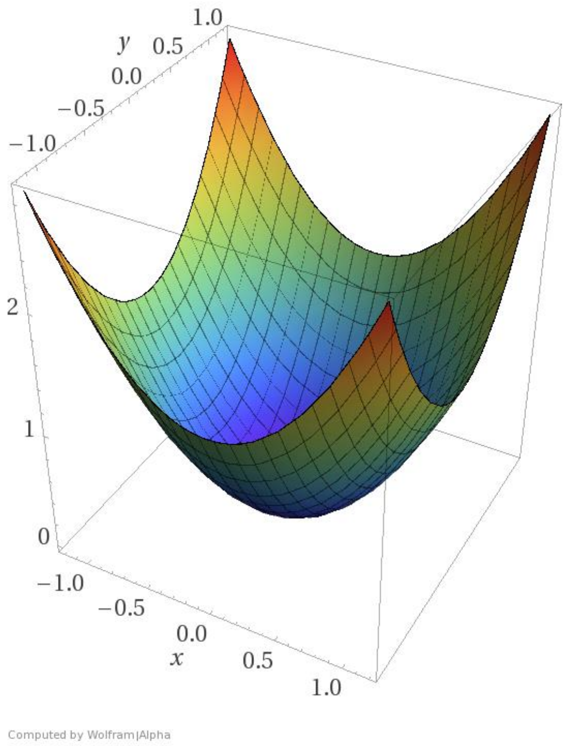
subject to the following constraints:

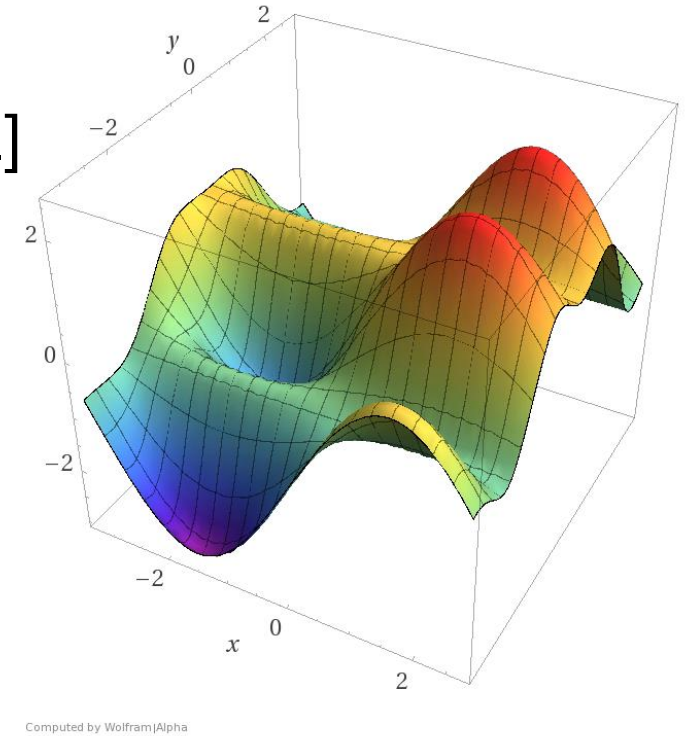
$$\sum_{k=1}^{n} z_k \alpha_k = 0, \quad \text{and } \alpha_k \geq 0$$

- Need to obtain values of α_k
- n_s (support) vectors are those for which $\alpha_k \neq 0$
- Soln. vector **a** given by: $\mathbf{a} = \sum_{k=1}^{n_s} \alpha_k z_k \mathbf{y}_k$
- Classification: $\mathbf{a}^t \mathbf{y} > 0$ decide $\mathbf{y} \in \omega_1$
  otherwise decide $\mathbf{y} \in \omega_2$

Convex Optimization

- Yields a *quadratic* constrained optimization problem.
- It is convex and the constraints are linear
  - form a convex set of feasible solutions
  - Has a unique global optima
- For more details see Ch. 21 of [1]

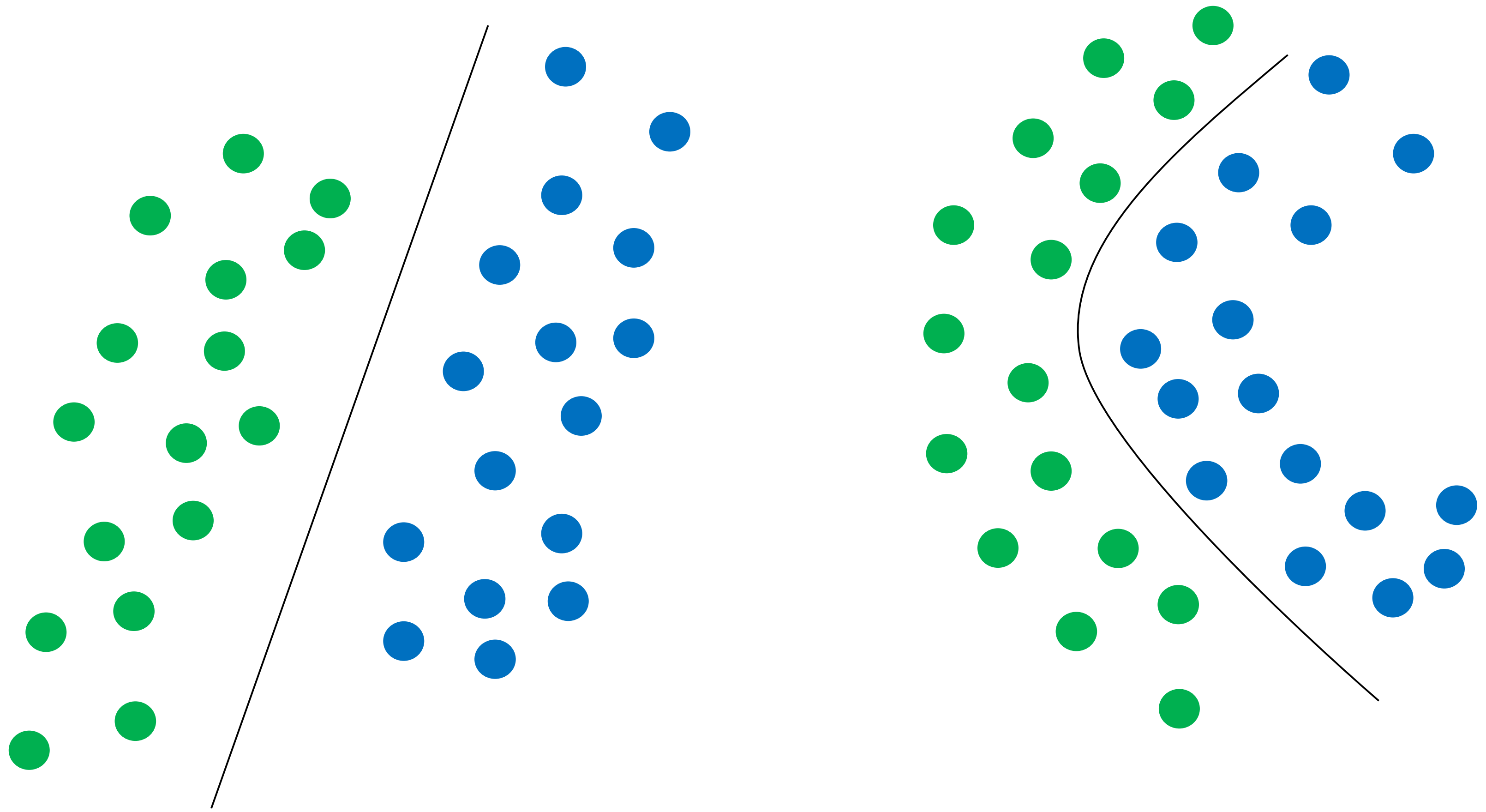




Linear vs Nonlinearly Separable

Linearly separable

Nonlinearly separable



Mapping Functions

Generalized Discriminant Functions

- **Quadratic Discriminant:**

$$g(\mathbf{x}) = \mathbf{x}^t \mathbf{W} \mathbf{x} + \mathbf{w}^t \mathbf{x} + w_0$$

- where
  - **W** is a matrix,
  - **w** is a vector, and
  - w_0 is a threshold weight
- This is the general form of the Bayesian classifier with normal distributions (Ch. 2)
- It can also be generalized to …

  a *polynomial* discriminant function
- Note: for now, we will classify a vector **x** instead of **y**

Polynomial Discriminant Functions

- A *linear* discriminant function

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

- can be written as:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

- A *quadratic* discriminant function

$$g(\mathbf{x}) = \mathbf{x}^t \mathbf{W} \mathbf{x} + \mathbf{w}^t \mathbf{x} + w_0$$

- can be written as:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$$

Polynomial Discriminant Functions

- We can continue adding more terms like this:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d}\sum_{j=1}^{d} w_{ij} x_i x_j + \sum_{i=1}^{d}\sum_{j=1}^{d}\sum_{k=1}^{d} w_{ijk} x_i x_j x_k$$

- and so on…

  yielding *polynomial discriminant functions*.

- Clearly, a k^{th} order polynomial requires $O(d^k)$ terms !!
- But…
  - do we need all these terms?
  - Probably not…

Phi Functions

- Indeed, we can use "truncated" series expansions of some arbitrary function $g(\mathbf{x})$…

  to get the *generalized linear discriminant function*:

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i \phi_i(\mathbf{x})$$

  or written in another way:

$$g(\mathbf{x}) = \mathbf{a}^t \mathbf{y}$$

- The *weight vector* $\mathbf{a}$ is a $\hat{d}$-dimensional vector, and

  the idea is that $\hat{d} >>> d$
- We have now $\hat{d}$ functions called the "phi functions" $\boldsymbol{\phi}$(.)

  and are *arbitrary* functions of $\mathbf{x}$

Mapping + SVM

- Such functions are *not linear* on **x**,

  but are *linear* on **y**.
- So, then the idea of finding a linear function in a "higher-dimensional" space, say the **y**-space (feature space).
- The linear function defines a hyperplane in the **y**-space,
  - which passes through the origin of the system
- Also called *homogeneous* linear discriminant function
- Thus, two problems:
  - Find a mapping from **x** to **y**
  - Find a good linear discriminant function in the **y**-space

    (we already know how to get this one, but will see another way)
  - Now, once we map to the new space, we classify **y**

Example 1

- Consider this quadratic discriminant function on *x*

$$g(x) = a_0 + a_1 x + a_2 x^2$$

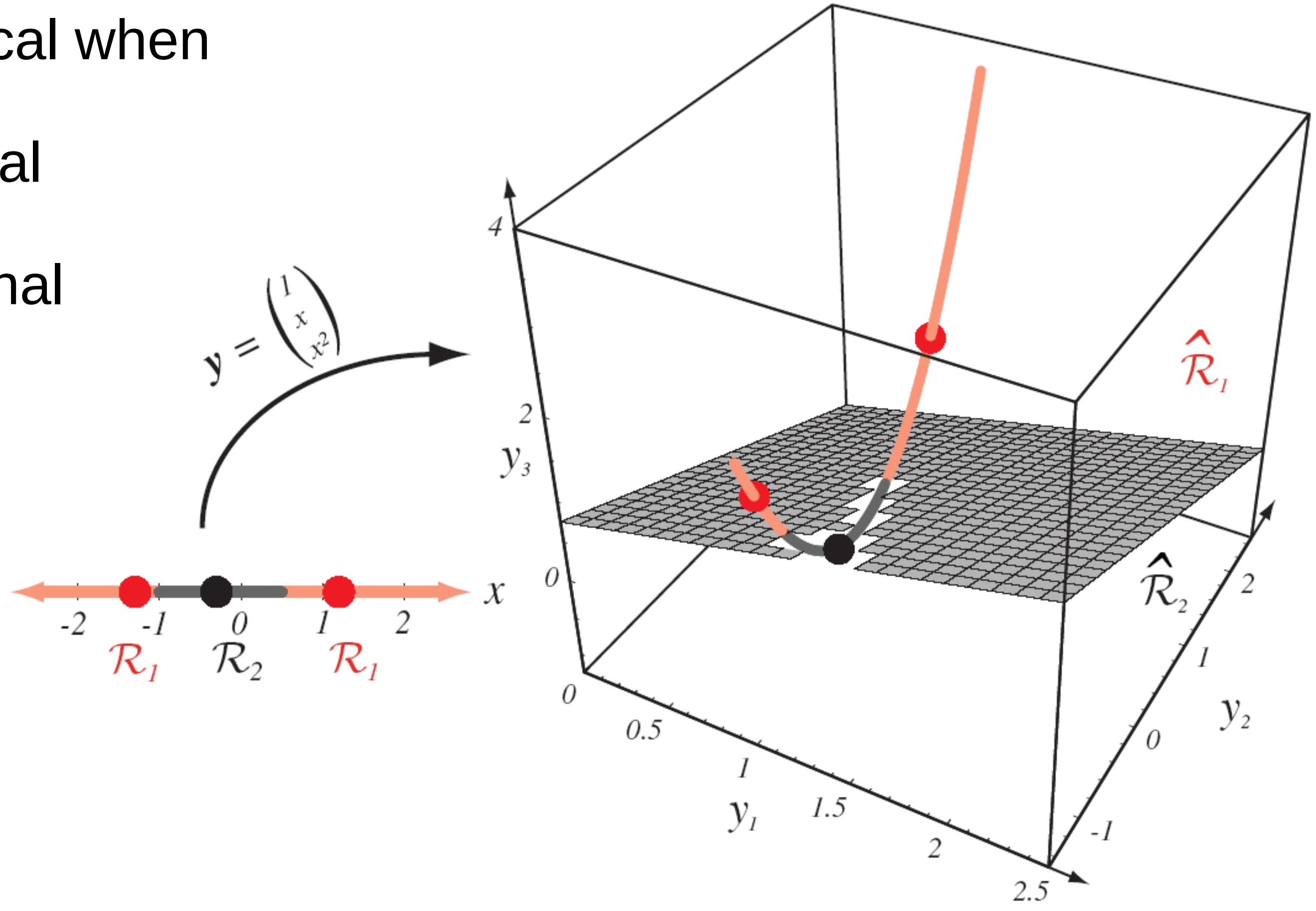
… it is a *one-variable* function (domain is in 1D space)

- Now, lets apply a mapping from *x* to **y** from the 1D space to the 3D space
- Resulting in a vector **y** as follows (example):

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

Graphically

- The data in the 3D space remains 1D, and lie in a *parabola*
- if x follows a distn $px(x)$, assume defined in $(-\infty,\infty)$,
- Then $\mathbf{y}$ follows another distn. $p\mathbf{y}(\mathbf{y})$,
  - being 0 in every $\mathbf{y} \notin$ the parabola
- This case is typical when mapping from a *lower*-dimensional space to a *higher*-dimensional space.



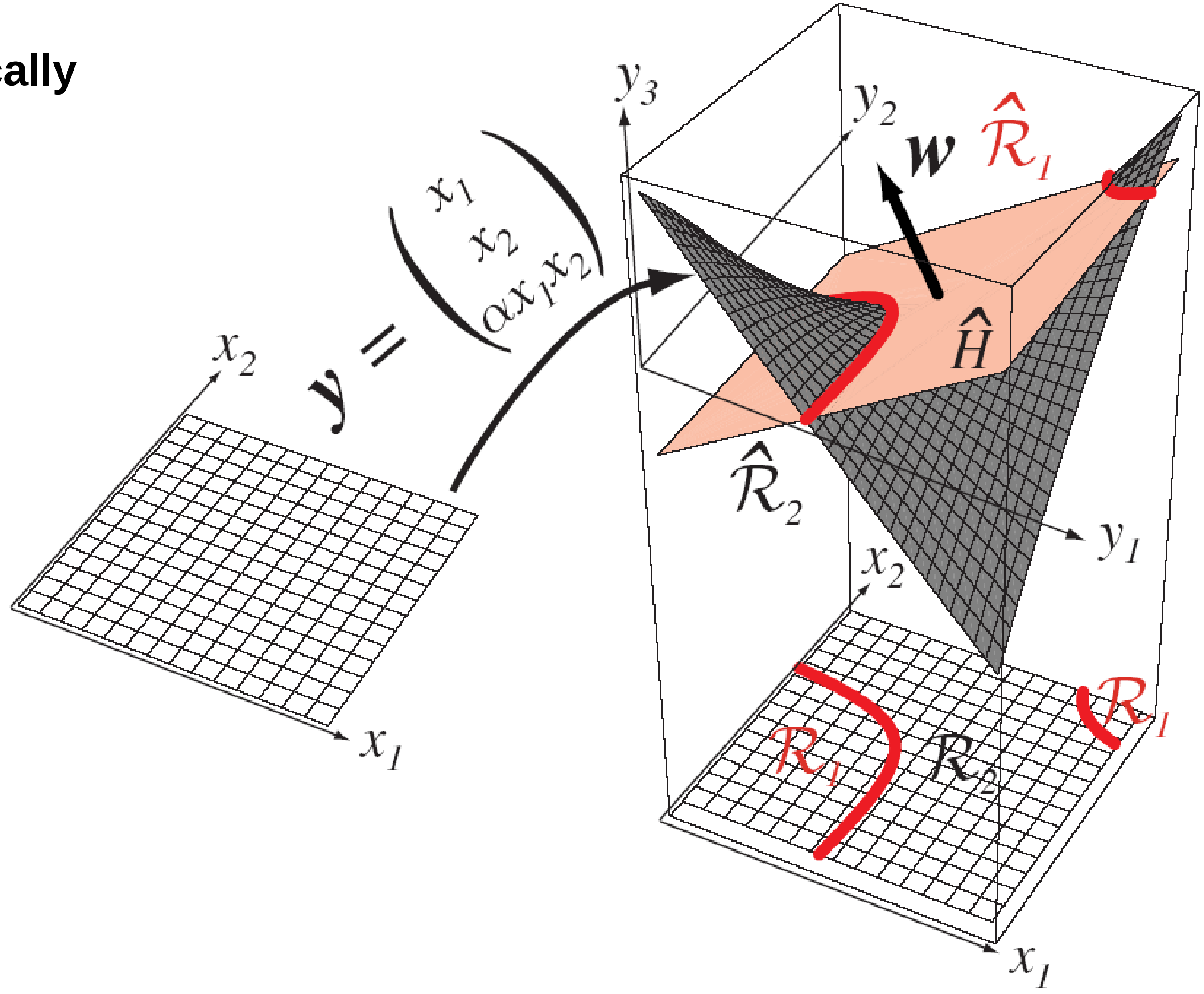
Example 2

- Let **x** = [x_1, x_2]^t in 2D space mapped to **y** in the 3D space:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \alpha x_1 x_2 \end{bmatrix}$$

- This transformation places all data in an *arbitrary surface* in 3D (see next page)
- The classifier in 2D space is a hyperbola that divides the space into two regions.
- It is equivalent to a plane in the 3D space!

Graphically



Moral: Data **nonlinearly** separable in *some* space
may become **linearly** separable in *another* space!

Example 3

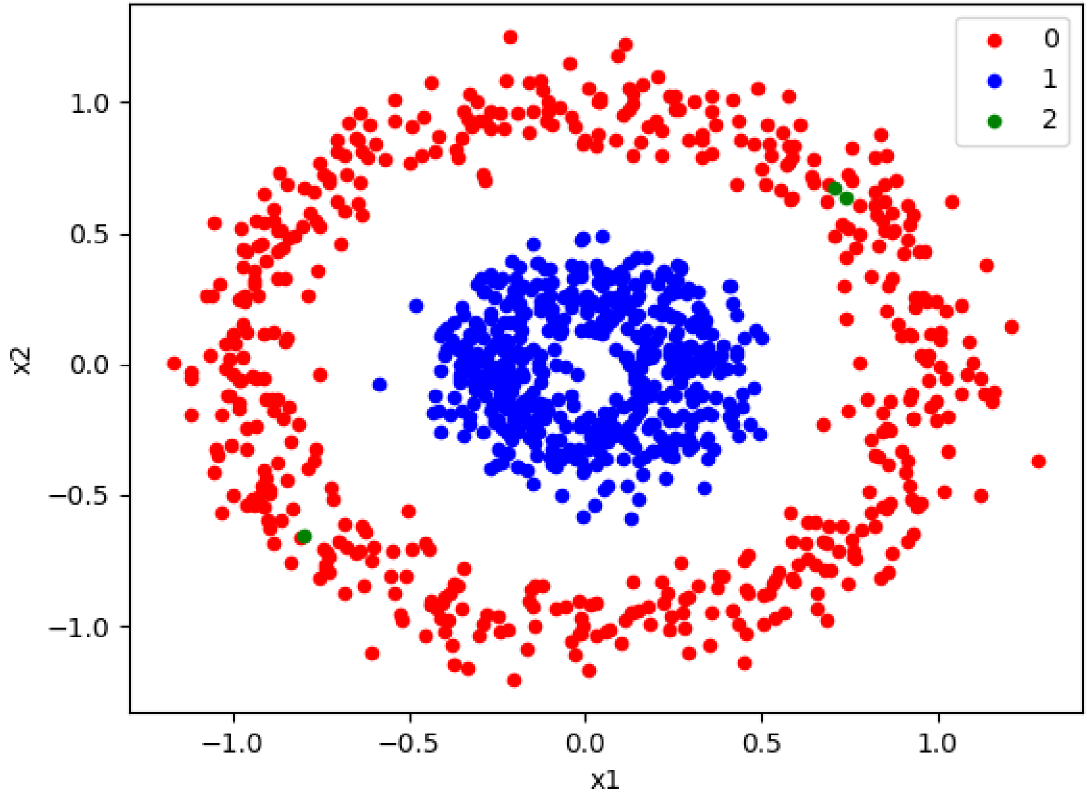
- Let **x** = $[x_1, x_2]^t$ in 2D space mapped to **y** in the 2D space:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_1^2 + x_2^2 \end{bmatrix}$$

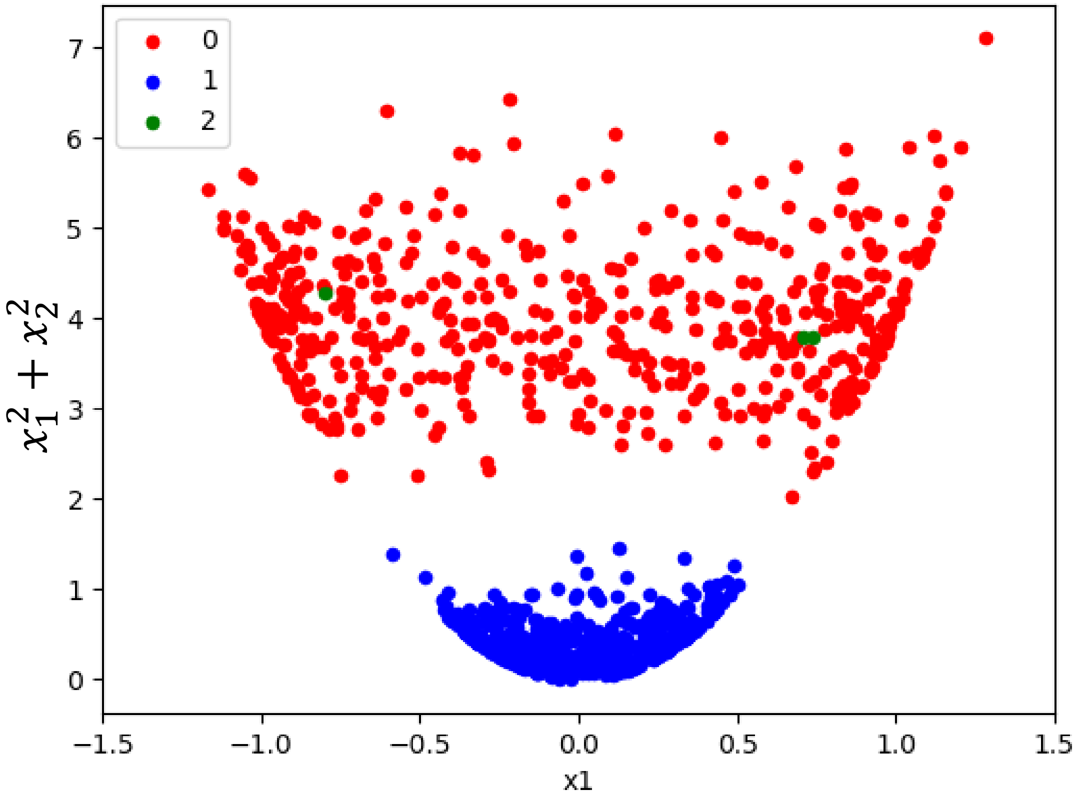
- This transformation places all data in an *arbitrary parabolic* form in 2D (see next page)
- The classifier in the original 2D space has a circular form (circle) to divide the space into two regions.
- It is equivalent to a line in the new 2D space!

Graphically

Original 2D space:

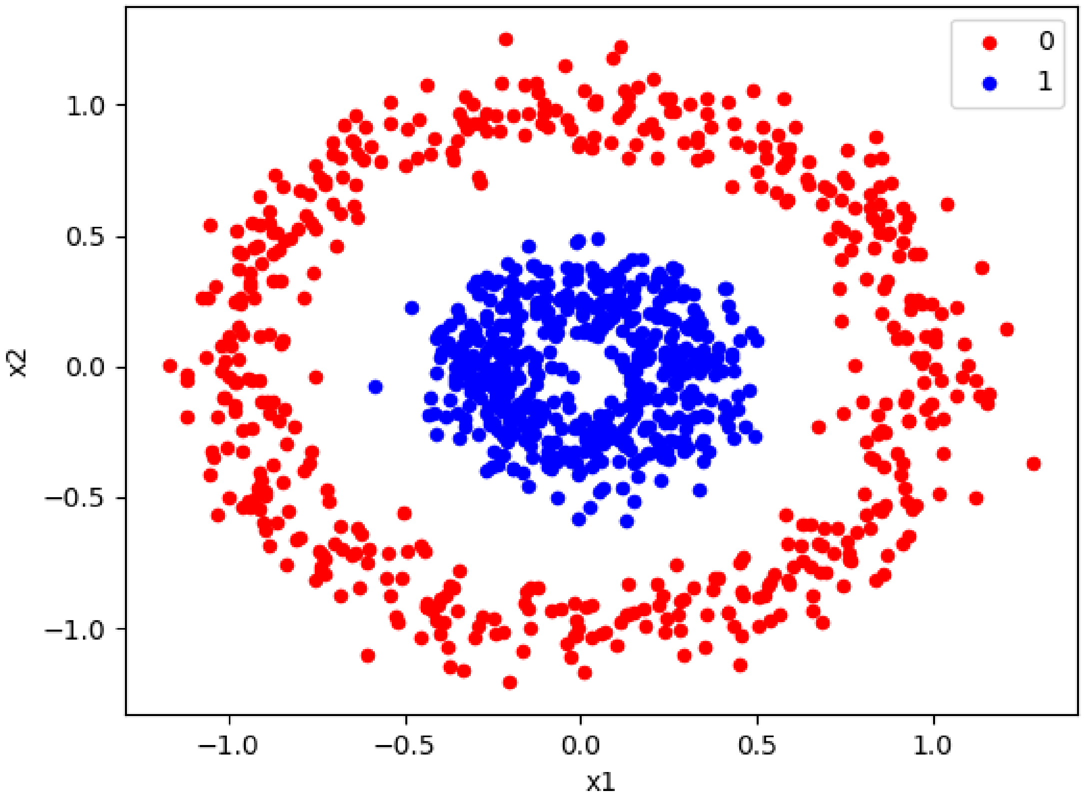


New 2D space:

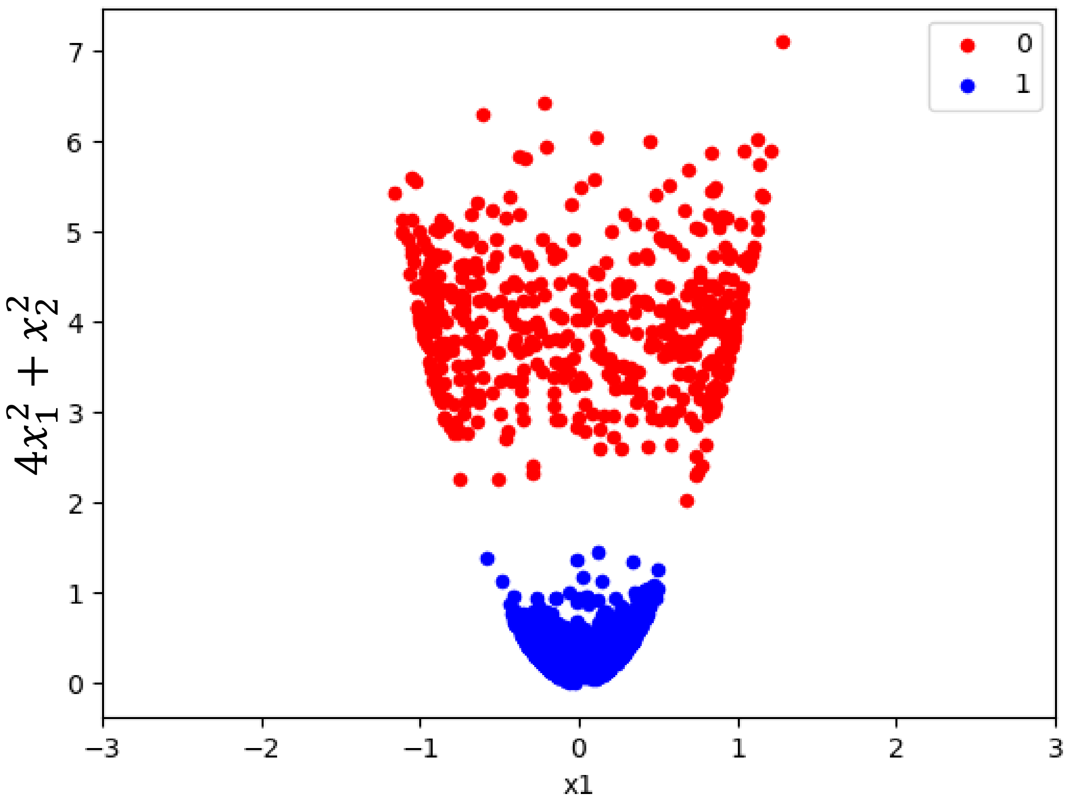


Graphically – cont'd

Original 2D space:



New 2D space:



Explicit Mapping

- Lets transform each vector $\mathbf{x}_k$ (a d-dim. vector)

  into a new one $\mathbf{y}_k$ in the $\hat{d}$-dim. space as:

  $$\mathbf{y}_k = \phi(\mathbf{x}_k)$$

  where, ideally, $\hat{d} \gg d$ and

  $\phi(\cdot)$ is an *arbitrary*, (typically) *nonlinear* function of $\mathbf{x}$

  called "phi function"

Note: It could be the case that ϕ is a linear function

- Then, derive the SVM classifier in the $\hat{d}$-dimensional space, and we are done
- However:
  - $\hat{d}$ is typically very large or it could be infinite!

Kernels

For the phi function:

y = $\boldsymbol{\phi}$(**x**), where $\boldsymbol{\phi}: \Re^d \rightarrow \Re^{\hat{d}}$

or $\mathbf{x} = [x_1, x_2, \ldots, x_d]^t \rightarrow \mathbf{y} = \boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_{\hat{d}}(\mathbf{x})]^t$

$\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_{\hat{d}}(\mathbf{x})$ are *arbitrary functions* of **x**

… and the corresponding kernel is:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}^t(\mathbf{x}_i)\, \boldsymbol{\phi}(\mathbf{x}_j) = \mathbf{y}_i^t\, \mathbf{y}_j$$

The kernel is a function of two vectors, and is defined as the inner product of these vectors

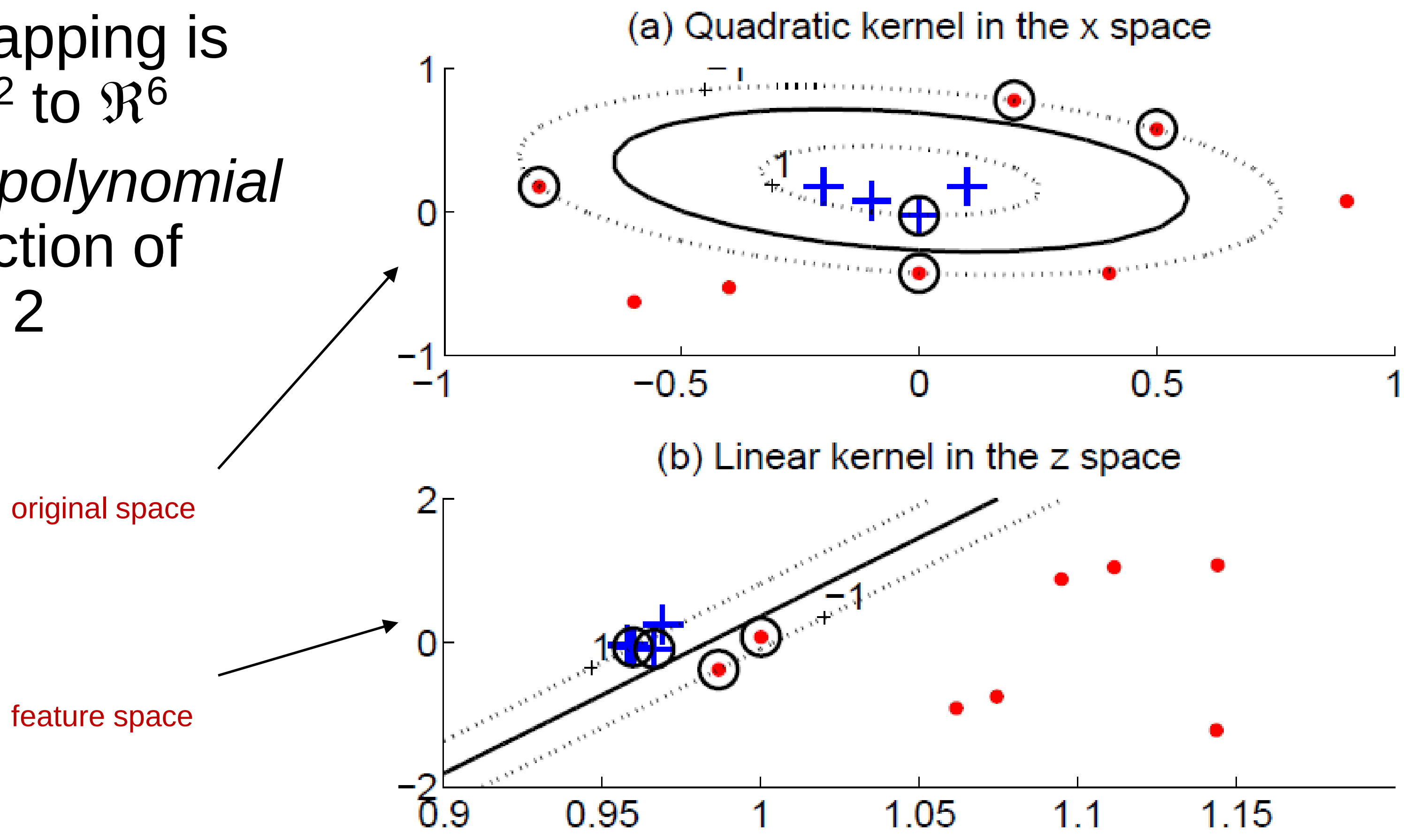
Notes:

- $\hat{d}$ can be even ∞
- Classification is nonlinear on $\Re^d$ but can linear on $\Re^{\hat{d}}$

Example 1

Polynomial phi function:

- Let $\mathbf{x}=[x_1,x_2]^t \rightarrow \mathbf{y}=\phi(\mathbf{x})=[x_1^2,x_2^2,\sqrt{2}x_1x_2,\sqrt{2}x_1,\sqrt{2}x_2,1]^t$ be a mapping
- This mapping is from $\Re^2$ to $\Re^6$
- Called *polynomial* phi function of degree 2



Kernels - examples

- <u>Linear (dot):</u>
  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^t \mathbf{x}_j$
  (linear phi function)

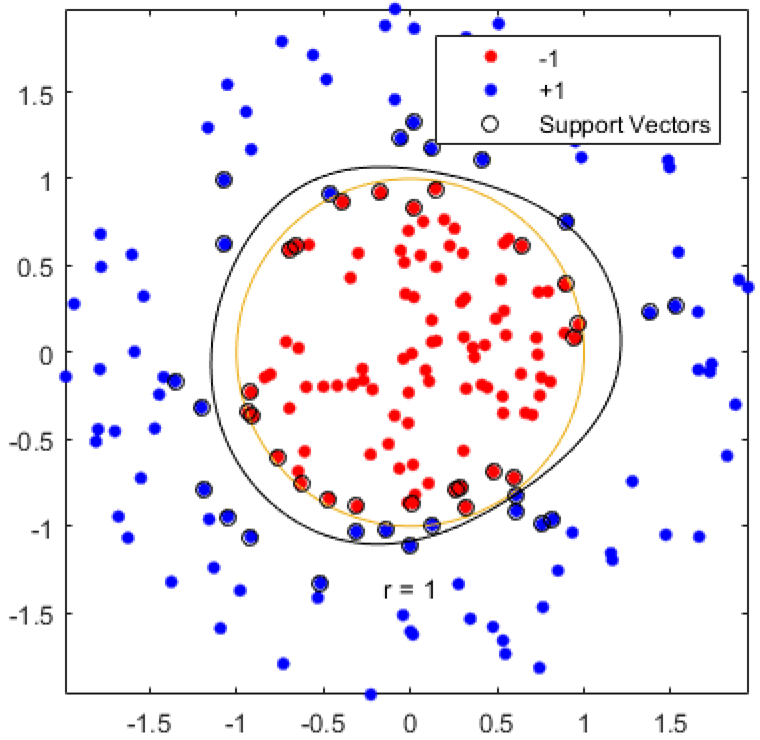
- <u>Polynomial:</u>
  $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^t \mathbf{x}_j + 1)^q$, $q > 0$

  If $q = 2$: $(\mathbf{x}_i^t \mathbf{x}_j + 1)^2 = \mathbf{y}_i^t \mathbf{y}_j$
  where, for $d=2$: $\mathbf{y} = \boldsymbol{\phi}(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]^t$

  (polynomial phi function)

Radial basis function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}{2\sigma^2} \right)$$



- Also known as *Gaussian* kernel
- σ controls the shape of the separating function
- σ is also known as “gamma” or γ

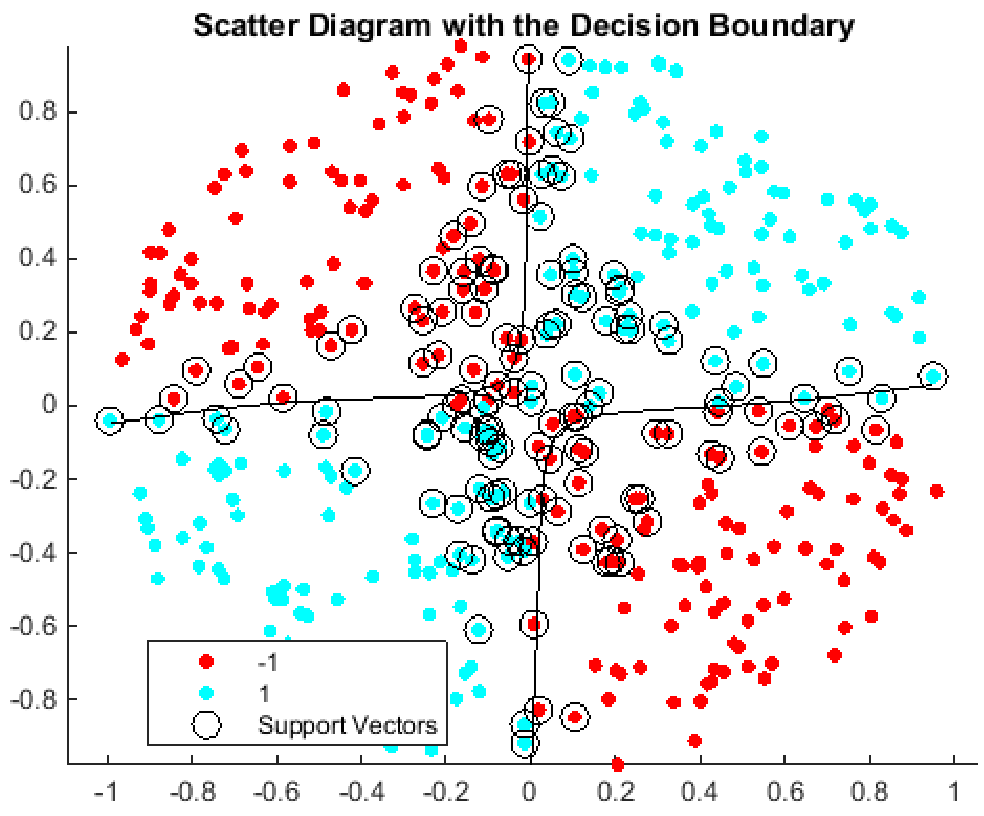
Hyperbolic tangent:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\beta\, \mathbf{x}_i^t \mathbf{x}_j + \gamma\right)$$

for values of β and γ that satisfy Mercer´s conditions

(e.g. $\beta = 2$ and $\gamma = 1$)

Aka *sigmoid* kernel or *multilayer perceptron* (neural network)



Other kernels:

- Anova
- Fourier series
- Three-layer (or multi-layer) neural network
- Spline
- B-spline
- Additive
- Tensor product
- Graph kernels
- and more …

Mercer's Conditions

- Let $\mathbf{x} \in \Re^d$ and $\boldsymbol{\phi} : \Re^d \to \mathcal{H}$
  $\boldsymbol{\phi}$ has $\hat{d}$ components, $\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_{\hat{d}}(\mathbf{x})$
  where *H* is a Hilbert space
  (a complete linear space + inner product)
- $K(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}^t(\mathbf{x})\, \boldsymbol{\phi}(\mathbf{z})$ is a symmetric function
  that satisfies:
  $$\int K(\mathbf{x},\mathbf{z}) g(\mathbf{x}) g(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq \mathbf{0}$$
  for any $g(\mathbf{x})$, $\mathbf{x} \in \Re^d$ such that
  $$\int g^2(\mathbf{x}) d\mathbf{x} < \infty$$
- $K(.,.)$ is a positive semidefinite kernel
- For a finite set of vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$,
  K is a positive semidefinite matrix

The Kernel Trick in SVM

- An advantage of using kernels:
  - The solution can be found without an explicit mapping to the higher dimensional space
- Recall: $$L(\boldsymbol{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j=1}^{n} \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k \qquad (4)$$
- L function in terms of kernels:
$$L(\boldsymbol{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j=1}^{n} \alpha_k \alpha_j z_k z_j K(\mathbf{x}_j, \mathbf{x}_k) \qquad (5)$$
- subject to the following constraints:
$$\sum_{k=1}^{n} z_k \alpha_k = 0, \quad \text{and } \alpha_k \geq 0$$
- Obtain values of α_k
- n_s (support) vectors are those for which $\alpha_k \neq 0$

Classification

- With kernel:

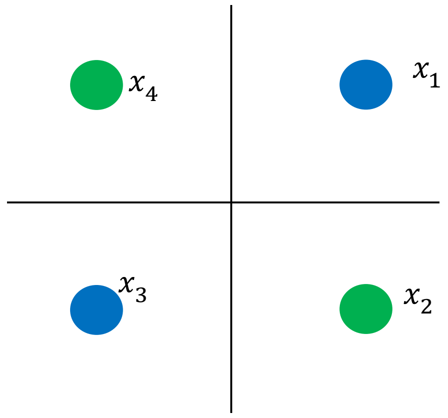
$g(\mathbf{x}) = \sum_{i=1}^{n_s} \alpha_i z_i K(\mathbf{x}_i, \mathbf{x})$ projects $\mathbf{x}$ onto the 1D space

$$\begin{aligned} g(\mathbf{x}) > 0 \quad &\text{decide} \quad \mathbf{x} \in \omega_1 \\ \text{otherwise } &\text{decide} \quad \mathbf{x} \in \omega_2 \end{aligned}$$

- With explicit mapping:

$$\begin{aligned} \mathbf{a}^t \mathbf{y} = \mathbf{a}^t \boldsymbol{\phi}(\mathbf{x}) > 0 \ &\text{decide} \quad \mathbf{x} \in \omega_1 \\ \text{otherwise } &\text{decide} \quad \mathbf{x} \in \omega_2 \end{aligned}$$

Example



- The XOR classification problem
- 4 samples:

  $\mathbf{x}_1 = [1,1]^t$ and $\mathbf{x}_3 = [-1,-1]^t \in \omega_1$

  $\mathbf{x}_2 = [1,-1]^t$ and $\mathbf{x}_4 = [-1,1]^t \in \omega_2$

- Lets use a *degree*-2 polynomial phi-function and explicit mapping
- So, mapping from 2D to 6D:

$$\mathbf{y} = \boldsymbol{\phi}(\mathbf{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]^t$$

- Eq. (4) results in:

$$L(\boldsymbol{\alpha}) = \sum_{k=1}^{4} \alpha_k - \frac{1}{2} \sum_{k,j=1}^{4} \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k \qquad (6)$$

subject to:

$$\sum_{k=1}^{4} \alpha_k z_k = 0, \quad \text{and } \alpha_k \geq 0$$

Expanding:

$$L(\boldsymbol{\alpha}) = \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}(9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4$$

$$+ 9\alpha_2^2 + 2\alpha_2\alpha_3 - 2\alpha_1\alpha_3 + 2\alpha_2\alpha_4 + 9\alpha_3^2 - 2\alpha_3\alpha_4 + 9\alpha_4^2)$$

subject to:

$$\alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 = 0, \ \alpha_k \geq 0, \ k = 1,2,3,4$$

Solution:

- Optimal values for the undetermined multipliers:

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{8} \neq 0$$

- Optimal vector **a** given by:

$$\mathbf{a} = \sum_{k=1}^{4} \alpha_k z_k \boldsymbol{\phi}(\mathbf{x}_k) = [0,0,0,\frac{1}{\sqrt{2}},0,0]$$

- In this example, *all* 4 training samples are support vectors

- So, the *optimal* hyperplane is given by:

$$\mathbf{a}^t \boldsymbol{\phi}(\mathbf{x}) = 0, \quad \text{or}$$

$$[0,0,0,\frac{1}{\sqrt{2}},0,0]\left[1,\sqrt{2}x_1,\sqrt{2}x_2,\sqrt{2}x_1x_2,x_1^2,x_2^2\right]^t = 0$$

- Solving for **y**, we have:

$$x_1 x_2 = 0$$

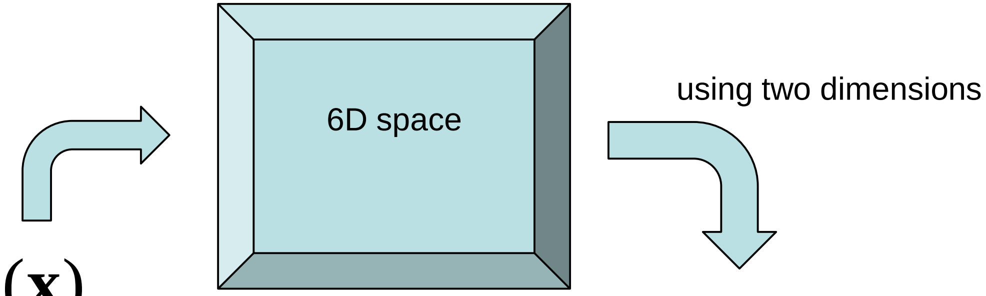
- The margin *b* is given by:

$$b = 1/\|\mathbf{a}\| = \sqrt{2}$$

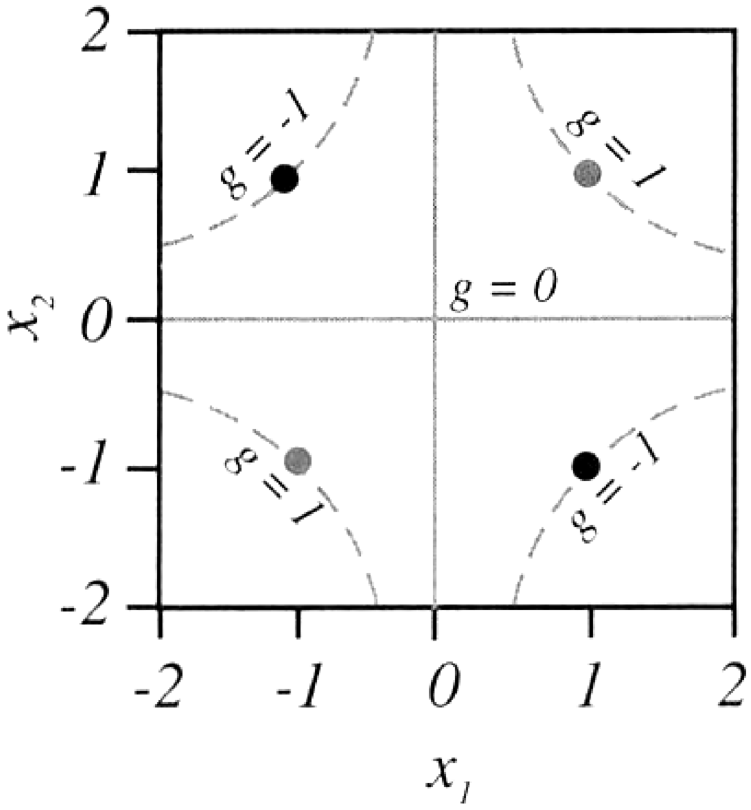
Notes:

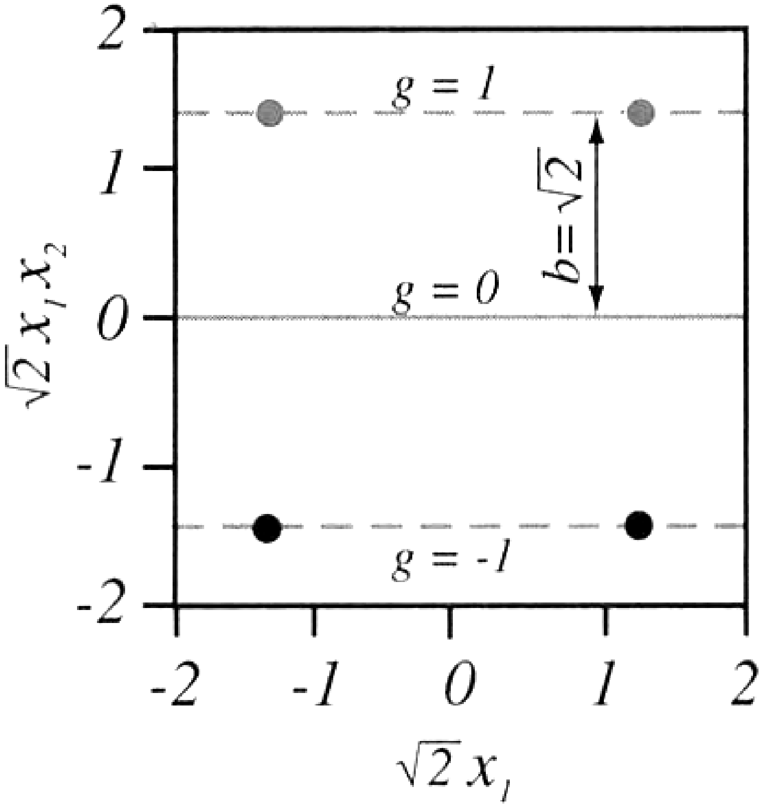
- A 2D representation/classifier is good enough
- Mapping to 6D not required at classification time
- Even a 1D classifier is enough!

The XOR problem graphically



$$\mathbf{y} = \phi(\mathbf{x})$$





Lets classify:

$$\mathbf{x} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

$$\boldsymbol{\phi}(\mathbf{x}) = \mathbf{y} = [1, \sqrt{2}(0.5), \sqrt{2}(0.5), \sqrt{2}(0.5)(0.5), 0.25, 0.25]^t$$

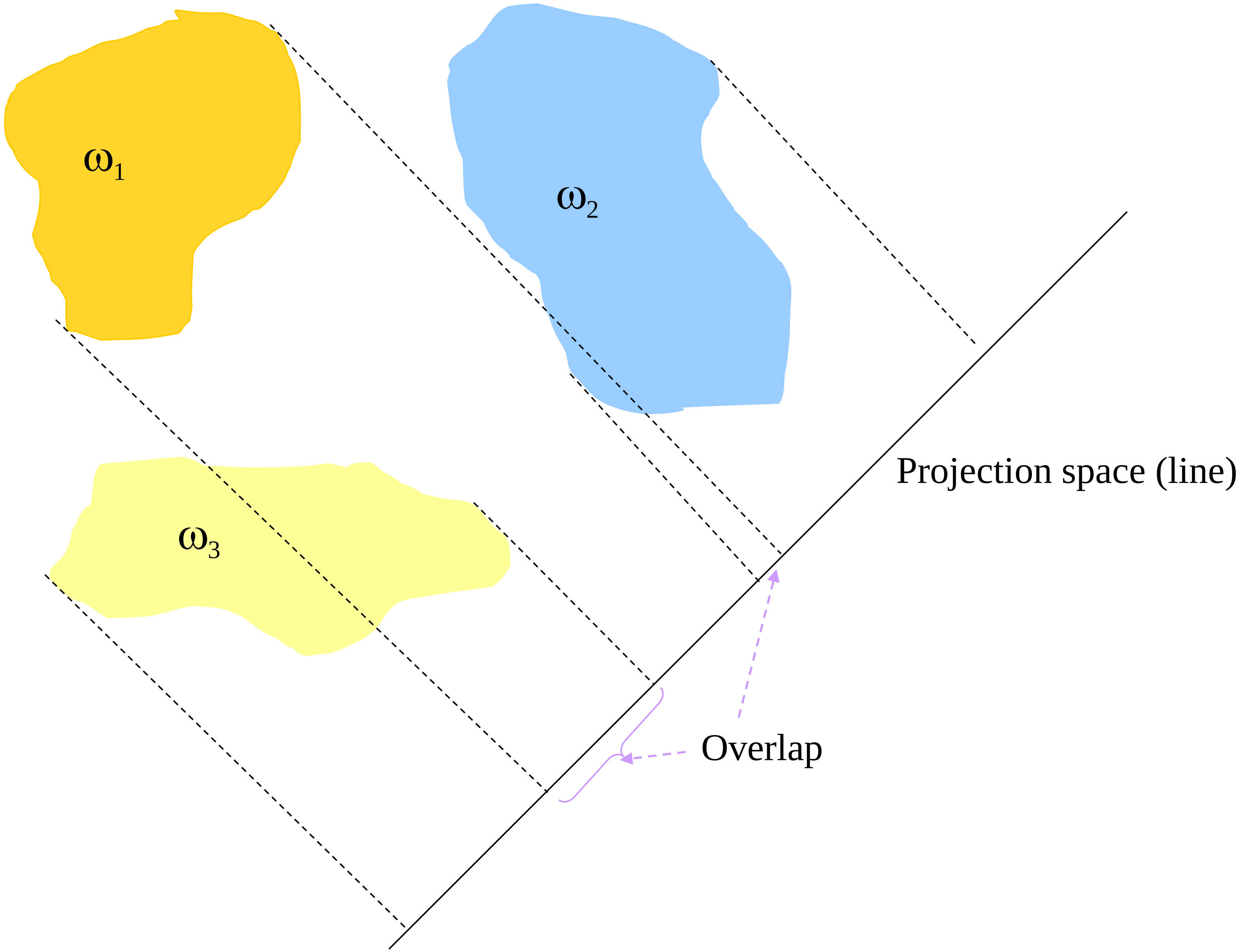
$$\mathbf{a}^t\mathbf{y} = [0,0,0,\frac{1}{\sqrt{2}},0,0]\begin{bmatrix} 1 \\ \sqrt{2}(0.5) \\ \sqrt{2}(0.5) \\ \sqrt{2}(0.5)(0.5) \\ 0.25 \\ 0.25 \end{bmatrix} = \frac{1}{\sqrt{2}}\sqrt{2}(0.5)(0.5)$$

$$\Rightarrow \mathbf{a}^t\mathbf{y} = 0.25 > 0 \Rightarrow \text{ decide } \mathbf{x} \in \omega_1$$

The Multi-Class Case

- Assume we have *c* classes: $\omega_1, \omega_2, \ldots, \omega_c$
- Various approaches:
  - All at once
  - One against all
  - One against one
  - Linear machine: Divide the space into *c* regions

All at once



One against all

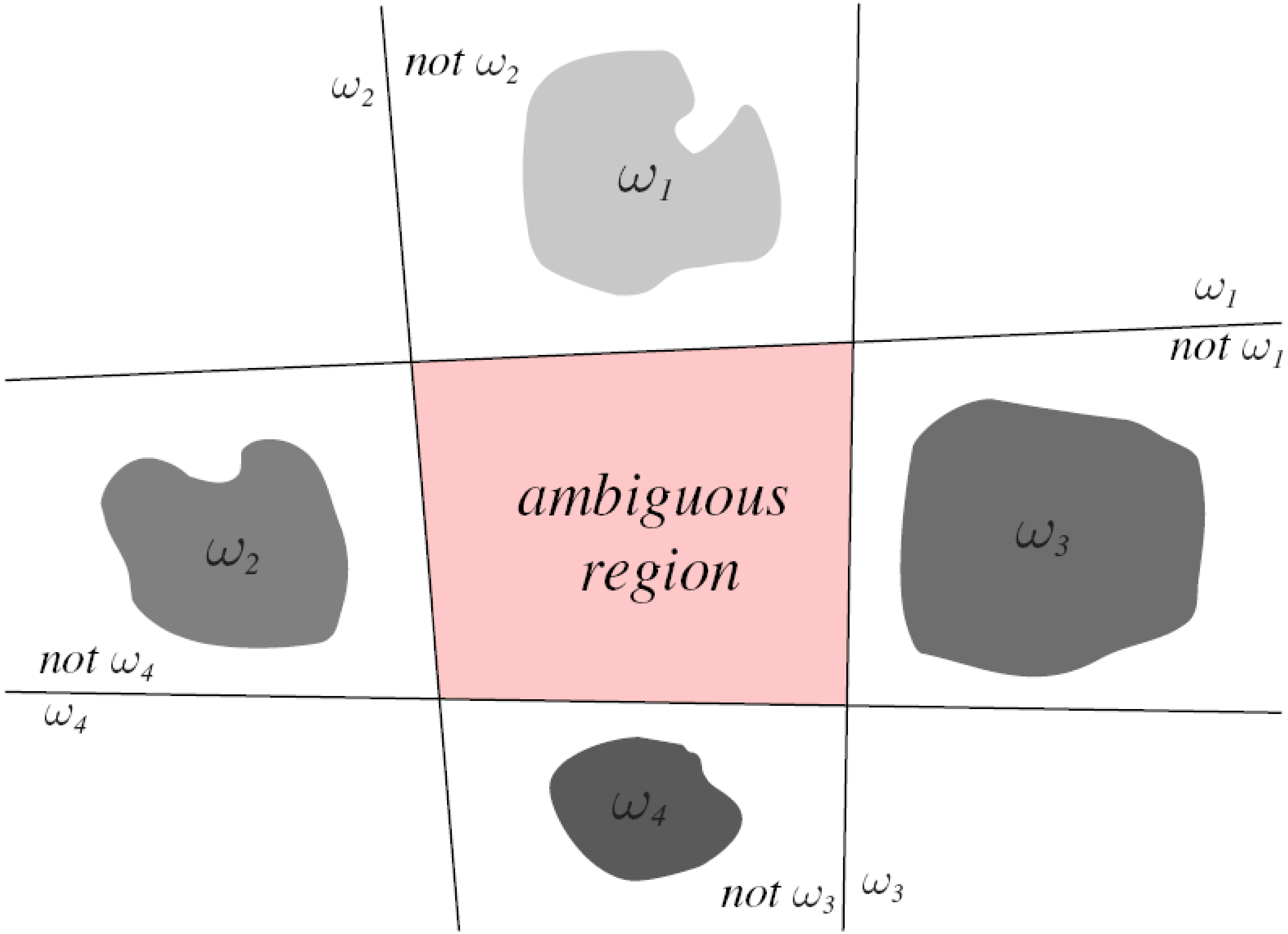
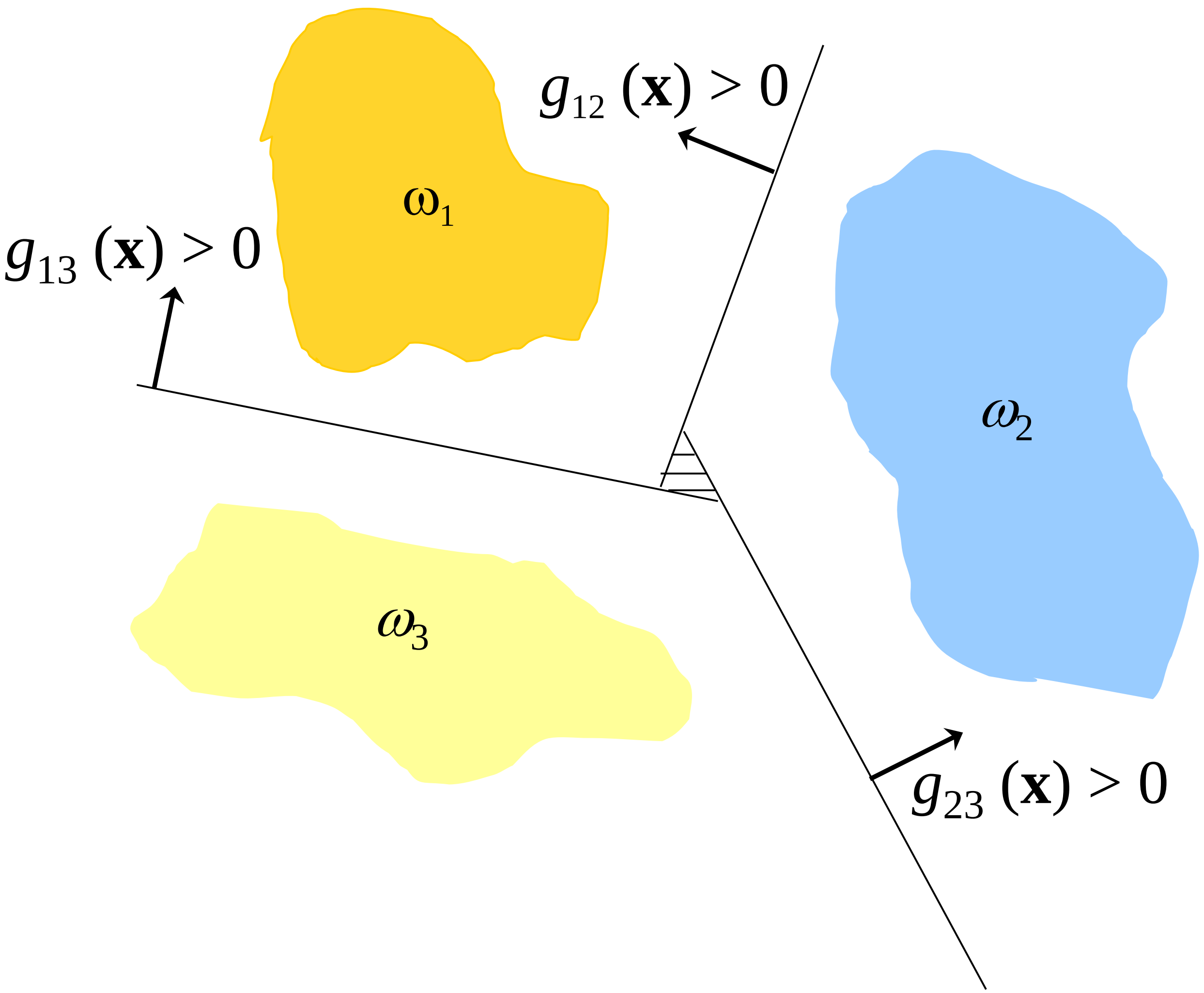


Image from Duda et al.

One against one



Tools for SVM

- **Resources:**
  - Kernel machines: http://www.kernel-machines.org/
  - SVMs: www.support-vector-machines.org
- **Tools:**
  - LibSVM: Interface with various languages including Matlab/Octave (http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
  - WEKA (by installing LibSVM or others)
  - Scikit: SVM/SVC (http://scikit-learn.org/stable/modules/svm.html)
  - SVM Light: http://svmlight.joachims.org/
  - OSU SVM: http://sourceforge.net/projects/svm/)
- **In Matlab:**
  - Bioinformatics toolbox http://www.mathworks.com/help/toolbox/bioinfo/ref/svmtrain.html
  - Syntax:
    - SVMStruct = svmtrain(Training,Group)
    - SVMStruct = svmtrain(Training,Group,Name,Value)

SVM in Scikit

- Available as part of the standard APIs
- Support two-class, multi-class and one-class classification
- Allows for different kernels: linear, polynomial, RBF, sigmoid, and custom kernels
- Multi-class implements the one-against-one approach
- Can deal with unbalanced classification problems via weight parameters
- Supports regression and density estimation
- Documentation:
  - http://scikit-learn.org/stable/modules/svm.html

Linear Classifiers in Scikit

- Linear Discriminant Analysis (LDA)
  - https://scikit-learn.org/0.16/modules/generated/sklearn.lda.LDA.html
- Linear and Quadratic classifiers in Scikit:
  - https://scikit-learn.org/stable/modules/lda_qda.html
- Mean square error (MSE) for regression
  - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
- Perceptron
  - https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html
- Multi-layer Perceptron (no GPU support)
  - https://scikit-learn.org/stable/modules/neural_networks_supervised.html

References

1. E. Chong et al., An Introduction to Optimization. 2nd Edition, Wiley, 2001
2. S. Abe, Support Vector Machines for Pattern Classification, 2nd Edition, Springer, 2010
3. R. Duda et al, Pattern Classification, 2nd Edition, Wiley, 2000
4. M. Kubat. An Introduction to Machine Learning, Second Ed., Springer, 2017
5. C. Aggarwal. Neural Networks and Deep Learning. Springer, 2018