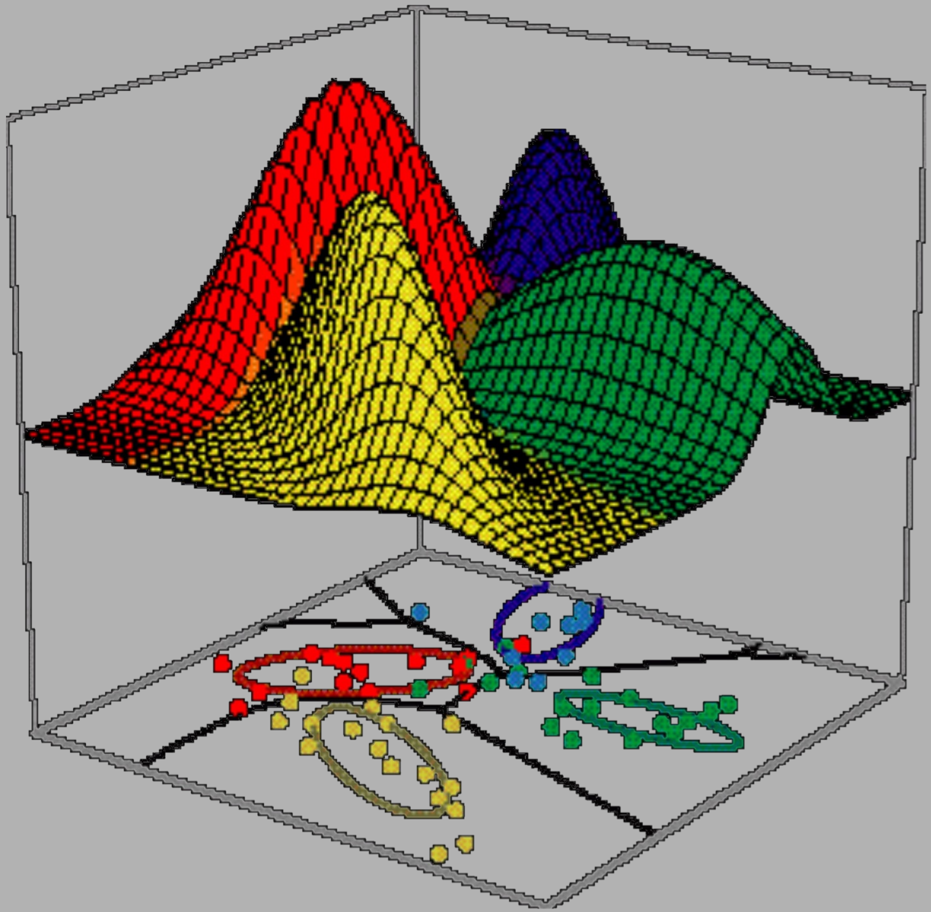


Pattern Recognition (Αναγνώριση Προτύπων)

Linear Discriminant Functions

(Γραμμικές Συναρτήσεις Διάκρισης)

Panos Trahanias

UNIVERSITY OF CRETE
DEPARTMENT of COMPUTER SCIENCE

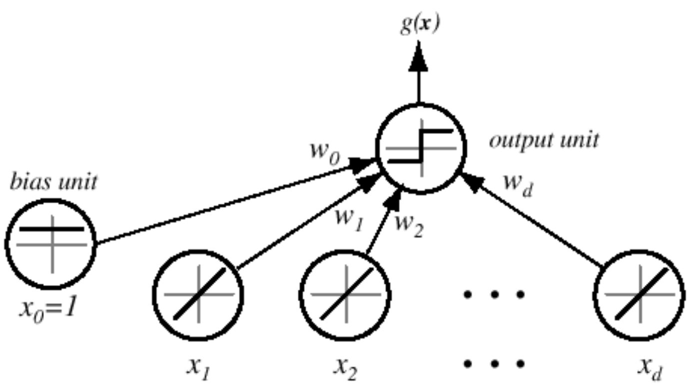
Linear Discriminant Functions

- Goal: Formulate linear –with respect to feature vector ***x***- discriminant functions that define hyperplanes as decision surfaces.
- Why? Simple form, straightforward implementation, optimal for Gaussian pdfs.
- How: Formulate the parameter (weights) estimation problem as an optimization of a criterion (cost) function.
- What is the criterion function? A real valued function of the weights (parameters) that is amenable to minimization, e.g. probability of mis-classification during training.
- How hard is to accomplish? In the general case, it is hard to formulate a linear classifier that minimizes error.
- So what? Employ alternative criteria (simple functions of the weights) and iterative optimization methods (e.g. gradient descent).

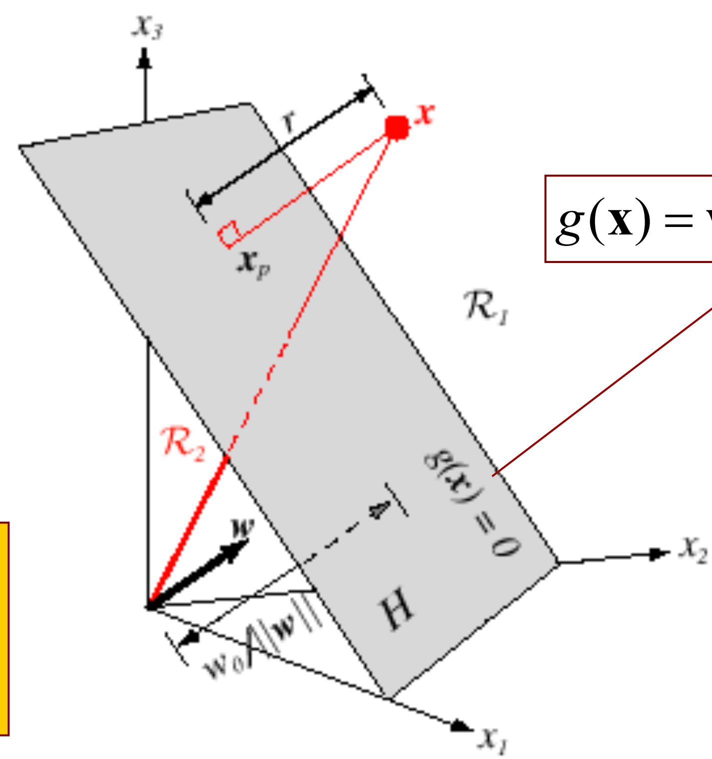
Discriminant Functions and Decision Surfaces

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

Weight vector — Bias, threshold weight



Weight vector, **w**, defines the orientation of the decision hyperplane and the threshold weight, w_0, defines its relative position with respect to the origin.



$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$$

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}, \text{ with } r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

Discriminant Functions and Decision Surfaces

If $\mathbf{x}_1$ and $\mathbf{x}_2$ are both on the decision surface, then

$$\mathbf{w}^t\mathbf{x}_1 + w_0 = \mathbf{w}^t\mathbf{x}_2 + w_0$$

$$\mathbf{w}^t(\mathbf{x}_1 - \mathbf{x}_2) = 0,$$

w normal to any vector on the hyperplane, hence normal to the hyperplane

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

$$w^t x + w_0 = w^t\left(x_p + r\frac{w}{\|w\|}\right) + w_0 = w^t x_p + r\frac{w^t w}{\|w\|} + w_0 = r\|w\|$$

$$\Longrightarrow \quad r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}.$$

The multi-class Case

Linear Machine: $\boldsymbol{x}$ in ω_i if $g_i(\boldsymbol{x}) > g_j(\boldsymbol{x})$

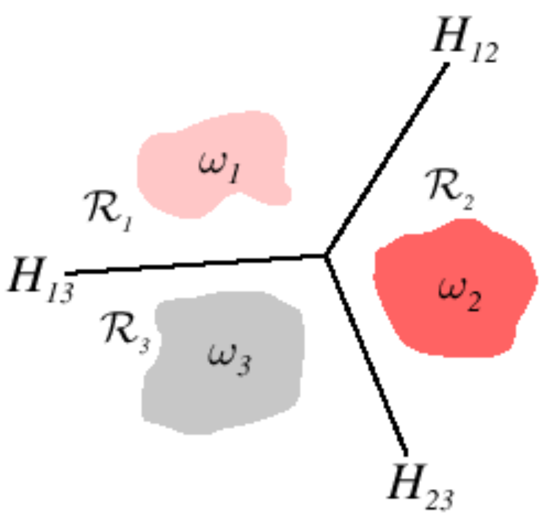
Decision boundaries: H_{ij}: $g_i(\boldsymbol{x}) = g_j(\boldsymbol{x}) \rightarrow (\boldsymbol{w}_i - \boldsymbol{w}_j)^t \boldsymbol{x} + (w_{i0} - w_{j0}) = 0$

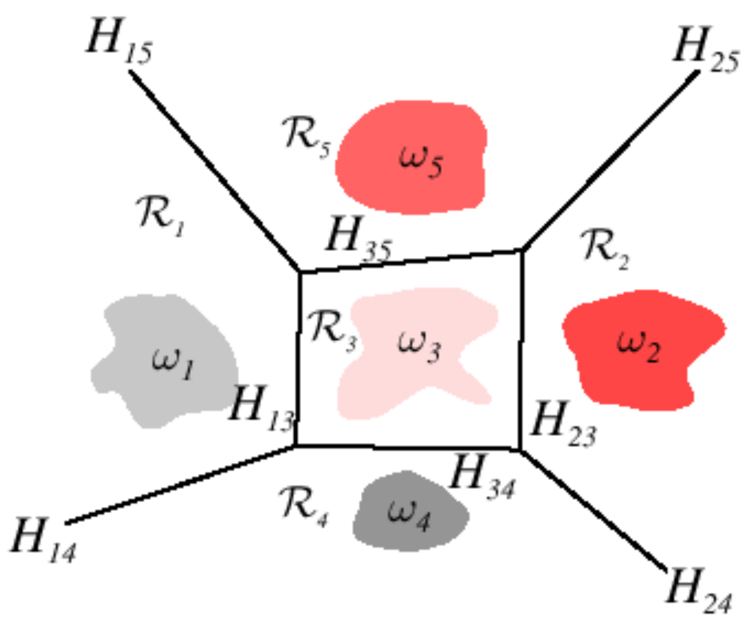
- part of hyperplane normal to vector $\boldsymbol{w}_i - \boldsymbol{w}_j$.

Distance of $\boldsymbol{x}$ from H_{ij}: $(g_i(\boldsymbol{x}) - g_j(\boldsymbol{x})) / ||\boldsymbol{w}_i - \boldsymbol{w}_j||$

- differences in weight vectors play a role.

Convex decision regions.





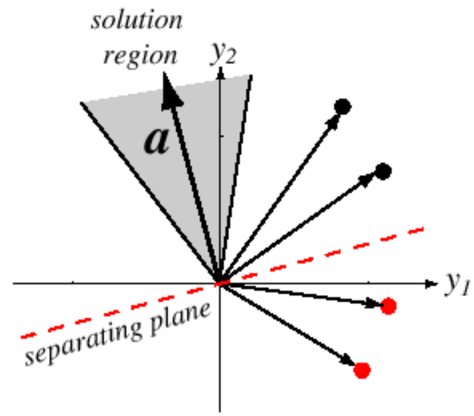
Linearly Separable Classes: Vectors and Solution Regions

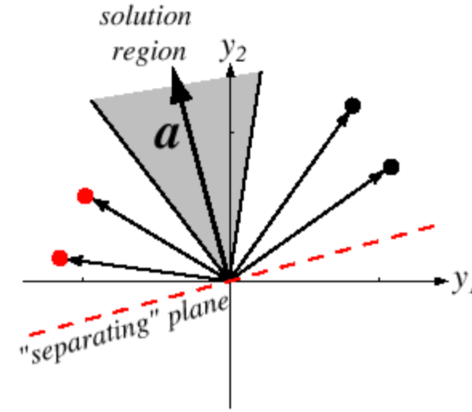
Augmented Weight and Feature Vectors:

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} x_0 \\ \mathbf{x} \end{bmatrix}$$

$$\mathrm{H}: \ g(\mathbf{y}) = \mathbf{a}^T \mathbf{y} = 0$$

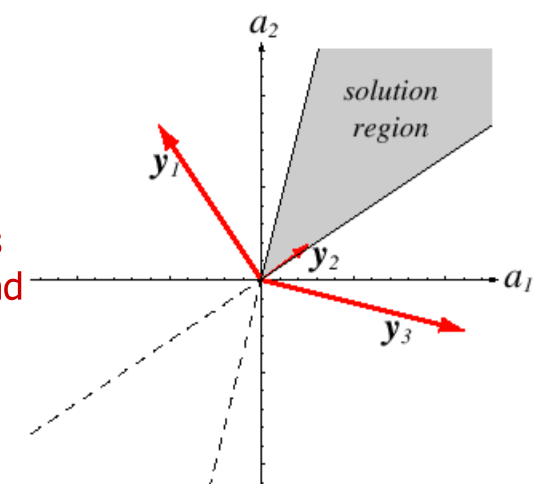
Distance of $\mathbf{y}$ from H: $|\mathbf{a}^T \mathbf{y}| / \|\mathbf{a}\|$



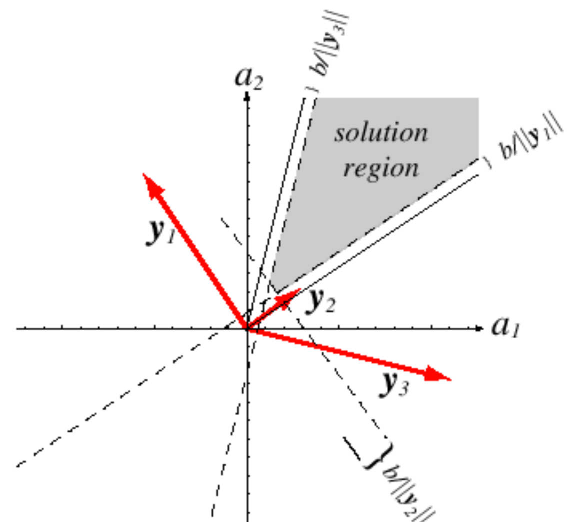


Normalization: replace all training samples from class ω_2 with their negatives, and find the separating vectors that satisfy the relation:

$$\mathbf{a}^T \mathbf{y}_i > 0 \quad \forall\, i$$



$$\mathbf{a}^T \mathbf{y}_i > 0 \quad \forall\, i$$



$$\mathbf{a}^T \mathbf{y}_i \geq b > 0 \quad \forall\, i$$

Optimization Procedures

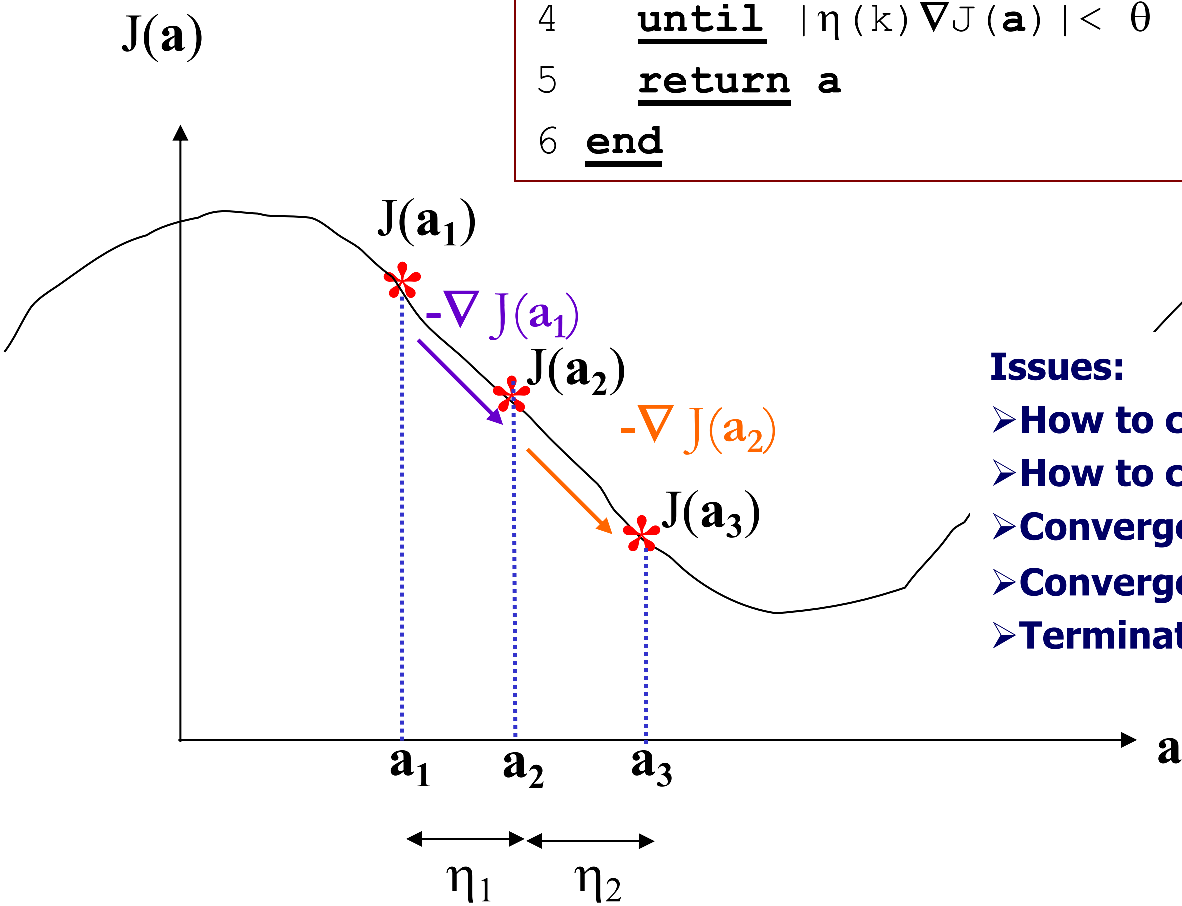
- Task: Find $\boldsymbol{a}$ that satisfies the set of linear inequalities $\boldsymbol{a}^t\boldsymbol{y}_i>0$ for all $i=1,\ldots,n$.
- How an appropriate solution is found?
  - Define a criterion function, $J(\boldsymbol{a})$, and minimize it to obtain $\boldsymbol{a}$ as a solution vector.
  - Accordingly, we transform the exhaustive search problem to that of minimizing a real-valued function.
- How is $J(\boldsymbol{a})$ minimized?
  - Select an initial point $\boldsymbol{a}_1$ and compute $J(\boldsymbol{a}_1)$.
  - Compute the derivative of $J(\boldsymbol{a}_1)$: $\nabla J(\boldsymbol{a}_1)$.
  - Find the next point $\boldsymbol{a}_2$ in the direction of steepest descent, $-\nabla J(\boldsymbol{a}_1)$, using a step $\eta(k)$, that is, the learning rate or the stepsize.

Steepest Descent Algorithm

Algorithm 1. Steepest Descent

```
begin initialize a, threshold θ, η(0)>0, k=0
   do  k ← k+1
   a ← a−η(k)∇J(a)
   until |η(k)∇J(a)|< θ
   return a
end
```

$$\mathbf{a}_{k+1} = \mathbf{a}_k - \eta_k \nabla J(\mathbf{a}_k)$$



Issues:

- **How to choose criterion function?**
- **How to choose learning rate *η(k)*?**
- **Convergence in local/global minimum?**
- **Convergence rate/speed/smoothness?**
- **Termination criterion?**

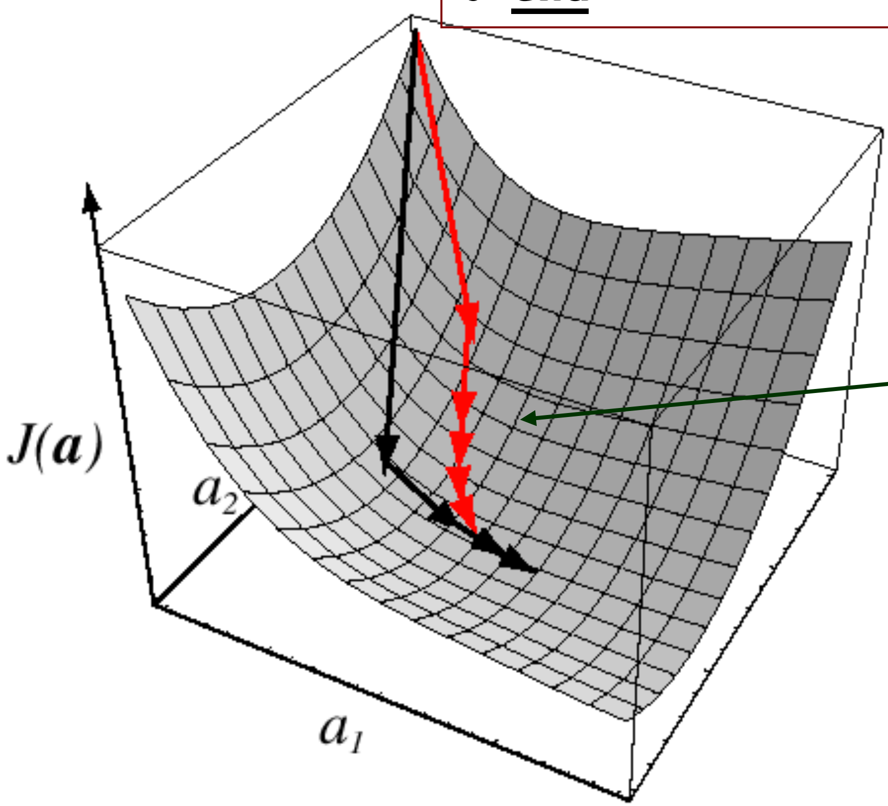
Newton Descent Algorithm

Algorithm 2. Newton Descent

```
begin initialize a, threshold θ
   a ← a-H^{-1}∇J(a)
   until |H^{-1}∇J(a)|< θ
   return a
end
```

$$\mathbf{a}_{k+1} = \mathbf{a}_k - \mathbf{H}_k^{-1} \nabla J(\mathbf{a}_k)$$

$$\mathbf{H}_k = \left[\partial^2 J(\mathbf{a}) / \partial a_i \partial a_j \right]_{\mathbf{a}=\mathbf{a}_k}$$



Red: Steepest Descent
Black: Newton Descent

Newton: greater improvement in each step at the computational cost of the inversion of the Hessian matrix *H*.

- Choices of the Criterion Function
  - A first choice: No of erroneously classified training samples. However: non-continuous function, hence non-differentiable.
  - A better choice: The perceptron critrerion function:

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

  where $\mathcal{Y}(\boldsymbol{a})$ is the set of training samples that are erroneously classified by $\boldsymbol{a}$.
  - If $\mathcal{Y}(\boldsymbol{a})$ is empty, then $J_p(\boldsymbol{a})=0$. Given $\boldsymbol{a}^t\boldsymbol{y}<0$ when $\boldsymbol{y}$ erroneously classified, $J_p(\boldsymbol{a})$ is never negative and becomes zero when $\boldsymbol{a}$ is a solution vector.
  - Geometric interpretation: $J_p(\boldsymbol{a})$ is proportional to the sum of the distances of the erroneously classified samples from the decision boundary.

Four Cost Functions

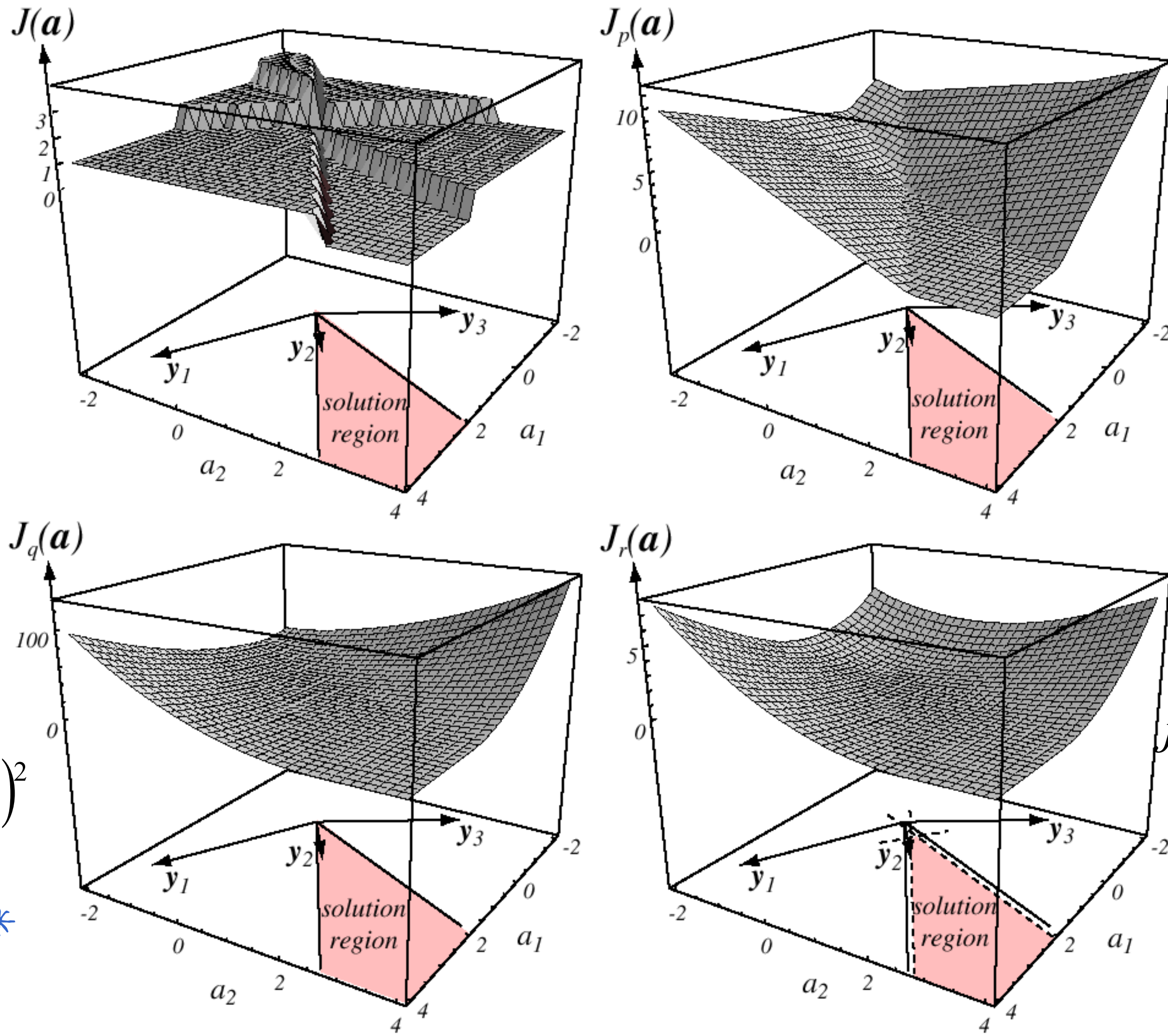
No of erroneous classifications

Bad

Perceptron Criterion

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

Good!



Total square error (TSE)

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

Better*

TSE with margin

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

Best*

* Αλλά θα μπορούσε να έχει μεγάλο υπολογιστικό κόστος

Batch Perceptron Algorithm

- After differentiating: $$\nabla J_p(\mathbf{a}) = \left[\frac{\partial J_p}{\partial a_i}\right] = \sum_{y \in \mathcal{Y}} -\mathbf{y}$$
- Recursive formulation: $$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k) \sum_{y \in \mathcal{Y}_k} \mathbf{y}$$

  where $\mathcal{Y}_k$ is the set of erroneously classified samples by $\boldsymbol{a}_k$.

- Next weight vector (w.v.) estimated as the sum of the current w.v. and a multiple of the sum of the erroneously classified samples.

Algorithm 3. Batch Perceptron

begin initialize **a**, criterion θ, $\eta(0)>0$, k=0

    do k ← k+1

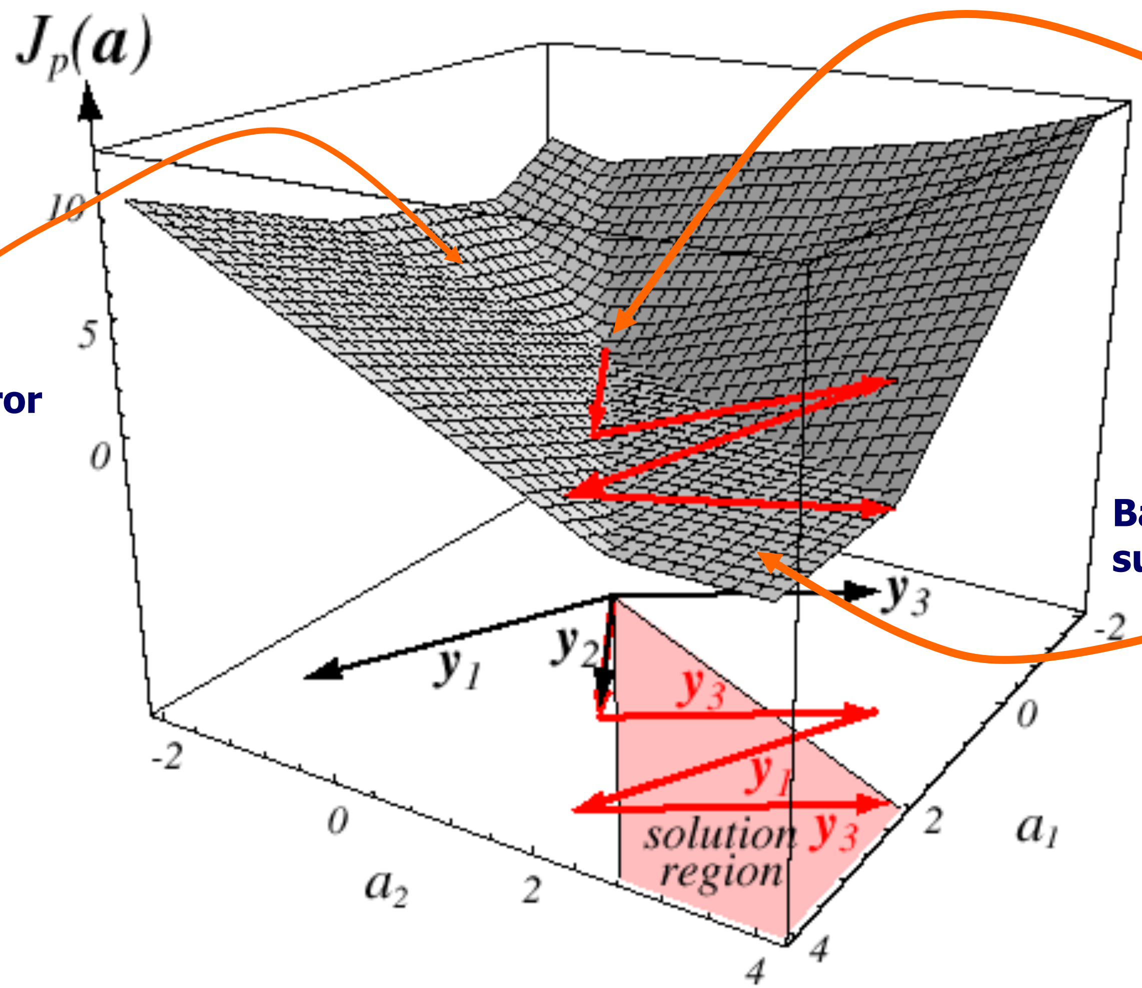
        $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{y \in \mathcal{Y}_k} \mathbf{y}$

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

    return **a**

end

Batch Perceptron Steps



Surface of Cost Function (also known as Error Surface)

a(1)=0

Base of error surface

Fixed-Increment Single-Sample Perceptron

- Instead of applying the weight vector ***a**(k)* to all samples and then correcting it based on the $\mathcal{Y}_k$ set of the erroneously classified samples, we use the samples <u>one at a time</u> and update or not the weight vector based on the classification result.
- Moreover, if we employ a <u>constant step *η(k)*</u>, then we derive the following algorithm:

Algorithm 4. Fixed-Increment Single-Sample Perceptron

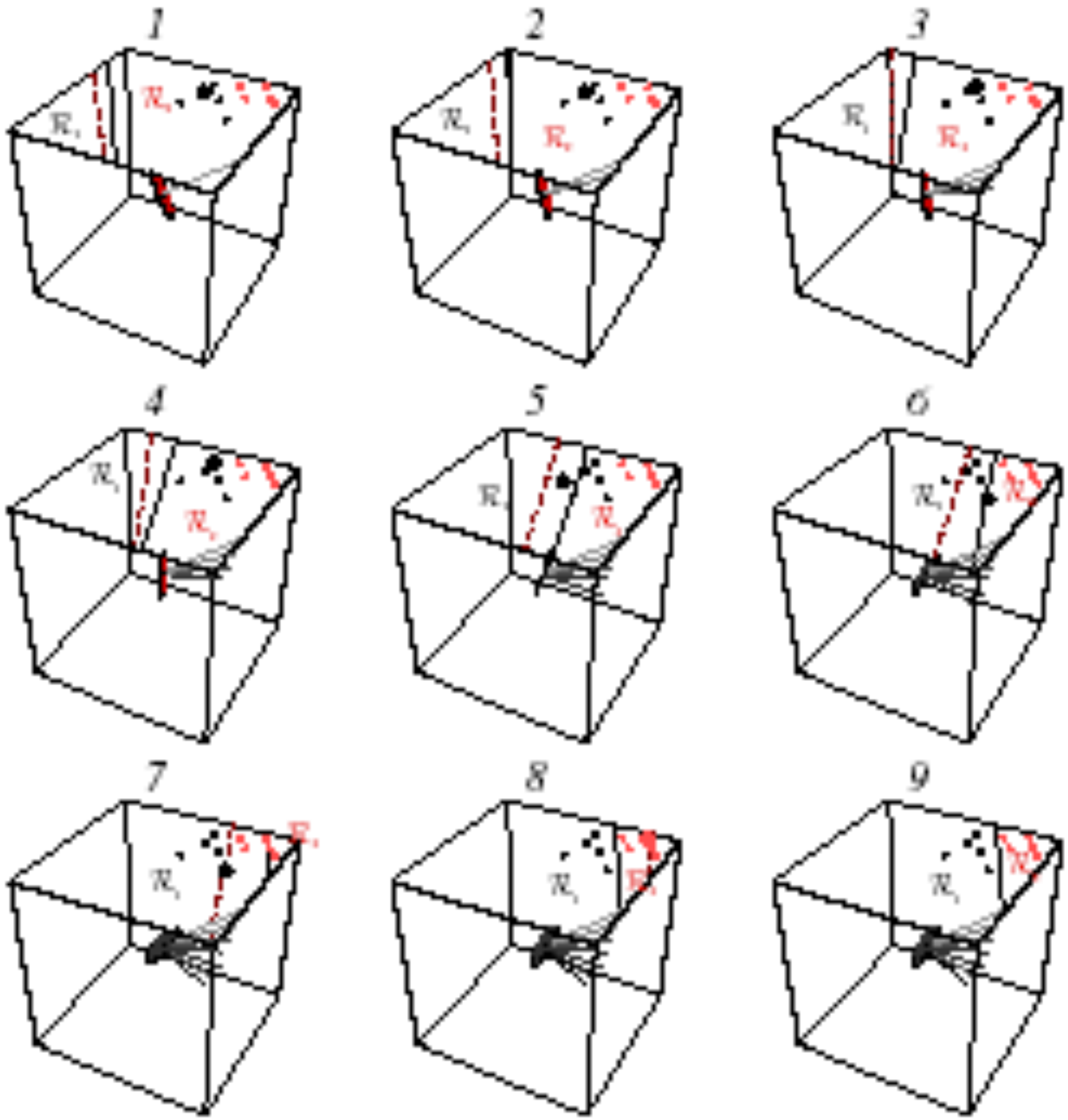
```
begin initialize a, k=0
   do k ← (k+1) mod n
      If y^k is misclassified by a, then a ← a + y^k
   until all patterns properly classified
   return a
end
```

Circular order of samples (green signifies the erroneously classified training samples):

$y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_1 \quad y_2 \quad y_3 \quad y_4$

$\rightarrow \quad y^1 \quad y^2 \quad y^3 \quad y^0 \quad y^1 \ldots = y_2 \quad y_1 \quad y_3 \quad y_2 \quad y_3$

Convergence



Variable-Increment Perceptron with Margin

- Use the samples <u>one at a time</u> and update the weight vector $\boldsymbol{a}(k)$ when its inner product with $\boldsymbol{y}^k$ is less than a preset threshold b: $\boldsymbol{a}(k)\boldsymbol{y}^k < b$.
- Moreover, if we employ a <u>variable step $\eta(k)$</u>, we get the following algorithm:

Algorithm 5. Variable-Increment Perceptron w. Margin

```
begin initialize a, threshold θ, margin b, η(0), k=0
   do  k ← (k+1) mod n
      if a^t y^k < b, then a ← a + η(k) y^k
   until a^t y^k > b for all k
   return a
end
```

Convergence Conditions:

$$\eta(k) \geq 0, \quad \lim_{m\to\infty} \sum_{k=1}^{m} \eta(k) = \infty, \quad \lim_{m\to\infty} \frac{\sum_{k=1}^{m} \eta^2(k)}{\left(\sum_{k=1}^{m} \eta(k)\right)^2} = 0$$

- Criterion Function:

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

where $\mathcal{Y}(\boldsymbol{a})$ is the set of samples for which $\boldsymbol{a}^t\boldsymbol{y}<b$.

- If $\mathcal{Y}(\boldsymbol{a})$ is empty, then $J_r(\boldsymbol{a})=0$. $J_r(\boldsymbol{a})$ can never become negative and is zero if and only if $\boldsymbol{a}^t\boldsymbol{y}>b$ for all the training samples.

- Gradient Vector:

$$\nabla J_r(\mathbf{a}) = \left[\frac{\partial J_r}{\partial a_i}\right] = \sum_{y\in\mathcal{Y}} \frac{\mathbf{a}^t\mathbf{y}-b}{\|\mathbf{y}\|^2}\mathbf{y}$$

- Update Rule:

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\sum_{y\in\mathcal{Y}_k} \frac{b-\mathbf{a}^t\mathbf{y}}{\|\mathbf{y}\|^2}\mathbf{y}$$

Batch Relaxation with Margin

Algorithm 6. Batch Relaxation with Margin

begin **initialize** $\mathbf{a}$, margin b, $\eta(0)$, $k=0$

  do $k \leftarrow (k+1) \bmod n$

    $\mathcal{Y}_k = \{\ \}$

    $j=0$

    do $j \leftarrow j+1$

      if $\mathbf{a}^t \mathbf{y}^k < b$, **then** append $\mathbf{y}^j$ to $\mathcal{Y}_k$

    until $j=n$

    $$\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{y \in \mathcal{Y}} \frac{b - \mathbf{a}^t \mathbf{y}}{\|\mathbf{y}\|^2} \mathbf{y}$$

  until $\mathcal{Y}_k = \{\ \}$

  return $\mathbf{a}$

end

Single-Sample Relaxation with Margin

Algorithm 7. Single-Sample Relaxation with Margin

begin initialize $\mathbf{a}$, margin b, $\eta(0)$, $k=0$

  do $k \leftarrow (k+1) \bmod n$

    if $\mathbf{a}^t\mathbf{y}^k < b$, **then** $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\dfrac{b-\mathbf{a}^t\mathbf{y}^k}{\|\mathbf{y}^k\|^2}\mathbf{y}^k$

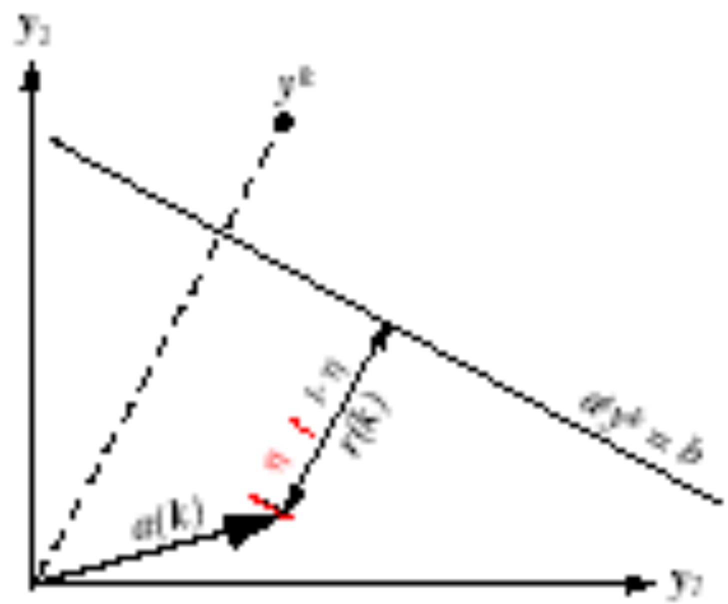
  until $\mathbf{a}^t\mathbf{y}^k > b$ for all $\mathbf{y}^k$

  return $\mathbf{a}$

end

At each step, weight vector $\mathbf{a}(k)$, is moved towards the hyperplane $\mathbf{a}^t\mathbf{y}^k=b$ by a fraction, $\eta(k)$, of its distance, $r(k)$, from that.

$\eta(k)<1$ → underrelaxation
$\eta(k)>1$ → overrelaxation
$0<\eta(k)<2$ for convergence



Minimum Square-Error – MSE

- Goal: Good performance in both cases, linearly separable and non-linearly separable.
- How: Criterion that involves all patterns. Moreover,
  - Before: Find $\boldsymbol{a}$ such as $\boldsymbol{a}^t\boldsymbol{y}_i>0$ for each pattern $\boldsymbol{y}_i$.
  - Now: Find $\boldsymbol{a}$ such as $\boldsymbol{a}^t\boldsymbol{y}_i=b_i$ for each pattern $\boldsymbol{y}_i$. (b_i positive constants).
- Accordingly: Instead of solving a problem of linear inequalities, solve one of linear equations.
- Notations:

$$\mathbf{Y}_{n\times\hat{d}} = \begin{bmatrix} \mathbf{y}_1^t \\ \mathbf{y}_2^t \\ \vdots \\ \mathbf{y}_n^t \end{bmatrix}$$

Pattern Matrix

$$\mathbf{b}_{n\times 1} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Vector of positive constants

$$\mathbf{a}_{\hat{d}\times 1} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix}$$

Vector of weights

- Formulation: Find $\boldsymbol{a}$ such as: $\boldsymbol{Ya}=\boldsymbol{b}$.

Minimum Square-Error – MSE

- Commonly $n>d+1$, number of patterns > dimensions → overdetermined system, hence no exact solution.
- Therefore: Minimization of the square of the error vector length,

e=Ya-b:

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

- Formulation:

$$\nabla J_s(\mathbf{a}) = \mathbf{0} \Rightarrow \mathbf{Y}^t\mathbf{Ya} = \mathbf{Y}^t\mathbf{b}$$

Normal Equations

- If $\boldsymbol{Y}^t\boldsymbol{Y}$ can be inverted,

$$\mathbf{a} = \underbrace{\left(\mathbf{Y}^t\mathbf{Y}\right)^{-1}\mathbf{Y}^t}_{\text{Pseudo-inverse}}\ \mathbf{b}$$

Widrow-Hoff (Least-mean squares-LMS)

- $J_s(a)$ can be minimized via recursive algorithms that do not require matrix inversions.
- Tangent vector: $\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t\left(\mathbf{Y}\mathbf{a} - \mathbf{b}\right)$
- Basic recursive formulae: $\mathbf{a}(k+1) = \mathbf{a}(k) + \eta(k)\mathbf{Y}^t\left(\mathbf{b} - \mathbf{Y}\mathbf{a}(k)\right)$
- By employing one sample at a step, the Widrow-Hoff (LMS) algorithm is derived:

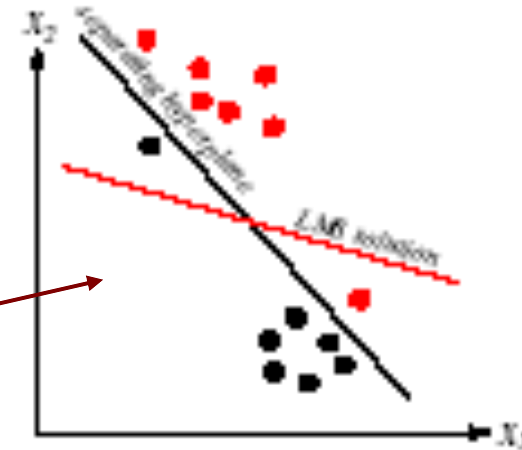
Αλγόριθμος 8. Widrow-Hoff (LMS)

begin initialize a, b, κριτήριο θ, η(), k=0
    do *k* ← *(k+1) mod n*
        $\mathbf{a} \leftarrow \mathbf{a} + \eta(k)\left(b_k - \mathbf{a}^t\mathbf{y}^k\right)\mathbf{y}^k$
    until $\left|\eta(k)\left(b_k - \mathbf{a}^t\mathbf{y}^k\right)\mathbf{y}^k\right| < \theta$
    return ***a***
end

Ho-Kashyap Method

- Perceptron and Relaxation procedures find separating Weight Vectors when the samples are linearly separable, but do not converge for non-separable classes.

- Least-mean squares offers always a solution vector (the one that minimizes $\|\mathbf{Ya}\text{-}\mathbf{b}\|^2$) which is not necessarily separating vector in the separable case.



- The Ho-Kashyap algorithm recursively solves the minimization problem:

$$\min_{\mathbf{a},\mathbf{b}} J_s(\mathbf{a},\mathbf{b}) = \left\|\mathbf{Ya} - \mathbf{b}\right\|^2 \quad s.t. \quad \mathbf{b} > \mathbf{0}$$

- Algorithmically it achieves ***b*** not to converge to ***0***, by setting all the positive components of the tangent vector $\nabla_{\boldsymbol{b}} J_s$ equal to zero.

Ho-Kashyap Algorithm

Algorithm 9. Ho-Kashyap

```
begin initialize a, b, η()<1, threshold b_min, k_max
   do k ← (k+1) mod n
        e ← Ya-b
        e+ ← (e+|e|)/2
        b ← b + 2η(k)e+
        a ← (Y^tY)^-1 Y^t b
        if Abs(e)< b_min then return a, b and exit
   until k= k_max
   print "No solution found"
end
```

- Goal: Linear separation of multiple classes:

  If $\boldsymbol{y} \sim \omega_1$, then $\boldsymbol{a}_1^t\boldsymbol{y}-\boldsymbol{a}_j^t\boldsymbol{y}>0$ for all $j=2,\ldots,c$.

- The system of *c-1* inequalities can be described as:
  The $c\hat{d}-D$ weight vector $\hat{\boldsymbol{\alpha}}$ classifies correctly all *c-1* $c\hat{d}-D$ patterns $\boldsymbol{\eta}_{12}$, $\boldsymbol{\eta}_{13}, \ldots, \boldsymbol{\eta}_{1c}$: $\quad \hat{\boldsymbol{\alpha}}^t \boldsymbol{\eta}_{1j} > 0 \quad \forall \quad j = 2, \ldots, c$ where:

$$\hat{\boldsymbol{\alpha}}_{c\hat{d}\times 1} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_c \end{bmatrix} \qquad \boldsymbol{\eta}_{12} = \begin{bmatrix} \mathbf{y} \\ -\mathbf{y} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \qquad \boldsymbol{\eta}_{13} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \\ -\mathbf{y} \\ \vdots \\ \mathbf{0} \end{bmatrix} \quad ,\cdots, \quad \boldsymbol{\eta}_{1c} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ -\mathbf{y} \end{bmatrix}$$

Kesler Structure

- Generally: $\quad \hat{\boldsymbol{\alpha}}^t \boldsymbol{\eta}_{ij} > 0 \quad \forall \quad j \neq i,$ with $\boldsymbol{\eta}_{ij} = \begin{bmatrix} \vdots \\ \mathbf{y} \\ \vdots \\ -\mathbf{y} \\ \vdots \end{bmatrix} \begin{matrix} \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \end{matrix}$

Multi-Class Perceptron Classification

- Let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ patterns from c classes, linearly separable. Let L_k a linear machine $\boldsymbol{a}_1(k), \ldots, \boldsymbol{a}_c(k)$. We seek to formulate a series of linear machines $L_1, \ldots, L_k, \ldots$ that converges to a separation (classification) machine L.

- Let $\boldsymbol{y}^k$ the k-th sample that is amenable to correction (correct classification). If $\boldsymbol{y}^k \sim \omega_i$, then at least one $j \neq i$ exists for which $\boldsymbol{a}_i^t(k)\, \boldsymbol{y}^k < \boldsymbol{a}_j^t(k)\, \boldsymbol{y}^k$.

- L_k correction rules specifies (perceptron with constant unit-step) :

$$\boldsymbol{\alpha}(k+1) = \boldsymbol{\alpha}(k) + \boldsymbol{\eta}_{ij}^k \quad \text{where} \quad \boldsymbol{\alpha}^t(k)\boldsymbol{\eta}_{ij}^k \leq 0 \quad \text{with} \quad \boldsymbol{\alpha}(k) = \begin{bmatrix} \mathbf{a}_1(k) \\ \vdots \\ \mathbf{a}_c(k) \end{bmatrix} \quad \text{and} \quad \boldsymbol{\eta}_{ij}^k = \begin{bmatrix} \vdots \\ \mathbf{y}^k \\ \vdots \\ -\mathbf{y}^k \\ \vdots \end{bmatrix} \begin{matrix} \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \end{matrix}$$

Therefore:

$$\begin{aligned} \mathbf{a}_i(k+1) &= \mathbf{a}_i(k) + \mathbf{y}^k \\ \mathbf{a}_j(k+1) &= \mathbf{a}_j(k) - \mathbf{y}^k \\ \mathbf{a}_l(k+1) &= \mathbf{a}_l(k) \quad l \neq i \quad \text{και} \quad l \neq j \end{aligned}$$