UNIVERSITÀ
di VERONA

Machine Learning

Linear Discriminant Functions

Cigdem Beyan

A. Y. 2024/2025

Linear discriminant functions

- The objective of this lecture is to present methods for learning **linear discriminant** functions of the form

$$g(x) = w^T x + w_0 \Leftrightarrow \begin{cases} g(x) > 0 & x \in \omega_1 \\ g(x) < 0 & x \in \omega_2 \end{cases}$$

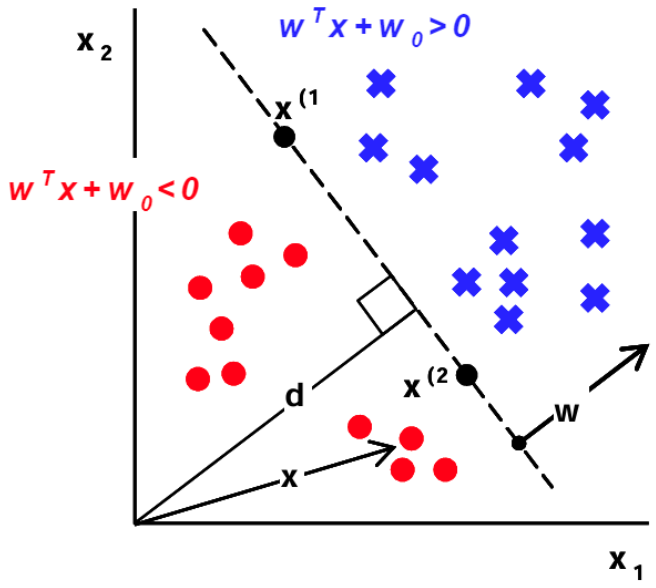
where w is the weight vector and w_0 is the threshold weight or bias.

$$\mathbf{x}=[x_1,\ldots,x_n],\ \mathbf{w}=[w_1,\ldots,w_n]$$

- For a problem with 2 classes, the goal is to create **a linear function** *g(x)* that separates the examples belonging to the two classes.

g(x)=0 ➔ either class

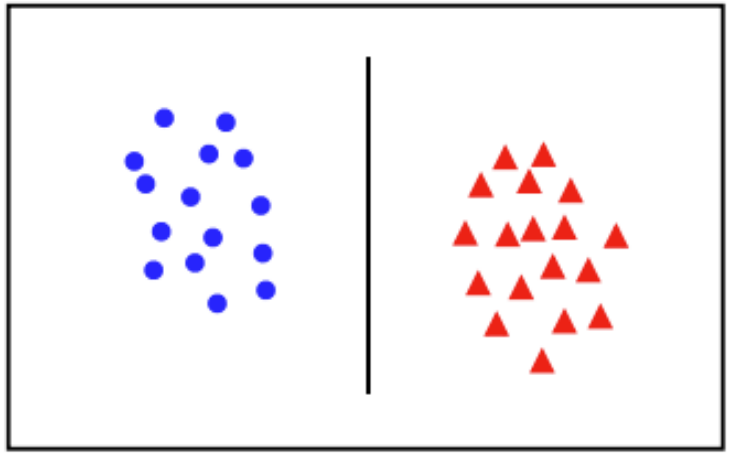
Linear discriminant functions

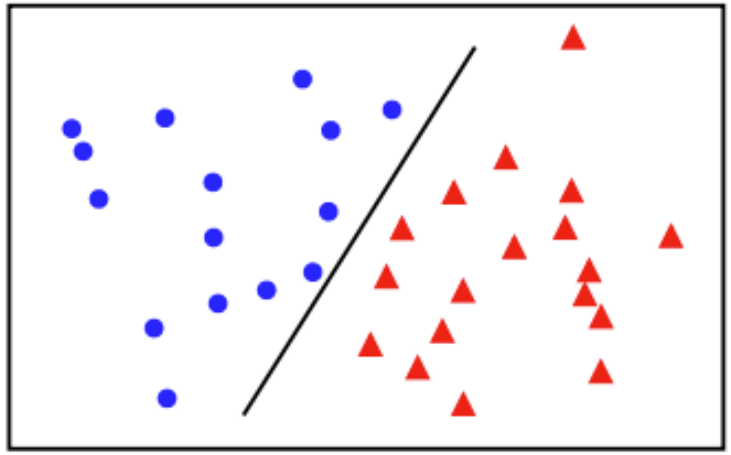


- Suppose we have a set of *m* samples $x_1,\ldots,x_m$, some tagged ω_1 and others tagged ω_2 .
- We want to use these samples to determine the weights **w** and w_0 of the linear discriminating function.
- Suppose also that there is a solution for which the probability of error is very low.
- Thus, a reasonable approach is to look for a weight vector that classifies all of the samples correctly or that the probability of making a mistake is zero.
- If such a **weight vector** exists, the samples are said to be *linearly separable*.

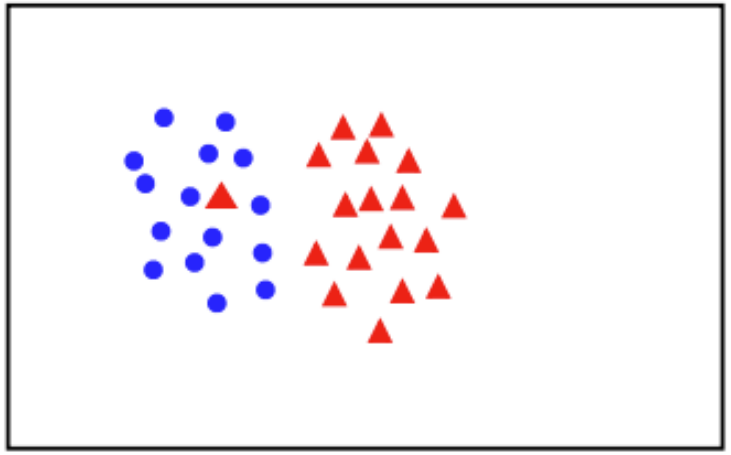
Linear separability

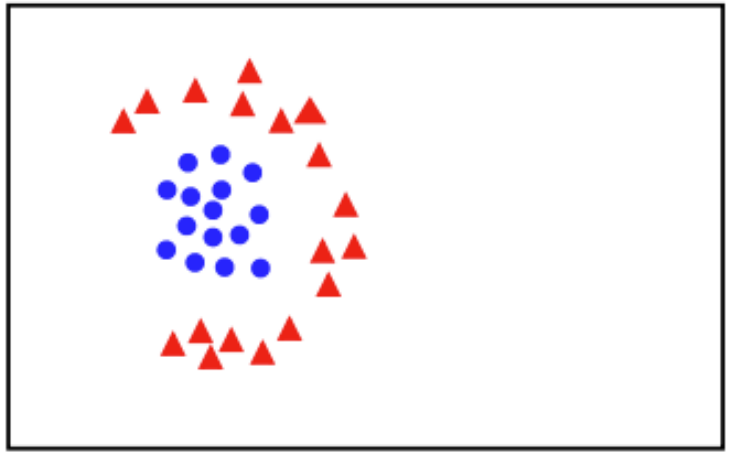
Linearly separable





NOT Linearly separable



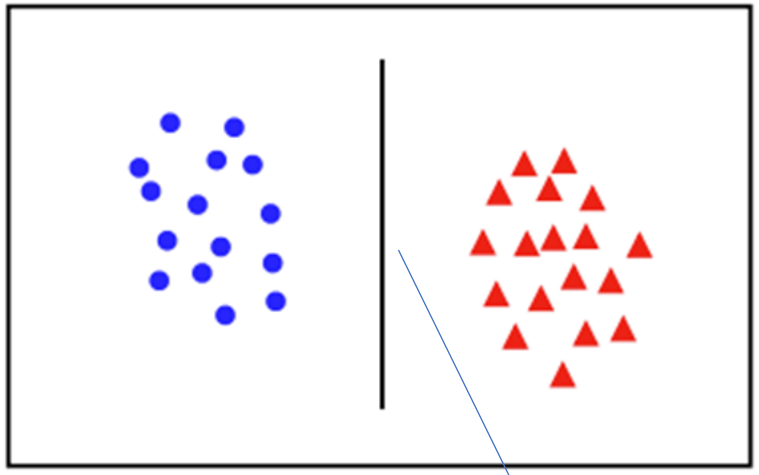


Linear discriminant functions

- Linear Discriminant functions are optimal for Gaussian distributions with equal covariance.
- May not be optimal for other data distributions, but they are very simple to use
- Knowledge of class densities is not required when using linear discriminant functions
  - We can say that this is a non-parametric approach

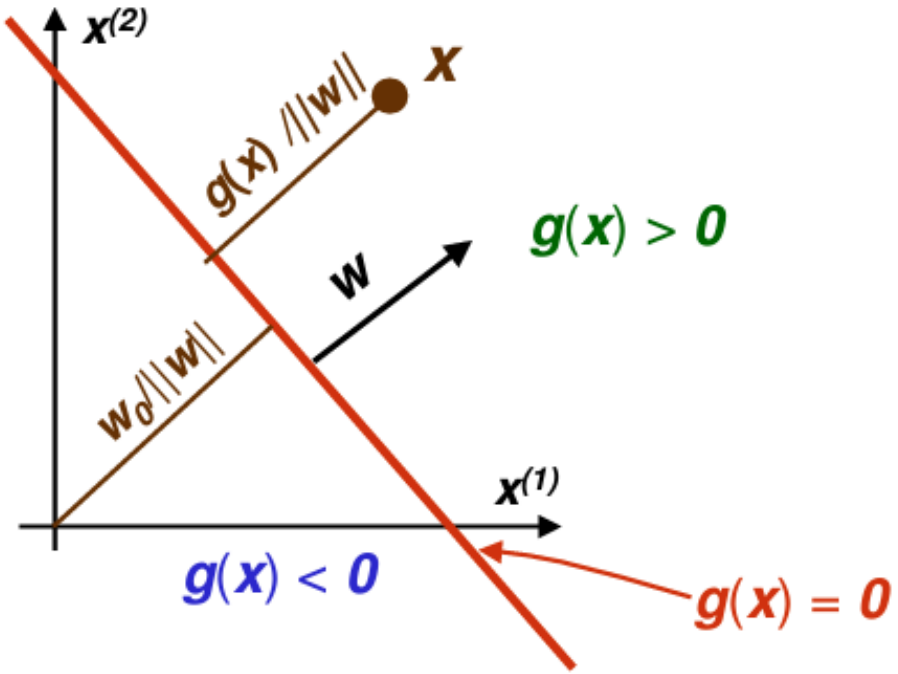
Linear classifiers

$$g(x) = w^T x + w_0 \Leftrightarrow \begin{cases} g(x) > 0 & x \in \omega_1 \\ g(x) < 0 & x \in \omega_2 \end{cases}$$



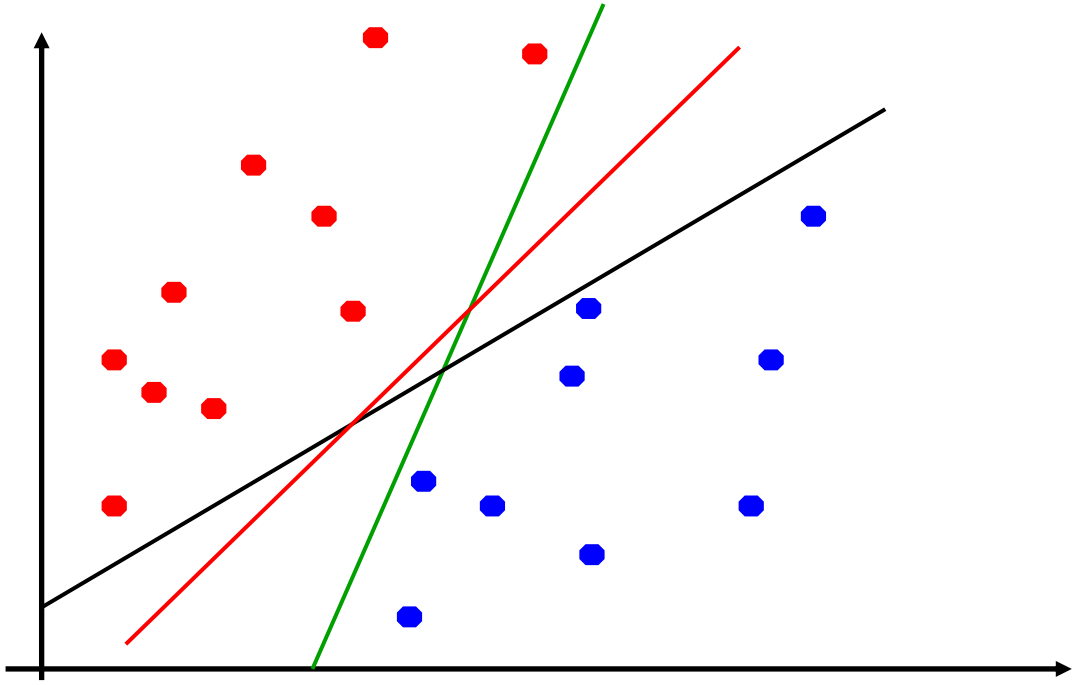
- Decision boundary $g(x)=w^T x+w_o=0$
- in 2D, is a line
- in 3D, is a plane
- in nD, is a hyperplane

Linear classifiers



- In linear classifiers, the training data is used to learn $\boldsymbol{w}$. *Only w is needed to classify the new data point.*
- The equation $\mathbf{w}^t\mathbf{x}_i = 0$ defines a hyperplane passing through the origin of the feature space having $\mathbf{w}$ as a normal vector.
- $\boldsymbol{w}$ determines orientation of the decision hyperplane.
- $\boldsymbol{w_0}$ determines location of the decision surface.

Which hyperplanes?



- If a solution vector exist (if there is a hyperplane) then,
  - Which hyperplane a linear classifier choose when the data is linearly separable?
- The solution vector is not unique!

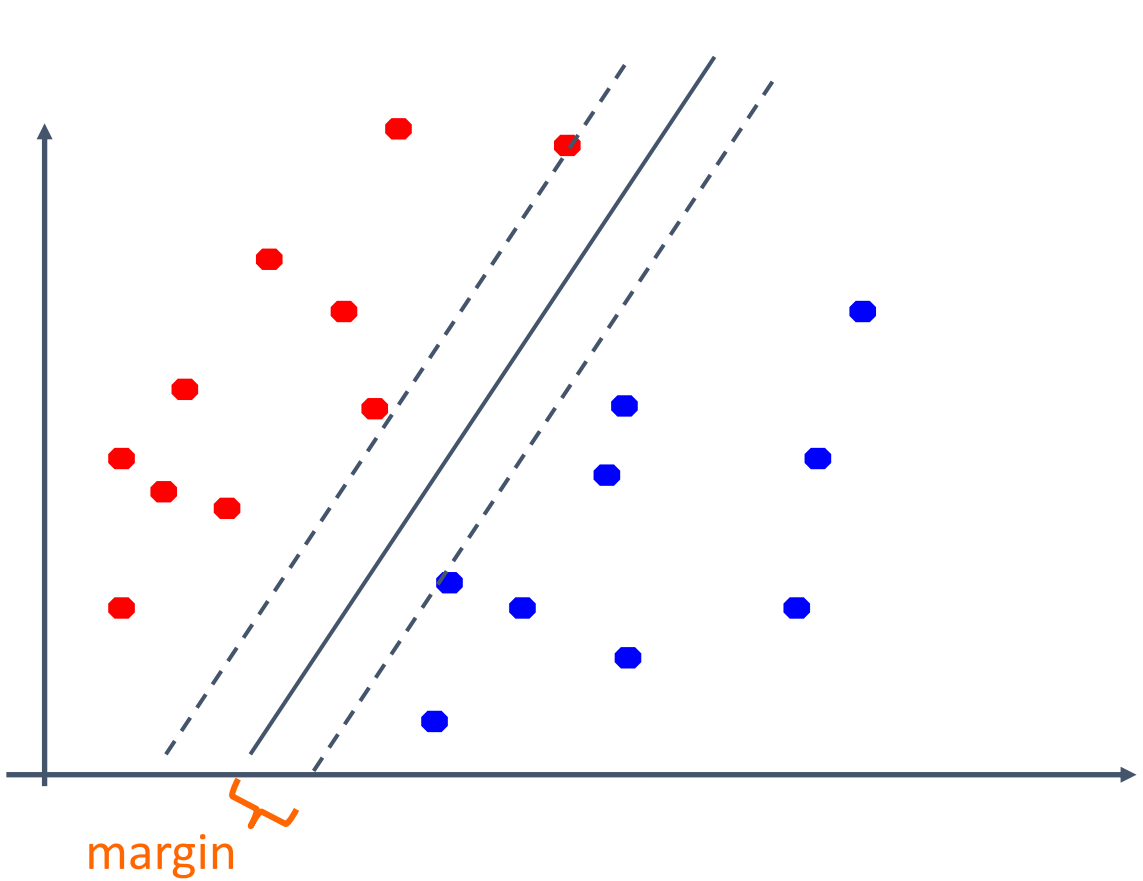
Linear discriminant functions

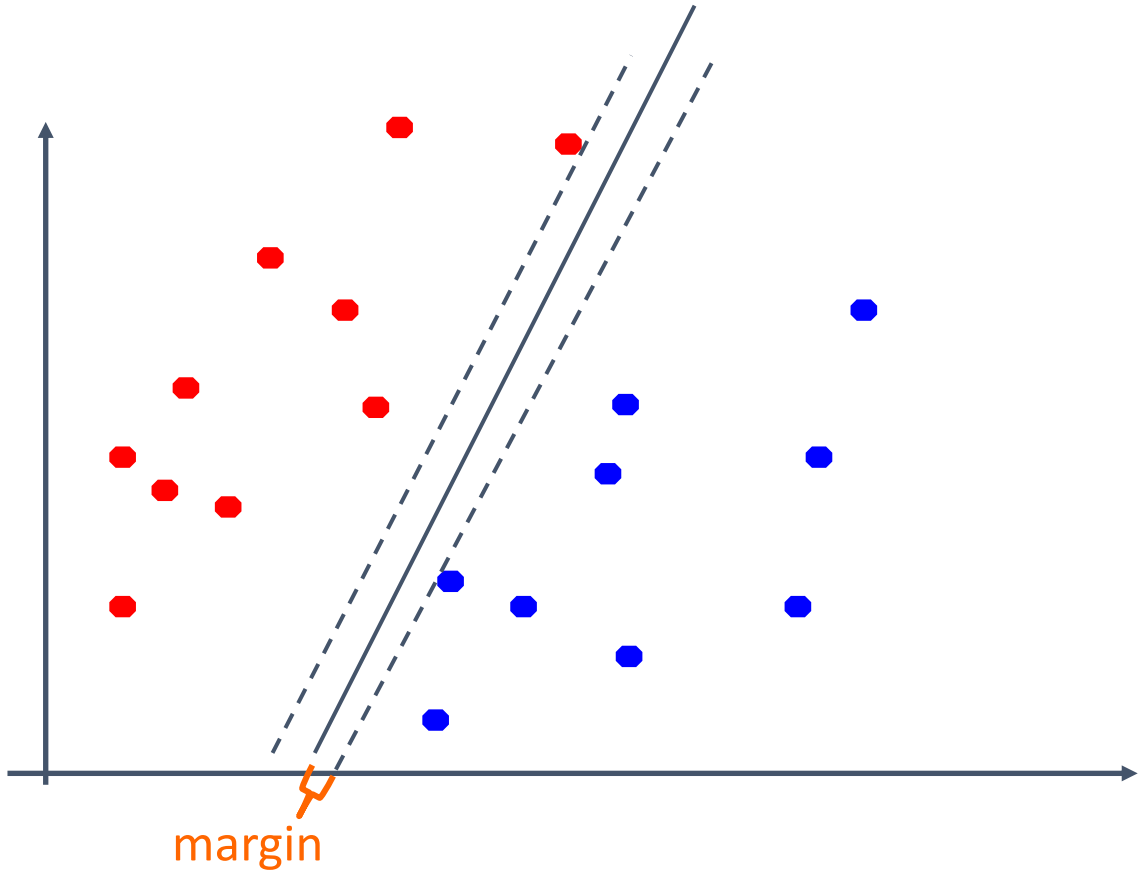
- One possibility is to seek a **unit-length weight vector** that maximizes the minimum distance from the samples to the separating plane.
- Another possibility is to seek the minimum-length weight vector satisfying

$$\mathbf{w}^t\mathbf{x}_i \geq b, \quad \forall i$$

where b is a positive constant called *margin*.

- These techniques attempt to find a solution vector closer to the "middle" of the solution region:
  - In this way the resulting solution is more likely to classify new test samples correctly.

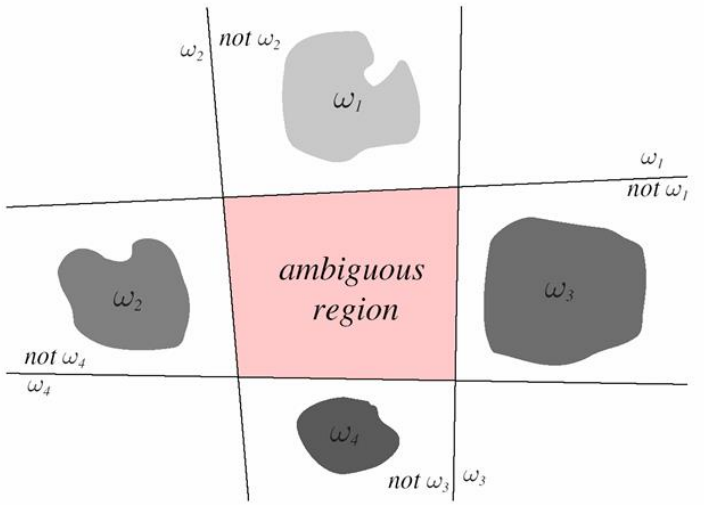


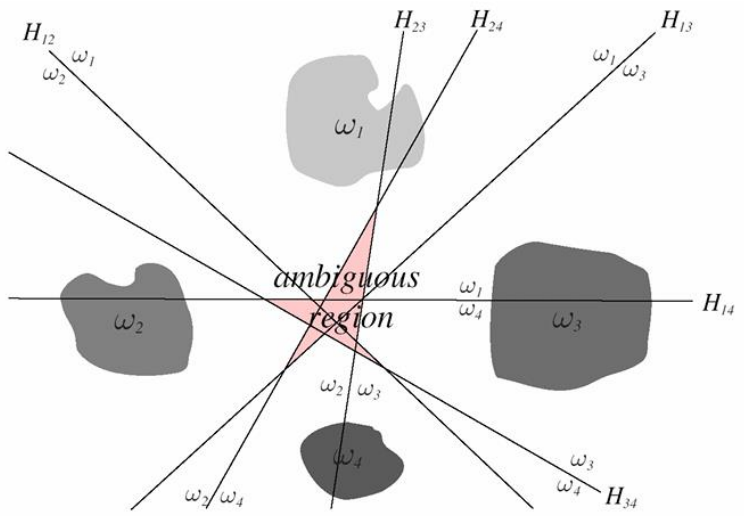


The **margin** of a classifier is the distance to the closest points of either class.

Linear discriminant functions

- In case of a multi-class (*C-class*) problem:
  - One-vs-rest classifiers: C-1 classifiers $g_i(\mathbf{x})$ are constructed, one for each class C_i versus non-C_i.
  - One-vs-one: C(C-1)/2 binary classifiers (one-vs-one), and then classify a test sample following a **majority voting**.
  - Both lead to areas of **ambiguity**.





LDF: Augmented feature vector

- LDF: $g(x)=w^T x+w_o$
- We can write it: $g(x) = [w_0 \quad w^t] \begin{bmatrix} 1 \\ x \end{bmatrix} = a^t y = g(y)$

  new weight vector a *new feature vector y*
- **y** is called the **augmented feature vector**
- We added a dummy dimension to get a completely equivalent new homogeneous problem.

old problem $g(x) = w^t x + w_0$ $\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$

new problem $g(y) = a^t y$ $\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$

- Feature augmenting is done only for simpler notation.
- Throughout these lecture notes we replace ***a*** with ***w*** for simplicity, since they both refer to *weights*.

LDF: Training Error

- For the rest of the lecture, assume we have 2 classes.
- We will use samples (training data) to determine weigths $\boldsymbol{a}$ (or indicated with $\boldsymbol{w}$) in the discriminant function $\mathbf{g(y)= a^T y}$
- What should be our criterion for determining **a**?
  - Suppose, for now that we want to minimize the training error (the number of misclassified samples $y_1,\ldots y_n$)
- Recall that $g(y_i) > 0$ ➔ y_i classified as class 1
  $g(y_i) < 0$ ➔ y_i classified as class 2
- Thus, training error is ***0*** *if*

$$\begin{cases} g(\boldsymbol{y}_i) > 0 \quad \forall \boldsymbol{y}_i \in \boldsymbol{c}_1 \\ g(\boldsymbol{y}_i) < 0 \quad \forall \boldsymbol{y}_i \in \boldsymbol{c}_2 \end{cases}$$

LDF: Training Error

- Thus, training error is ***0*** *if*

$$\begin{cases} g(y_i) > 0 & \forall y_i \in c_1 \\ g(y_i) < 0 & \forall y_i \in c_2 \end{cases}$$

$$\begin{cases} a^t y_i > 0 & \forall y_i \in c_1 \\ a^t y_i < 0 & \forall y_i \in c_2 \end{cases}$$

$$\begin{cases} a^t y_i > 0 & \forall y_i \in c_1 \\ a^t (-y_i) > 0 & \forall y_i \in c_2 \end{cases}$$

LDF: "Normalization"

- This suggest a so-called **normalization**
  - Replace all examples from class 2 by their negatives

$$y_i \rightarrow -y_i \qquad \forall y_i \in c_2$$

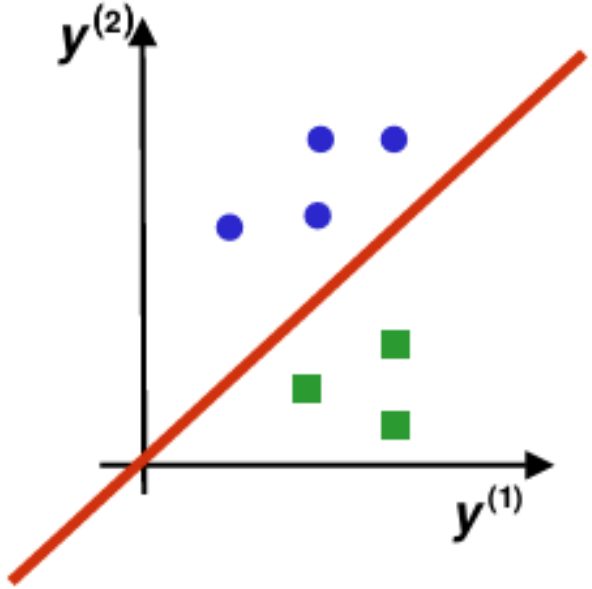
  - See weight vector **a** such that

$$a^t y_i > 0 \qquad \forall y_i$$

  - If such **a** exists, it is called a separating or solution vector
  - Original samples $x_1,\ldots x_n$ can indeed be separated by a line in that way.

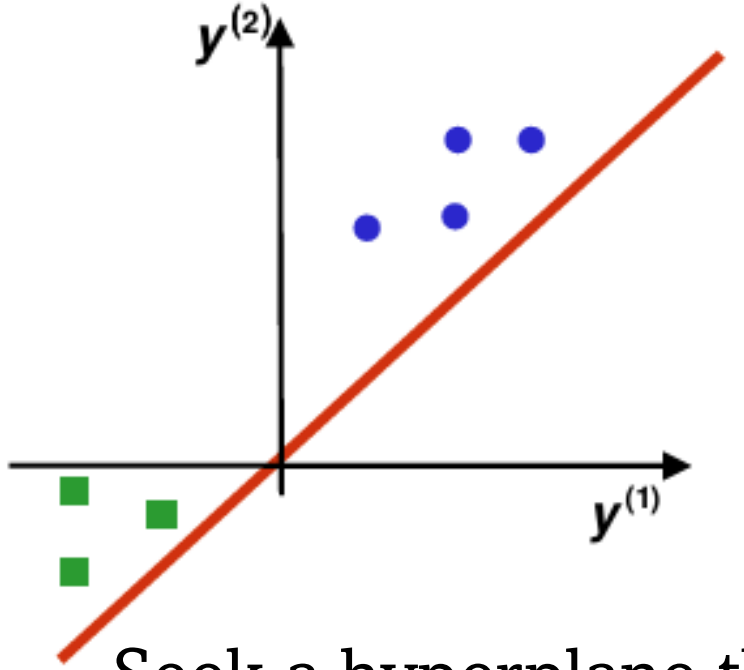
LDF: "Normalization"

Before normalization



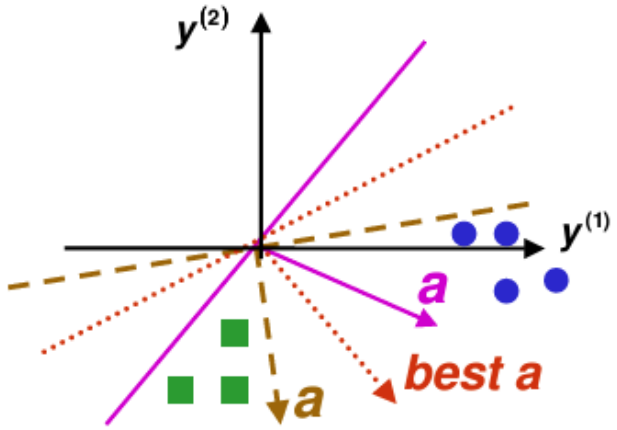
Seek a hyperplane that separates patterns from different categories.

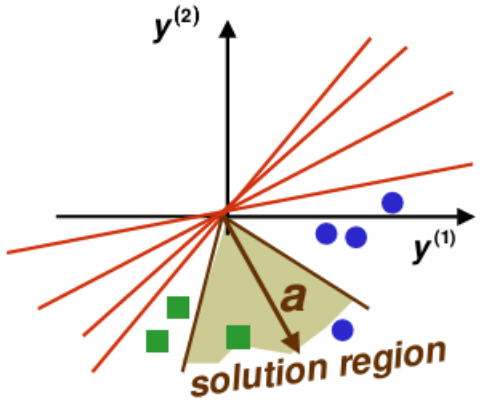
After normalization



Seek a hyperplane that puts normalized patterns on the same positive side.

LDF: Solution Region





- Coming back to the "which hyperplace" question….
- We can see that many solution for weight vector **a** exist.

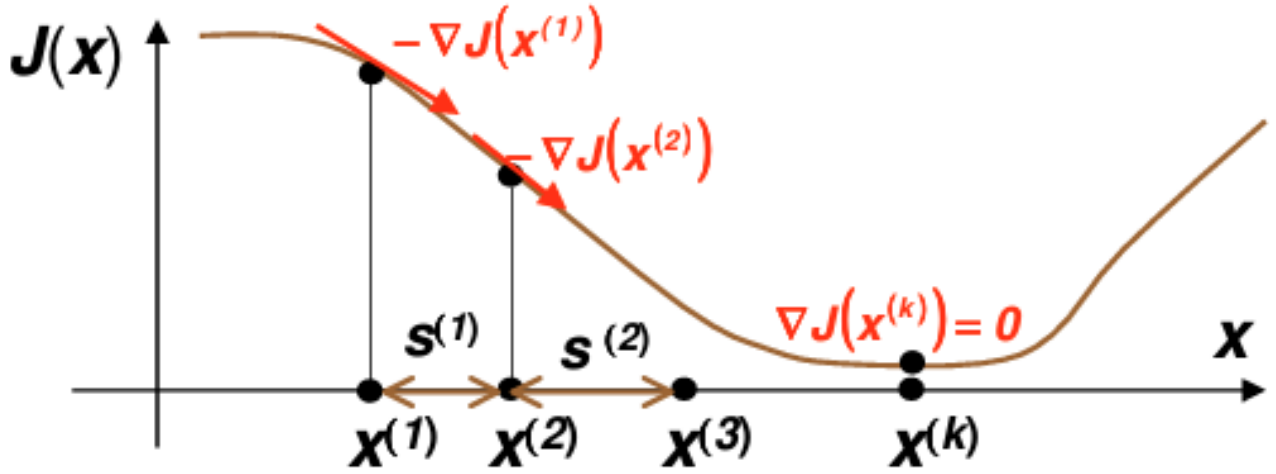
$$a^t y_i = \sum_{k=0}^{d} a_k y_i^{(k)} > 0$$

- Solution region for **a**: set of all possible solutions defined in terms of normal **a** to the separating hyperplane.

Gradient Descent

- **Gradient Descent** is one of the simplest approaches to calculating a discriminant function.
- It is an iterative method of to progressively adjusting the **weights**.

*The gradient vector in space **W** points to the direction of maximum deviation of a function to be maximized/minimized.*



Gradient Descent

- The procedure consists of updating the value of the weight vector at the k+1 step with a contribution proportional to the gradient module calculated at the previous step:

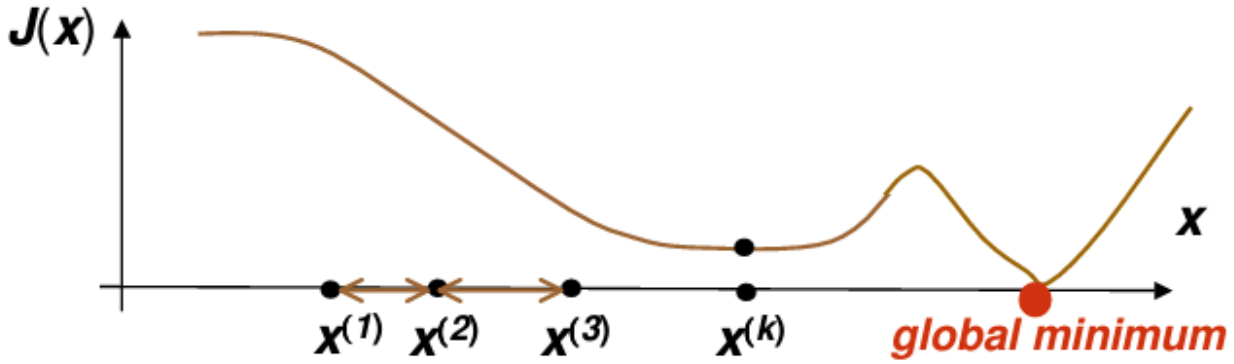
$$\mathbf{w}(k+1) = \mathbf{w}(k) - \rho_k \nabla J(\mathbf{w})\Big|_{\mathbf{w}=\mathbf{w}(k)}$$

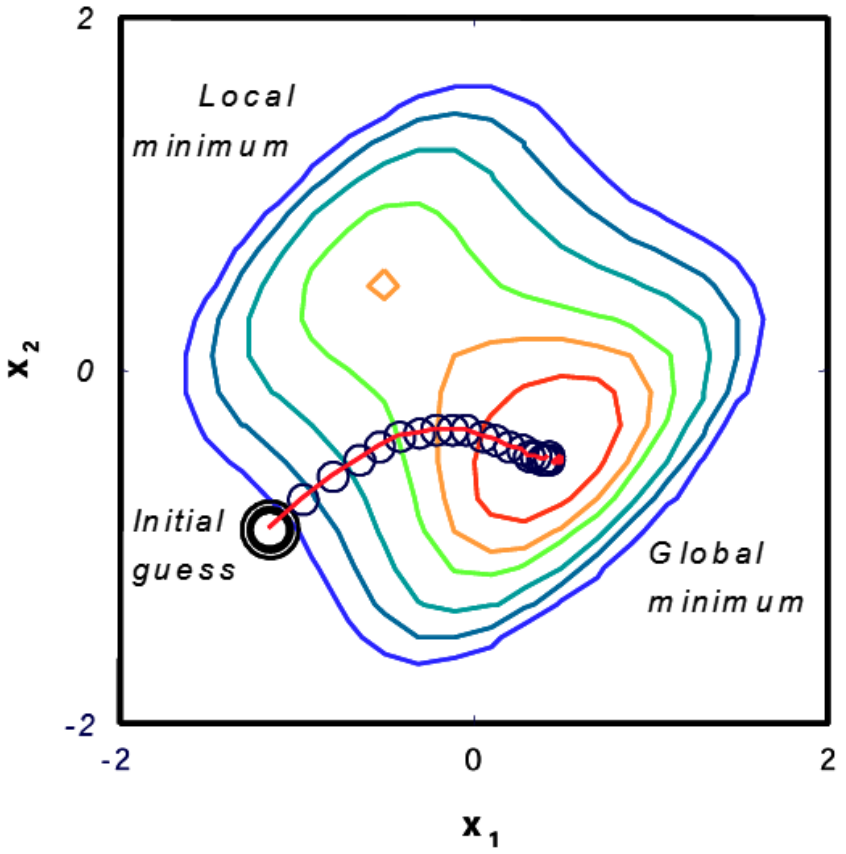
where ***J(w)*** is an evaluation function (*loss function*) that must be minimized.

- ***J(w)*** is chosen in such a way as to reach the minimum when ***w*** approaches to the optimal solution, that is, ***J(w)*** should be convex, and ***w*** is the solution vector.

Gradient Descent

- The minimum of ***J(w)*** is obtained by moving **w** in the direction opposite to the gradient (*in the direction of steepest descent*)





- Gradient descent is guaranteed to find only a **local minimum**.
- Very popular, as being simple and
- applicable to any function.

Gradient Descent

- ∇ is the gradient operator symbol, given by:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \vdots \\ \vdots \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{bmatrix}$$

- ρ_k (**learning rate**) is an appropriate scalar value that varies with the iteration k, fixing the "magnitude" of the correction.

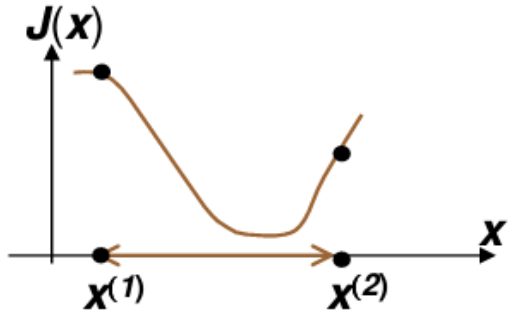
Gradient Descent

- The procedure at the *k*-th step can be summarized as:
  - Arbitrarily choose a weight vector **w(1)** and calculate the gradient vector *J(w(1))*;
  - Obtain a vector **w(2)** at a certain distance (also fixed by means of ρ_k) in the direction of **maximum descent** (steepest descent).

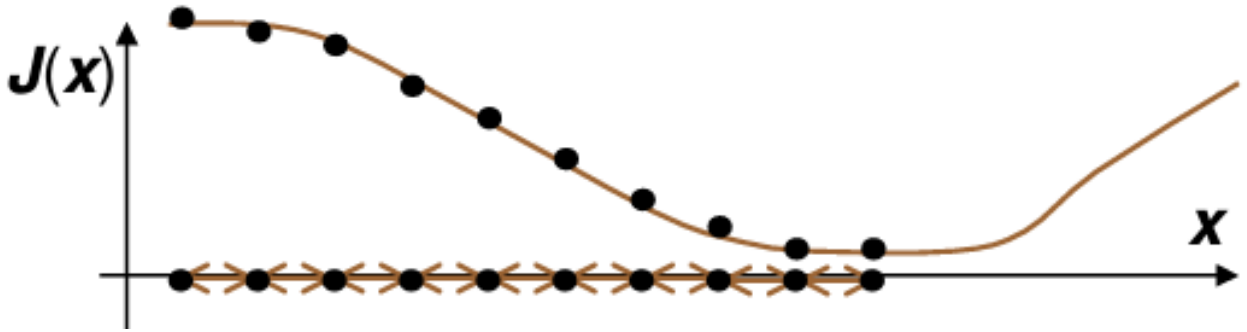
- Main issue: How to determine the optimal choice of ρ_k

Gradient Descent

- Main issue: How to determine the optimal choice of ρ_k
- If ρ_k is too **large**, gradient descent may overshoot the minimum and possibily never find the minimum.



- If ρ_k too **small**, need too many iterations



Perceptron

- Let's learn a **binary classifier** with linear discriminant function.

!!!! From this point to the rest ***w*** and ***x*** are used for an augmented feature vector for simplicity, instead of using ***a*** and ***y***.

- To do so, we need to choose a *J(w)* such that the gradient descent can solve it, in this way we can find the minimum of *J(w)*.
- A good choice is **Perceptron** criterion function: $J(\mathbf{w}) = -\sum_{i \in X} \mathbf{w}^t \mathbf{x}_i$

where *X* is a set of samples misclassified by ***w***.

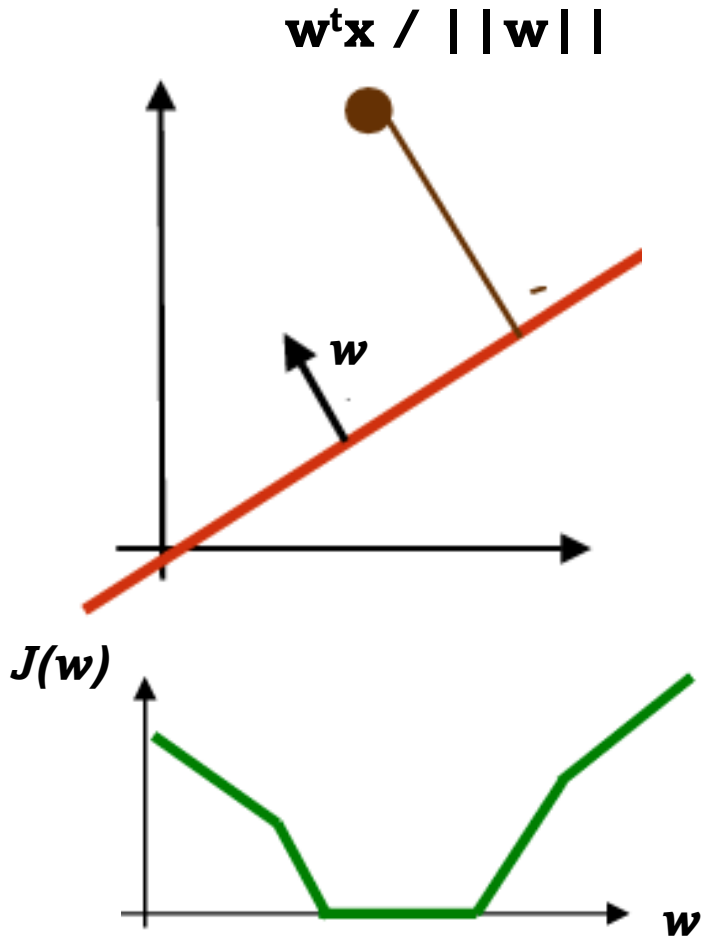
J(w) is non-negative since $\mathbf{w}^t\mathbf{x}_i < 0$ for all misclassified samples.

- Do not forget to "normalize" the training set by replacing all examples from class ω_2 by their negatives.

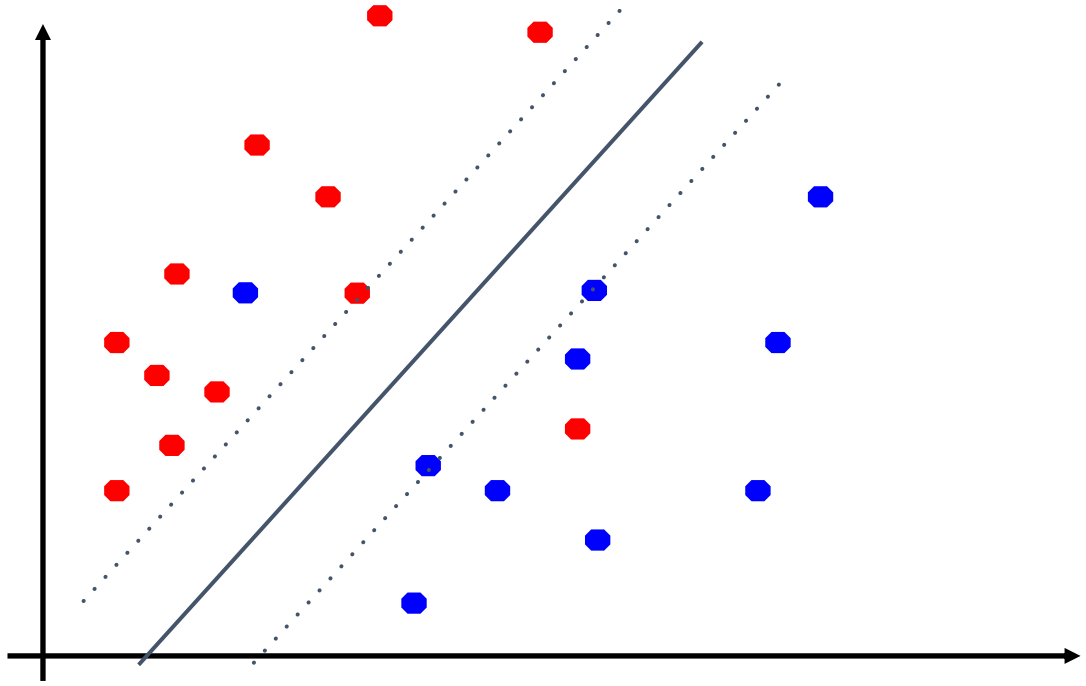
Perceptron

$$J(\mathbf{w}) = -\sum_{i \in X} \mathbf{w}^t \mathbf{x}_i$$

- Geometrically, ***J(w)*** is proportional to the sum of the distances of misclassified samples from the decision boundary.
- ***J(w)*** is piecewise linear and thus suitable for gradient descent.
- However, ***J(w)*** converges only if there is a linear separation between the 2 classes.
  - Not always happens!



Perceptron



We want to gradient descent to solve the ***J(w)*** so that it can converge however for such a data (not linearly separable), we cannot converge.

Perceptron

- As we said, to find the minimum of ***J(w)***, we use gradient descent, which is defined by:
  - *i*-th component of the gradient of $J(\mathbf{w})$ is equal to $\partial J/\partial w_i$, thus

$$J(\mathbf{w}) = -\sum_{i \in X} \mathbf{w}^t \mathbf{x}_i \quad \Rightarrow \quad \nabla J = -\sum_{i \in X} \mathbf{x}_i$$

Note that this does not depend on ***w***

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \rho_k \cdot \sum_{i \in X} \mathbf{x}_i, \quad \text{when} \quad \rho_k = \frac{1}{k}$$

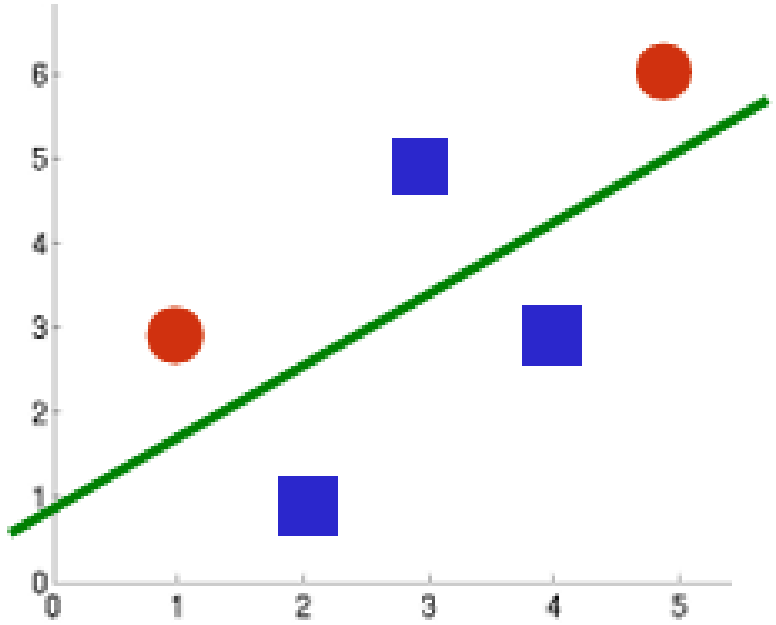
Perceptron rule

- Perceptron is the first Neural Network, which is then evolved to more complex versions such as Multilayer Perceptron.

Perceptron -- summary

- If classes are linearly separable, the perceptron rule is guaranteed to converge to a valid solution
  - Some version of the perceptron rule use a variable learning rate $p(k)$, for now we fixed it as $1/k$
- However, if the two classes are **not linearly separable**, the perceptron rule will **not converge**
  - Since no weight vector w can correctly classify every sample in a non separable dataset, the corrections in the perceptron rule will never cease.
  - One ad-hoc solution to this problem is to enforce convergence by using variable learning rates $p(k)$ that approach zero as $k \to \infty$
    - We will see once we introduce NN architectures.

Perceptron – example (nonseparable scenario)



- Suppose we have 2 features and 5 samples for 2 classes.
- Class 1: [2,1], [4,3], [3,5]
- Class 2: [1,3], [5,6]
- These samples are not separable with a line, still we will try to get approximate separation by a line.

- Obtain the augmented features and normalize the data, by doing so we obtain:

$$\begin{bmatrix}1\\2\\1\end{bmatrix}\ \begin{bmatrix}1\\4\\3\end{bmatrix}\ \begin{bmatrix}1\\3\\5\end{bmatrix}\ \begin{bmatrix}-1\\-1\\-3\end{bmatrix}\ \begin{bmatrix}-1\\-5\\-6\end{bmatrix}$$

Perceptron – example (nonseparable scenario)

- Let's apply perceptron method
  - Initial weigths [1 1 1]
    - This is a line $x_1 + x_2 + 1 = 0$
- Fixed learning rate $\boldsymbol{p(k)} = 1$

$$\begin{bmatrix}1\\2\\1\end{bmatrix} \begin{bmatrix}1\\4\\3\end{bmatrix} \begin{bmatrix}1\\3\\5\end{bmatrix} : \begin{bmatrix}-1\\-1\\-3\end{bmatrix} \begin{bmatrix}-1\\-5\\-6\end{bmatrix}$$

$w_1^t x_1 \rightarrow [1\ 1\ 1][1\ 2\ 1]^t > 0$

$w_1^t x_2 \rightarrow [1\ 1\ 1][1\ 4\ 3]^t > 0$

$w_1^t x_3 \rightarrow [1\ 1\ 1][1\ 3\ 5]^t > 0$

$w_1^t x_4 \rightarrow [1\ 1\ 1][-1\ -1\ -3]^t = -5 < 0$

Update w!!! $w_2 = [1\ 1\ 1] + [-1\ -1\ -3] = [0\ 0\ -2]$

$w_2^t x_5 \rightarrow [0\ 0\ -2][-1\ -5\ -6]^t > 0$

Perceptron – example (nonseparable scenario)

UNIVERSITÀ di VERONA

$w_2^t x_5$ ➔ $[0\ 0\ -2][-1\ -5\ -6]^t > 0$

Continue iteration for $w_2^t x_1$ ➔ $[0\ 0\ -2][1\ 2\ 1] < 0$

Update w again!!!

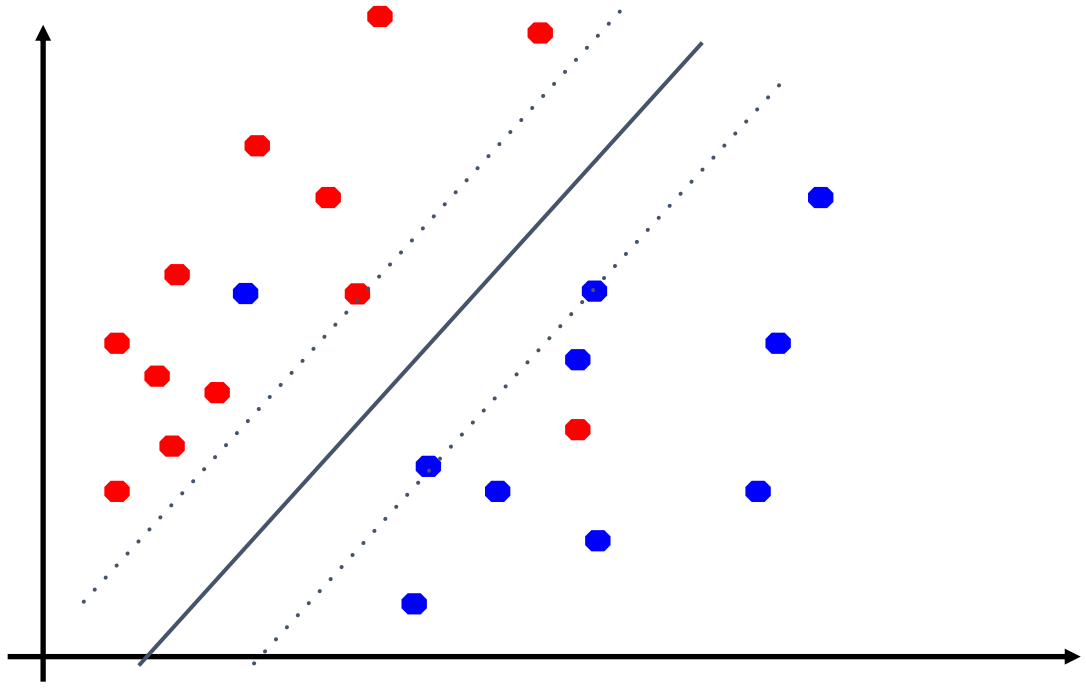
$w_{3=}$ $[0\ 0\ -2]+[1\ 2\ 1]=[1\ 2\ -1]$

……..

……..

We can continue this forever since there is no solution vector **w** satisfaying the perceptron rule.

- We want to learn something like this, but our Perceptron constraint do not allow!
- Solution: let's apply some relaxation



The Relaxation Method

- In this case the ***J(w)*** takes the form $J_q(\mathbf{w}) = \sum_{i \in X} \left(\mathbf{w}^t \mathbf{x}_i\right)^2$

- Also this function takes into account the misclassified samples and it is minimized when **w** is the solution vector.
- Its main difference is that its gradient is now continuous, whereas the gradient of the perceptron function *J* is not.
- So, such a function is suitable for finding smooth surfaces, but this may lead to problems.
  - In fact, in this way the points very far from the origin are weighted a lot.

Relaxation with margin

- Another option for ***J(w)*** is:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i \in X}\left\{\frac{\left(\mathbf{w}^t\mathbf{x}_i - b\right)^2}{\|\mathbf{x}_i\|^2}\right\}$$ where $b \geq 0$ margin (e.g., $b = 1$) and $\mathbf{w}^t\,\mathbf{x}_i \leq b$

- The gradient descent becomes: $\mathbf{w}_{k+1} = \mathbf{w}_k - \rho_k \sum_{i \in X}\left\{\frac{\left(\mathbf{w}^t\mathbf{x}_i - b\right)\mathbf{x}_i}{\|\mathbf{x}_i\|^2}\right\}$

with $\mathbf{x}_i$ that verify $\mathbf{w}^t\,\mathbf{x}_i \leq b$ and $\mathbf{w}$ is calculated with the usual rule.

$J(\mathbf{w}) > 0$ always, and $J(\mathbf{w}) = 0$ if $\mathbf{w}^t\,\mathbf{x}_i \geq b \;\; \forall \mathbf{x}_i$

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- The classical MSE criterion provides an alternative to the perceptron rule.
  - The perceptron rule only considers misclassified samples, since these are the only ones that violates $J(w)$
- Instead, the MSE criterion looks for a solution involving all of the samples (not just the misclassified ones).
  - The goal is to verify $w^t x_i = b_i$, where the b_i are some arbitrarily specified positive constants.

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- We have replaced the problem of finding the solution to a set of linear inequalities, we now have the problem of finding the solution of a set of linear equations.
  - **Perceptron:** $w^t x_i > 0$ for all samples x_i this solve system of linear inequalities.
  - **MSE:** $w^t x_i = b_i$ for all samples x_i solve system of linear equations.

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- MSE procedure
  - Choose positive constants $b_1, b_2, \ldots, b_n$
  - Try to find weight vector $\boldsymbol{w}$ such that $\boldsymbol{w^t x_i = b_i}$ for all samples x_i
  - If we can find weight vector $\boldsymbol{w}$ such that $\boldsymbol{w^t x_i = b_i}$ for all samples x_i, then w is a solution because b_i 's are all positive
  - Consider all the samples

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- For consistency, we will continue assuming that examples from class ***ω2*** have been replaced by their **negative vector**, although this is not a requirement for the MSE solution. Thus, we have,

$$x_{i,n+1} = 1 \text{ for } \omega_1 \text{ and } x_{i,n+1} = -1 \text{ for } \omega_2$$

- Suppose we have the following dataset:

1^{th} sample: $x_{1,1}w_1 + x_{1,2}w_2 + \ldots + x_{1,n}w_n + x_{1,n+1}w_{n+1} = b_1, \quad b_1 \geq 0$

2^{th} sample: $x_{2,1}w_1 + x_{2,2}w_2 + \ldots + x_{2,n}w_n + x_{2,n+1}w_{n+1} = b_2, \quad b_2 \geq 0$

……

N-th sample: $x_{N,1}w_1 + x_{N,2}w_2 + \ldots + x_{N,n}w_n + x_{N,n+1}w_{n+1} = b_N, \quad b_N \geq 0$

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

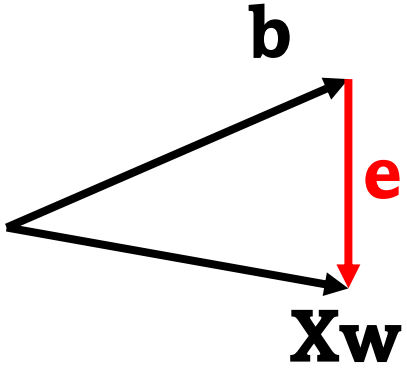
- We then obtain N equations (equal to the training samples, N_1 for ω_1 and N_2 for ω_2), with $N = N_1+N_2$.
- We need to solve N equations.
- In matrix notation we have: $\mathbf{X} \cdot \mathbf{w} = \mathbf{b}$ when

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} & \pm 1 \\ x_{2,1} & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & \vdots & \vdots \\ x_{N,1} & \cdots & \cdots & x_{N,n} & \pm 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ w_{n+1} \end{bmatrix}$$

- Thus, we need to solve a linear system: $\mathbf{X} \cdot \mathbf{w} = \mathbf{b}$

Minimum Squared Error (MSE) Solution

- Generally, N >> n+1, so there are more equations than unknowns.
  - $\mathbf{X} \cdot \mathbf{w} = \mathbf{b} \rightarrow \mathbf{w} = \mathbf{X}^{-1}\,\mathbf{b}$
- Let's now inject the error vector (needed to obtain loss function), defined as: $\mathbf{e} = \mathbf{X}\,\mathbf{w} - \mathbf{b}$ *(imagine this as b is the real class label and Xw is your predictions)*
- MSE finds *w*, which minimizes the length of the error vector **e**. Thus, minimize the minimum squared error criterion function J(w):



$$J_M(\mathbf{w}) = (\mathbf{X}\,\mathbf{w} - \mathbf{b})^t\,(\mathbf{X}\,\mathbf{w} - \mathbf{b}) = \|\,\mathbf{X}\,\mathbf{w} - \mathbf{b}\,\|^2$$

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- !!! Unlike the perceptron criterion function, we can optimize the MSE criterion function analytically by setting the gradient to zero.

- Now, let's calculate the gradient of the objective function $J_M(\mathbf{w})$:
- We need to find derivative of $J_M(\mathbf{w})$ wrt. to w.

- Given $J_M(\mathbf{w}) = (\mathbf{X}\,\mathbf{w} - \mathbf{b})^t\,(\mathbf{X}\,\mathbf{w} - \mathbf{b}) = \|\,\mathbf{X}\,\mathbf{w} - \mathbf{b}\,\|^2$ in matrix format,

we differentiate the expression $(\mathbf{X}\,\mathbf{w} - \mathbf{b})^t\,(\mathbf{X}\,\mathbf{w} - \mathbf{b})$ with respect to w.

Let's denote $Y = \mathbf{X}\,\mathbf{w} - \mathbf{b}$, **then** $J_M(\mathbf{w}) = Y^t\,Y$

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

Let's denote $Y = \mathbf{X}\,\mathbf{w} - \mathbf{b}$, **then** $J_M(\mathbf{w}) = Y^t\,Y$

Now, differentiating $J_M(\mathbf{w})$ wrt. $\mathbf{w}$, we have:

$$\frac{dJ_M(w)}{dw} = \frac{d}{dw}(Y^T Y)$$

$$= 2Y^T \frac{dY}{dw} \quad \rightarrow \quad \frac{dY}{dw} = \frac{d}{dw}(Xw - b) = X$$

Remember the chain rule, if not clear to you

Therefore, $$\frac{dJ_M(w)}{dw} = 2Y^T X = 2(Xw - b)^T X$$

Minimum Squared Error (MSE) Solution

$$\frac{dJ_M(w)}{dw} = 2Y^T X = 2(Xw - b)^T X$$

- By setting the gradient to zero, this becomes

$2\,(\mathbf{X}\,\mathbf{w} - \mathbf{b})^t\,\mathbf{X} = 0 \Rightarrow \mathbf{X}^t\,\mathbf{X}\,\mathbf{w} = \mathbf{X}^t\,\mathbf{b}$ //the proof can be found in the next slide

- If we multipy by $[\mathbf{X}^t\,\mathbf{X}]^{-1}$, we get $\mathbf{w} = [\mathbf{X}^t\,\mathbf{X}]^{-1}\,\mathbf{X}^t\,\mathbf{b} = \mathbf{X}^{\#}\,\mathbf{b}$
- where $\mathbf{X}^{\#} = [\mathbf{X}^t\,\mathbf{X}]^{-1}\,\mathbf{X}^t$ is the *pseudoinverse* of $\mathbf{X}$. **Pseudo-inverse solution**
- Consequently, $\mathbf{w} = \mathbf{X}^{\#}\,\mathbf{b}$ is the optimal solution according to the MSE method.

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

1 $$2(Xw - b)^T X = 0$$

2 Since we have a factor of 2 on the left side, divide both sides by 2:

$$(Xw - b)^T X = 0$$

3 The expression $(Xw - b)^T X = 0$ can be rewritten by expanding the transpose:

$$(Xw)^T X - b^T X = 0$$

4 We know that $(Xw)^T X = w^T X^T X$ (since $(AB)^T = B^T A^T$):

$$w^T X^T X - b^T X = 0$$

This equality holds if:

5 $$X^T Xw = X^T b$$

This is the desired result, so we have shown that:

$$2(Xw - b)^T X = 0 \Rightarrow X^T Xw = X^T b.$$

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- $\mathbf{w} = \mathbf{X}^{\#}\,\mathbf{b}$ shows that the MSE solution depends on the margin vector **b**.
  - Different choices of **b** lead to diferent properties of the solution.
- If **b** is arbitrarily fixed, there is no evidence that the MSE solution gives a separator vector in the case of linear separability.
- So, how to choose **b** in MSE procedure?
  - Good choice is $b_1 = b_2 = \ldots\ldots\ b_n = 1$. In this case MSE solution is identical to Fischer's linear discriminant solution.

Minimum Squared Error (MSE) Solution

UNIVERSITÀ di VERONA

- MSE pays too much attention to outlier examples!
- While $\mathbf{w} = \mathbf{X}^{\#}\,\mathbf{b}$ is the optimal solution according to the MSE method when $\mathbf{X}^{\#} = [\mathbf{X}^{t}\,\mathbf{X}]^{-1}\,\mathbf{X}^{t}$ is the *pseudoinverse* of $\mathbf{X}$,
  - $\mathbf{X}^{\#}$ is difficult to obtain since we need to make the inverse of n+1×n+1 matrix, expensive especially when X is a large matrix.
  - Having $\mathbf{X}^{t}\,\mathbf{X}$ is singular also problematic.
  - It is not an iterative process, indeed $\boldsymbol{w}$ is obtained directly though mathematical computations in **one step**.

Example

- Compute the perceptron and MSE solution for the dataset

X1=[(1,6), (7,2), (8,9), (9,9)]

X2= [(2,1),(2,2),(2,4),(7,1)]

Perceptron learning

- Assume ρ_k =0.1and an online update rule
- Assume w_1 = [0.1, 0.1, 0.1]

1) Normalize the dataset

$$\begin{bmatrix} 1 & 1 & 6 \\ 1 & 7 & 2 \\ 1 & 8 & 9 \\ 1 & 9 & 9 \\ -1 & -2 & -1 \\ -1 & -2 & -2 \\ -1 & -2 & -4 \\ -1 & -7 & -1 \end{bmatrix}$$

Example

2) Iterate through all the examples ($\mathbf{w}^t\mathbf{x}_i$) and update *w(k)*

Data point 1: $[0.1\ 0.1\ 0.1]^t$ [1 1 6] this is >0 so correctly classified therefore there is no update

Data point 2: $[0.1\ 0.1\ 0.1]^t$ [1 7 2] this is >0 so correctly classified therefore there is no update

……

Data point 5: $[0.1\ 0.1\ 0.1]^t$ [-1 -2 -1] this is <0 so we have to apply an update $w(2)=w(1)+ \rho_1$ [-1 -2 -1] = [0 -0.1 0]

……

Example

Data point 1: $[0\ -0.1\ 0]^t\ [1\ 1\ 6]$ this is <0 so update

$w(3)=w(2)+\rho_1\ [1\ 1\ 6] = [0\ -0.1\ 0]+0.1[1\ 1\ 6] = [0.1\ 0\ 0.6]$

Data point 2: $[0.1\ 0\ 0.6]^t\ [1\ 7\ 2] > 0$ ➔ no update

The perceptron with this data and setup converges after 175 iterations, when $w=[-3.5\ 0.3\ 0.7]$

EXAMPLE

MSE: This solution will be found in one shot $\mathbf{w} = [\mathbf{X}^t \mathbf{X}]^{-1} \mathbf{X}^t \mathbf{b}$

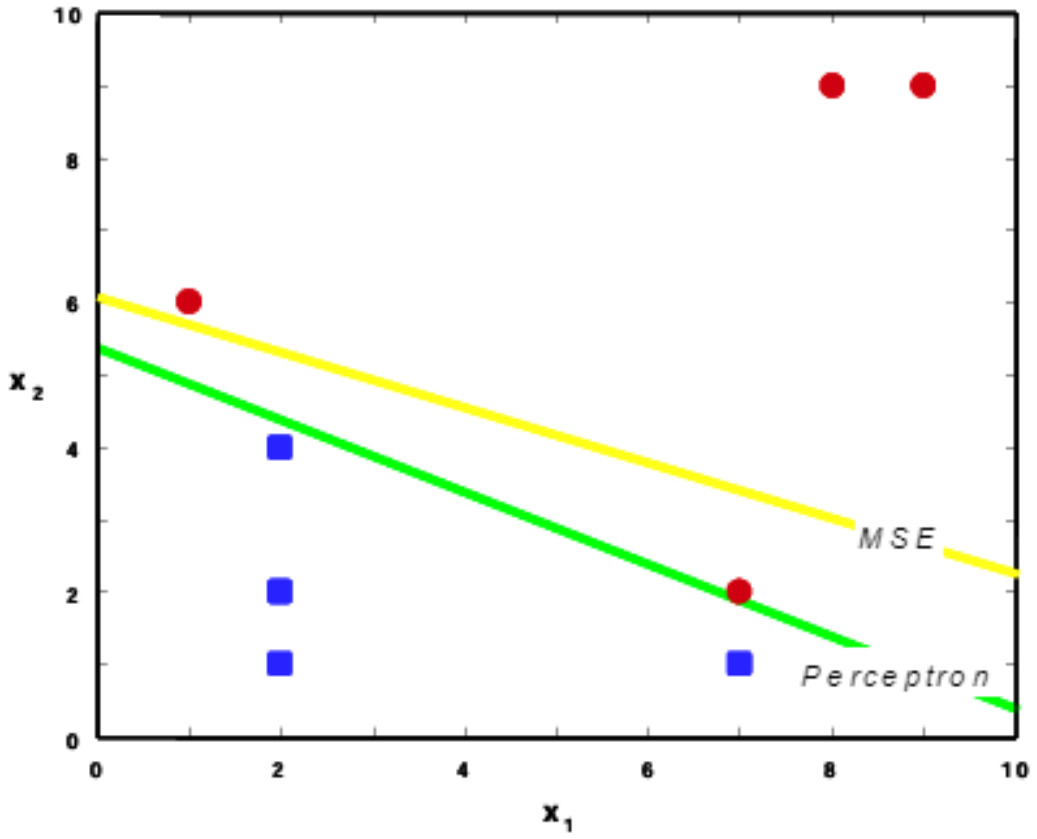
For the choice of b=[1 1 1 1 1 1 1 1], the solution is

w=[-1.1870 0.0746 0.1959]

X1=[(1,6), (7,2), (8,9), (9,9)]

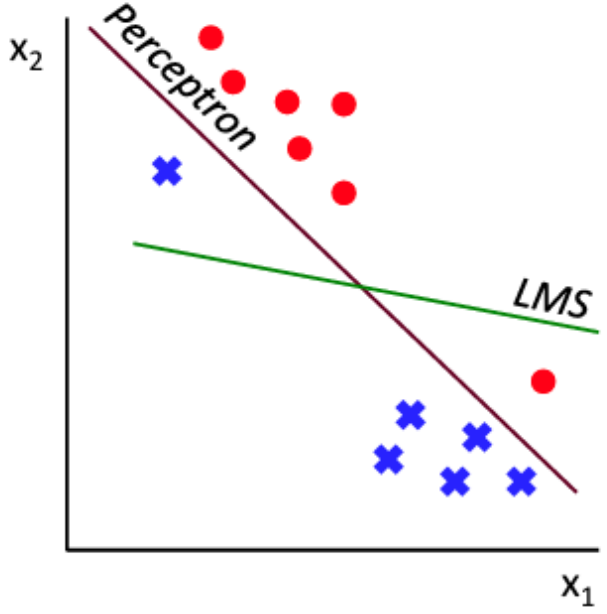
X2= [(2,1),(2,2),(2,4),(7,1)]

- As seen in Fig. The MSE misclassifies one of the samples



Perceptron vs. MSE

- **Perceptron rule –**The perceptron rule always finds a solution if the classes are linearly separable, but does not converge if the classes are non-separable
- **MSE criterion:** The MSE solution has guaranteed convergence, but it may not find a separating hyperplane if classes are linearly separable
  - Notice that MSE tries to minimize the sum of the squares of the distances of the training data to the separating hyperplane, as opposed to finding this hyperplane



The Least MSE (LMSE == Widrow-Hoff Method)

UNIVERSITÀ di VERONA

- The objective function $J_M(\mathbf{w})$ can also be minimized using a gradient descent procedure. In this way, we avoid
  - the problems that arise when $\mathbf{X^t X}$ is singular (*happens if samples are correlated*),
  - the need for working with large matrices.
- Since $\nabla J_M(\mathbf{w}) = 2\,\mathbf{X}^t(\mathbf{Xw} - \mathbf{b})$, the gradient descent algorithm is

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \rho_k\,\mathbf{X}^t(\mathbf{Xw}_k - \mathbf{b}) \quad \text{with } \mathbf{w}_1 \text{ arbitrary,}$$

- It can be shown that if $\rho_k = \rho_1/k$ where ρ_1 is any positive constant, this rule generates a sequence of weigth vectors that converge to a solution to $\mathbf{X}^t(\mathbf{X\,w} - \mathbf{b}) = 0$
  - The gradient descent algorithm leads to the solution regardless of whether $\mathbf{X^t X}$ is singular or not.

The Least MSE (LMSE == Widrow-Hoff Method)

UNIVERSITÀ di VERONA

- So, using this approach and considering one sample at a time, we get

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \rho_k (b_k - \mathbf{w}_k^t \mathbf{x}_k) \mathbf{x}_k$$ **LMS rule**

such that $\mathbf{b_k}$ is the value for the k-th sample of the margin $\mathbf{b}$ ($b_k \geq 0$), i.e., it depends on the sample $\mathbf{x_k}$.

- This method is called Widrow-Hoff, least-mean-squares (LMS) or delta rule reduces storage requirements by considering single samples sequentially.
- Note that Widrow-Hoff rule "corrects" $\mathbf{w}_k$ whenever $\mathbf{w}_k^t \mathbf{x}_k \neq b_k$.

The Least MSE (LMSE == Widrow-Hoff Method)

UNIVERSITÀ di VERONA

- In many cases, it is impossible to satisfy all the equations $\mathbf{w}_k^t \mathbf{x}_k = b_k$ and therefore the correction never ceases.
- Thus, ρ_k must decrease with k to achieve the convergence of the method.
- This is one of the most widely used techniques for linear classifiers.
- The disadvantage is that we have to iterate at least 3-4 times on all samples.

The Ho-Kashyap Method

UNIVERSITÀ di VERONA

- The main limitation of the MSE criterion is the lack of guarantees that a separating hyperplane will be found in the linearly separable case.

- All we can say about the MSE rule is that it minimizes
  $||\mathbf{X}\,\mathbf{w} - \mathbf{b}\,||^2$ when $\mathbf{b}$ is chosen arbitrarily.

This means that whether MSE finds a seperating hyperplane or nor dependes on how properly the margin $\mathbf{b}$ is selected (*notice that in some sources b is referred as target outputs*).

The Ho-Kashyap Method

- If the training samples are linearly separable, then there must be vectors w* and b* such that $Xw^* = b^* > 0$ where $b^* > 0$, meaning that every component of b* is **positive**.
- If b* were **known**, one could simply use MSE solution to compute separating hyperplane
- Idea: find both w* and b*
- Minimize the following criterion function, restricting to positive **b**

$$J_{HK}(\mathbf{w},\mathbf{b}) = \| \mathbf{X}\,\mathbf{w} - \mathbf{b} \|^2$$

- $J_{HK}(w^*, b^*) = 0$

The Ho-Kashyap Method

$$J_{HK}(\mathbf{w},\mathbf{b}) = \| \mathbf{X}\,\mathbf{w} - \mathbf{b} \|^2$$

- As usual, take partial derivatives with respect to **w** and **b**

$$\nabla_w J_{HK} = -2\,\mathbf{W}^{\mathbf{T}}(\mathbf{Xw} - \mathbf{b}) = 0$$

$$\nabla_b J_{HK} = -2\,(\mathbf{Xw} - \mathbf{b}) = 0$$

- Use gradient descent to find a minimum $J_{HK}(\mathbf{w},\mathbf{b})$
- Alternate the two steps below until convergence
  - Fix **b** and minimize $J_{HK}(\mathbf{w},\mathbf{b})$ with respect to **w**
  - Fix **w** and minimize $J_{HK}(\mathbf{w},\mathbf{b})$ with respect to **b**

The Ho-Kashyap Method

- The resulting iterative Ho-Kashyap rule is:

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \frac{1}{2}\rho_k[\nabla_b J_{HK}(\mathbf{w}_k,\mathbf{b}_k) - |\nabla_b J_{HK}(\mathbf{w}_k,\mathbf{b}_k)|]$$

$$\nabla_b J_{HK} = -2\,(\mathbf{Xw} - \mathbf{b}) = 0$$

Let $e_k = \mathbf{Xw}_k - \mathbf{b}_k = -\frac{1}{2}[\nabla_b J_{HK}(\mathbf{w}_k,\mathbf{b}_k)]$

Then $\mathbf{b}_{k+1} = \mathbf{b}_k - \frac{1}{2}\rho_k[-2ek \_ |2ek|]$

$$= \mathbf{b}_k + \rho_k[e_k + |e_k|]$$

The Ho-Kashyap Method

(0) Start with arbitrary $b_1 > 0$ and w_1, let k=1

Repeat steps (1) to (4)

(1) $e_k = \mathbf{X}\mathbf{w}_k - \mathbf{b}_k$

(2) Solve for $\mathbf{b}_{k+1}$ using $\mathbf{w}_k$ and $\mathbf{b}_k$

$$\mathbf{b}_{k+1} = \mathbf{b}_k + [e_k + |e_k|]$$

(3) Solve for $\mathbf{w}_{k+1}$ using $\mathbf{b}_{k+1}$

$$\mathbf{w}_{k+1} = (w^T w)^{-1} Y^T \mathbf{b}_{k+1}$$

(4) k=k+1

until $e_k = 0$ *or* $k > k_{max}$ or $\mathbf{b}_{k+1} = \mathbf{b}_k$ (for convergence fix learning rate as $0 < \rho_k < 1$)

The Ho-Kashyap Method

- In the linearly separable case,
  - $e_k = 0$, solution is found, STOP!
  - One of the components of e_k is positive, algorithm continues

- In non separable case,
  - e_k will have only negative components eventually, thus found proof of nonseperability.
  - No bound on how mainy iterations is needed for the proof of nonseperability.

Summary

- **Perceptron** Procedure
  - Find a separating hyperplane in the linearly separable case,
  - Do not converge in the non-separable case
  - Can force convergence by using a decreasing learning rate, but are not guaranteed a reasonable stopping point
- **MSE** Procedure
  - Converge in separable and not separable case
  - May not find separating hyperplane if classes are linearly separable
  - Use pseudoinverse if X^tX is not singular and not too large
  - Use gradient descent (Widrow-Hoff procedure) otherwise
- **Ho-Kashyap** Procedure
  - Always converge
  - Find separating hyperplane in the linearly separable case
  - More costly

Cigdem Beyan
cigdem.beyan@univr.it
https://cbeyan.github.io/