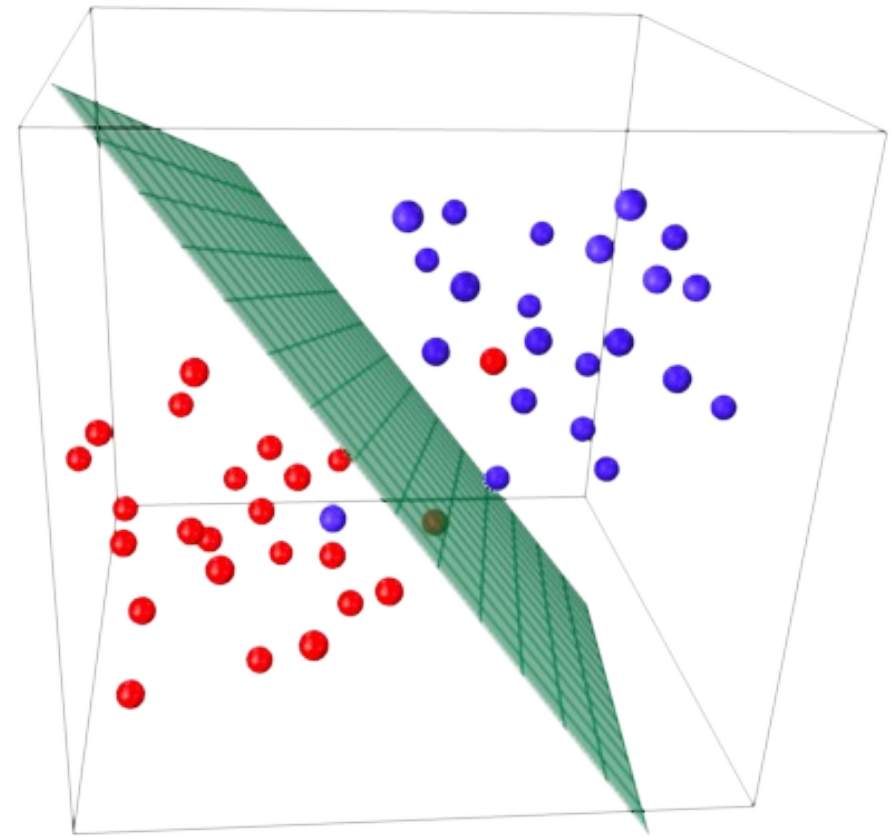


Linear Classifiers

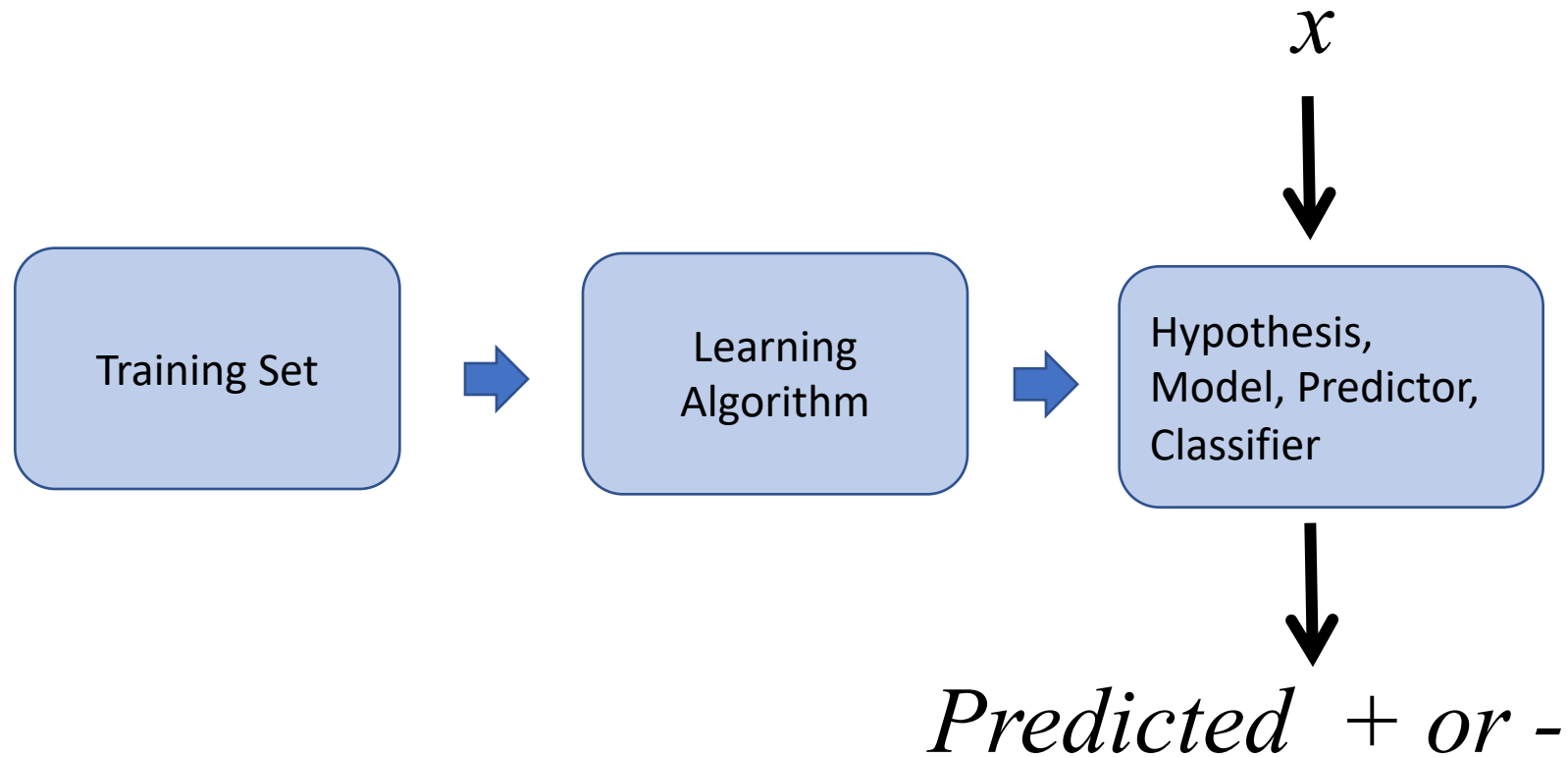
Ariel Shamir
Zohar Yakhini



Outline

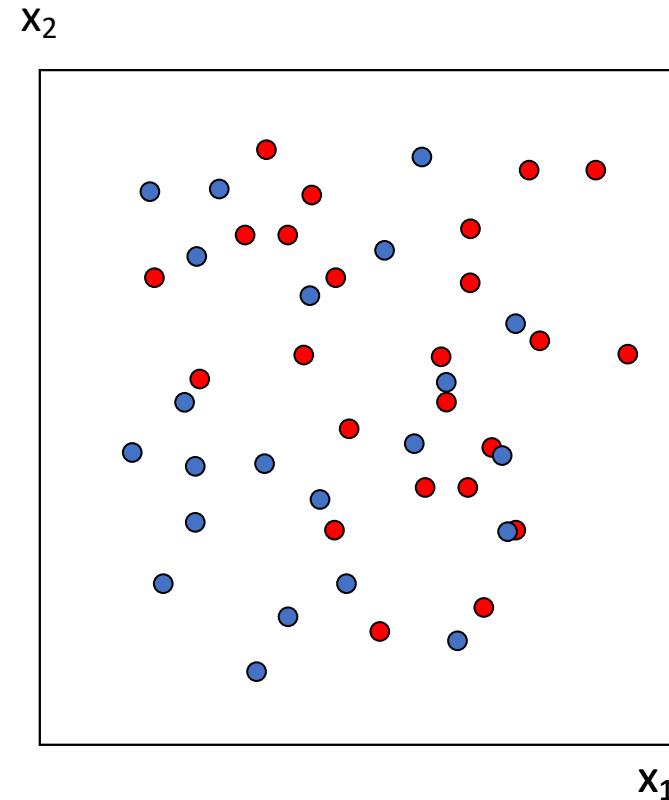
- Decision Functions
 - Linear Separability
 - Linear decision boundaries
 - The perceptron algorithm
 - Least Means Squares
 - Nonlinear mapping of features
 - Logistic regression
 - Mapping to higher dimension
- Coming up:
 - More examples of mapping to higher dimensions
 - Cover's Theorem
 - Dual perceptron
 - Kernels
 - SVMs

Classification



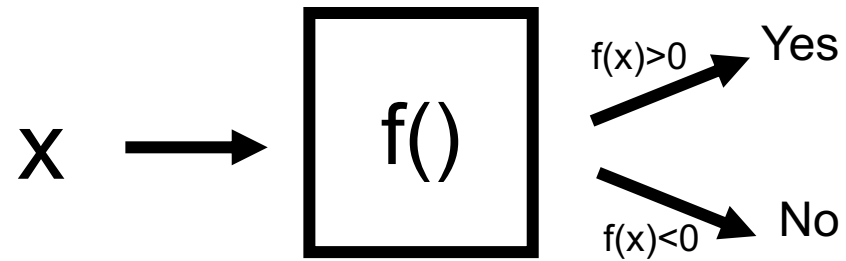
2D Feature Space

- Two numerical features x_1 , x_2 .
- Each instance is a 2D point
- Classification training set: each instance has a label (blue, red)
- Hypothesis can be defined as a function

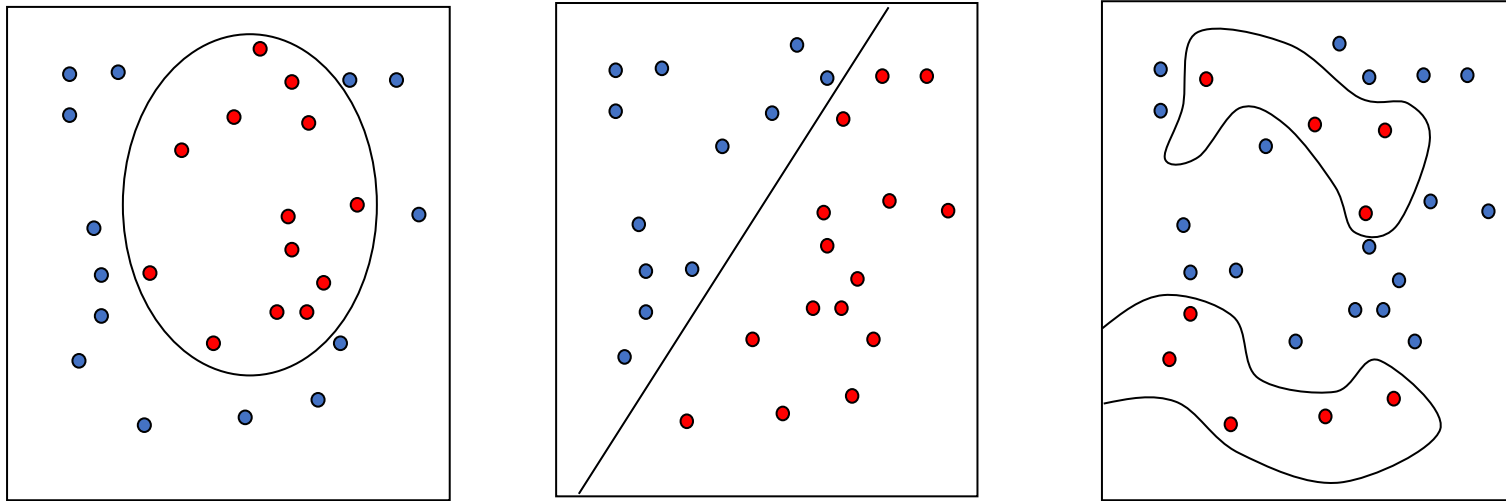


Discriminant Functions

- We define a scalar function $f:X \rightarrow \mathbb{R}$ from feature space to \mathbb{R}
- To classify an instance x , we check if $f(x) > 0$ or $f(x) < 0$



$f(x) = 0$ is the **decision boundary**



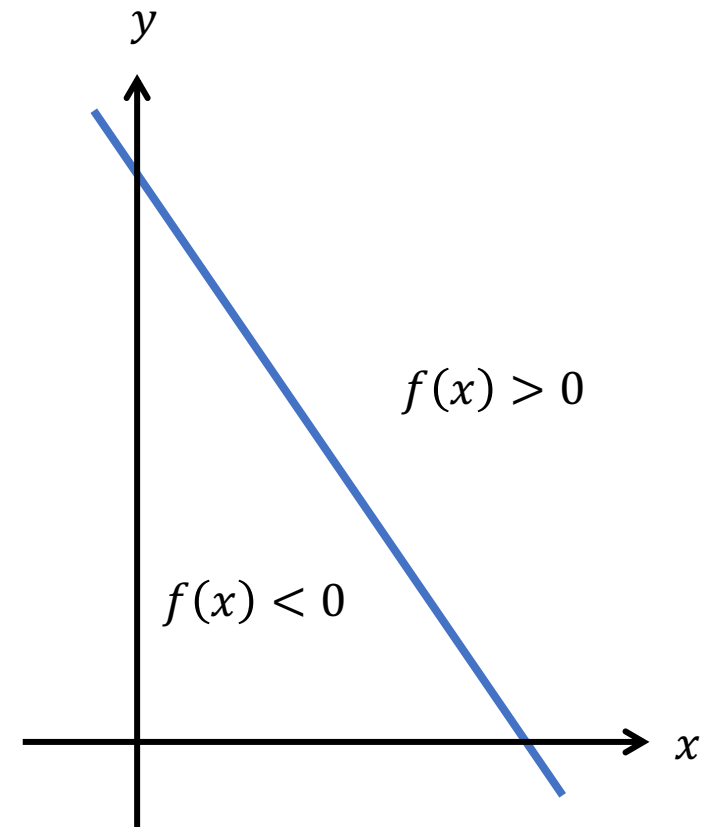
Key Decision: **which function?**

Simplest Function: Linear

- For example, Assume that our instance space is \mathbb{R}^2 (two features)
- Each instance is a point
- To represent a line in 2D we can use a linear implicit function in 2D

$$f(x, y) = w_1x + w_2y + w_0$$

- A line is defined by setting $f(x, y) = 0$



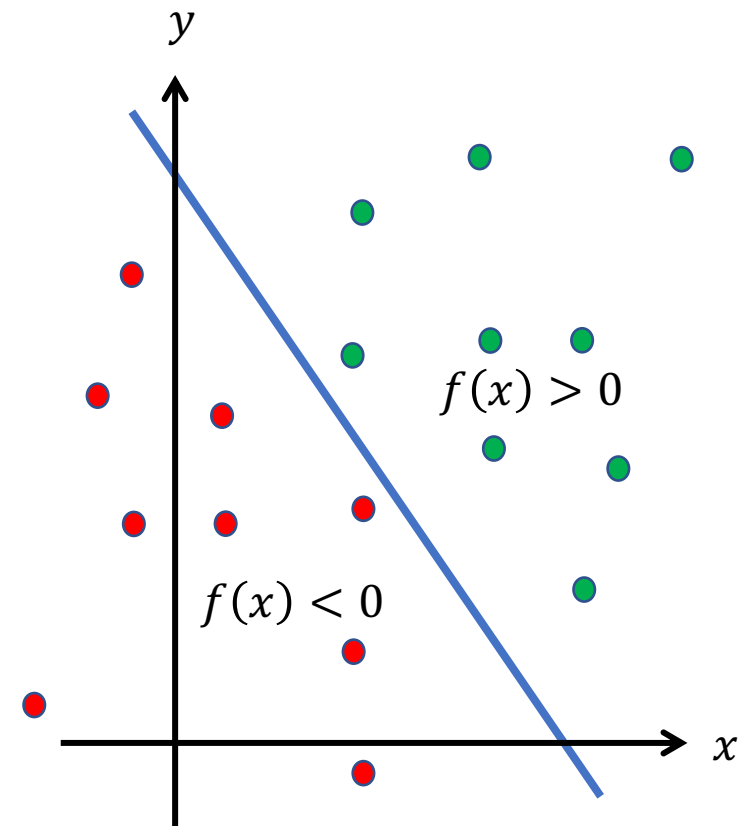
Simplest Function: Linear

- A dichotomy in \mathbb{R}^2 is separable by this line if:

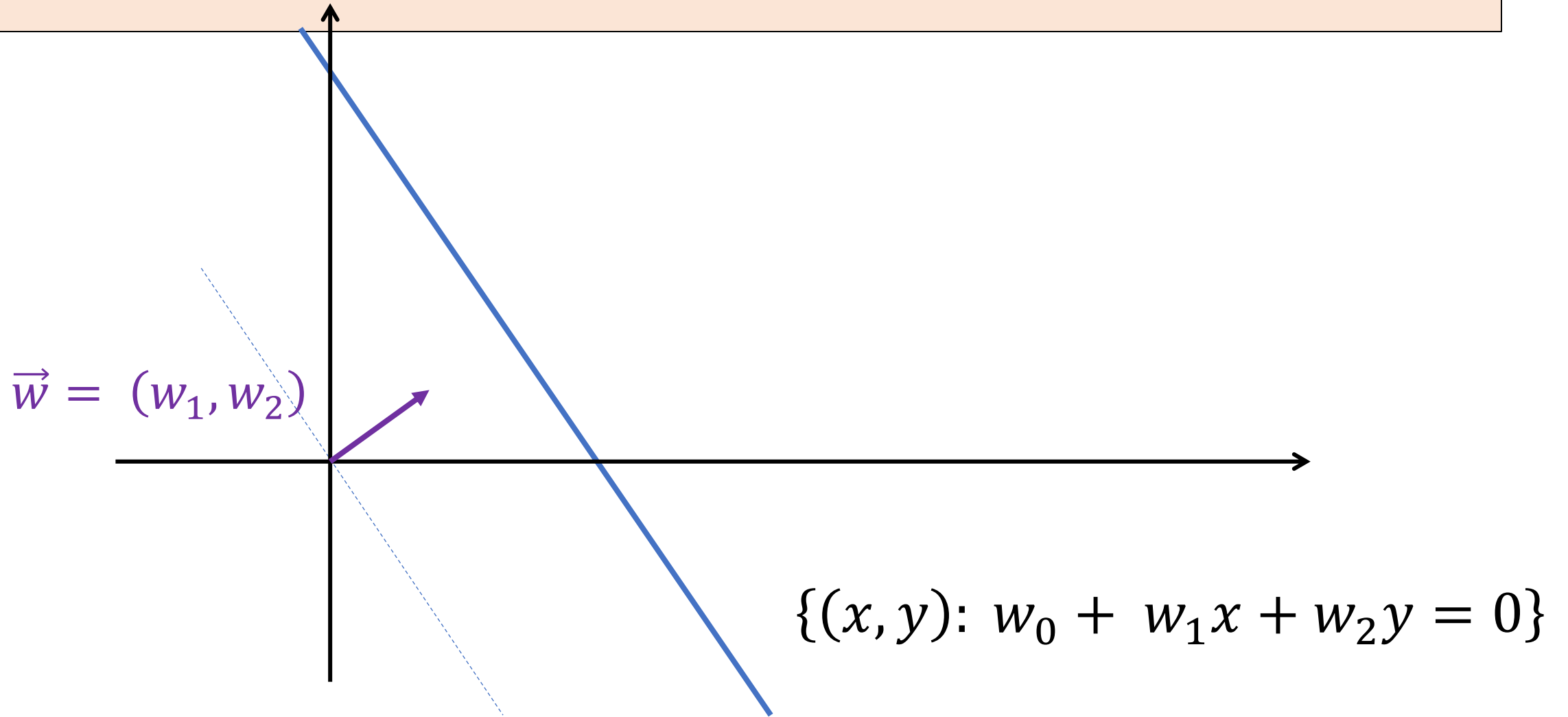
$$f(x, y) > 0 \Rightarrow +1,$$

$$f(x, y) < 0 \Rightarrow -1$$

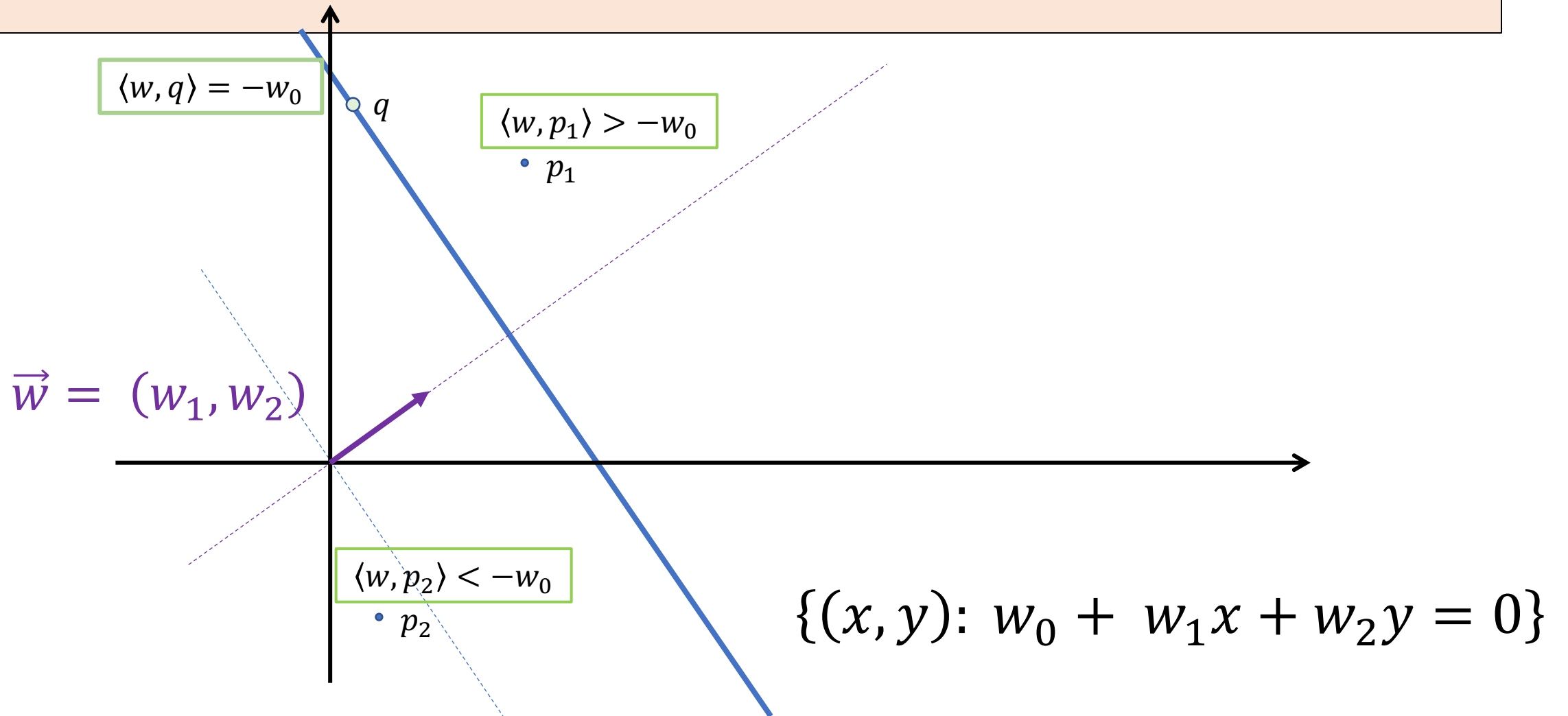
- This is called **Linear Separability**



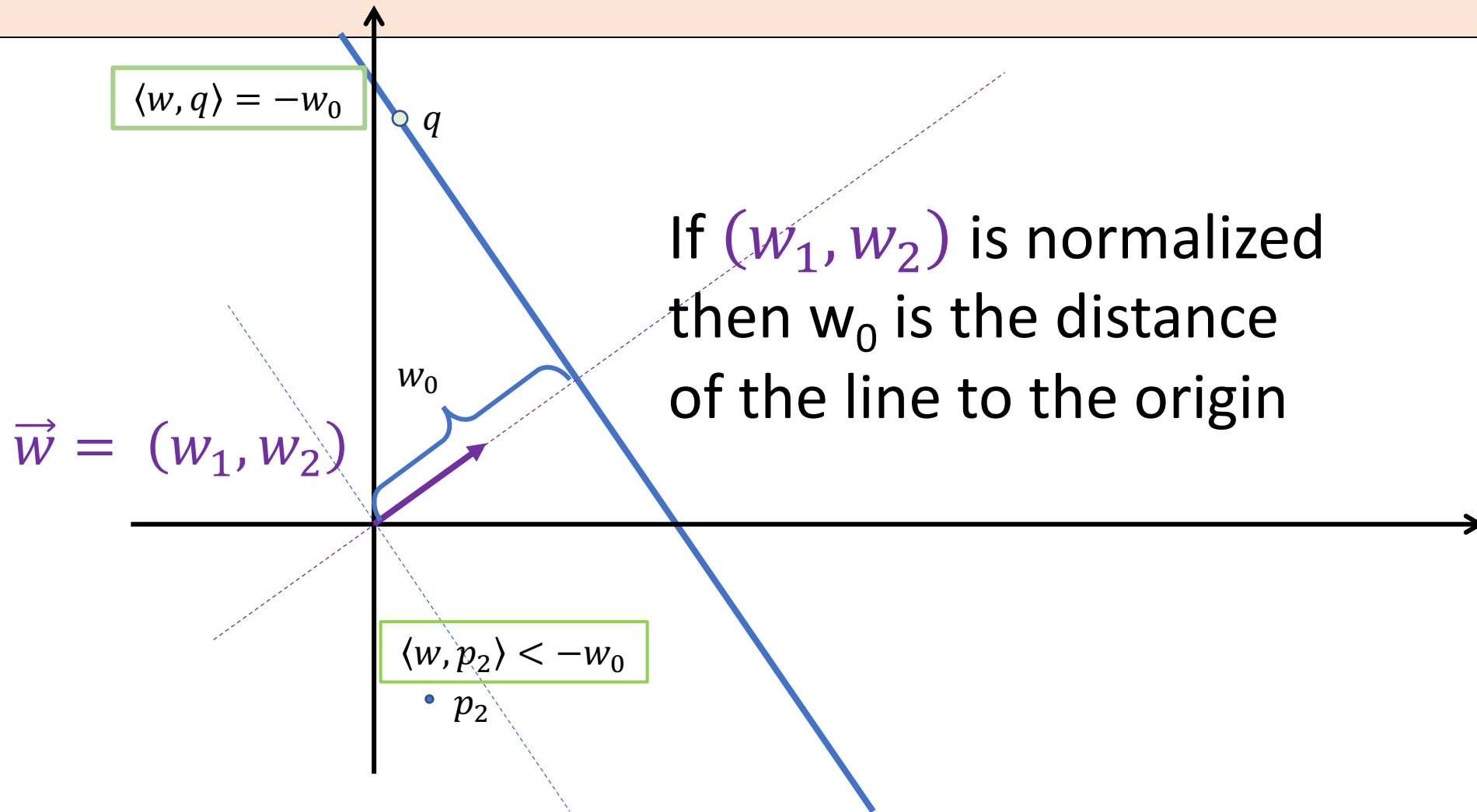
Geometric Interpretation



Geometric Interpretation



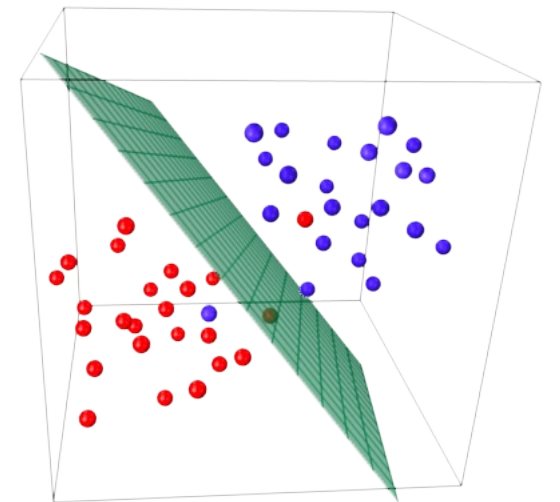
Distance from the origin to a hyperplane



Higher dimensions

A hyperplane is defined by the vector perpendicular to it (its normal vector) and a scalar w_0 (which determines the position):

$$U = \{\vec{x} \in \mathbb{R}^d : \vec{x} \cdot \vec{w} = w_0\}$$



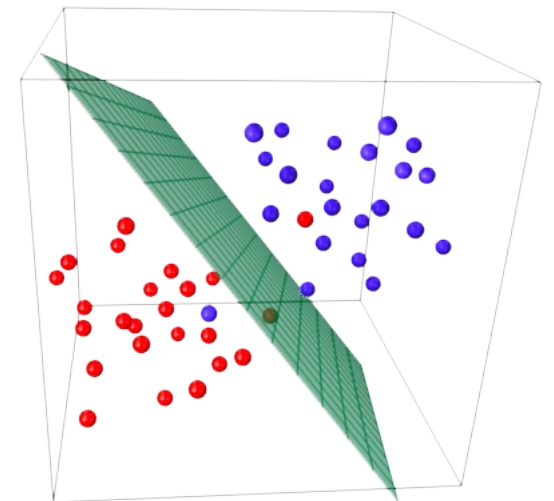
Linear Discriminant Function

Note: we assume
we have $x_0 = 1$

- **Linear** discriminant function

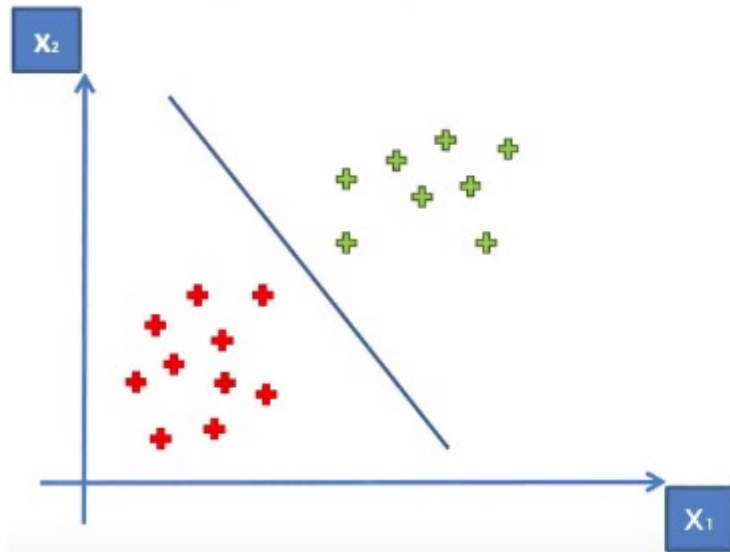
$$f(\vec{x}) = w_0 + w_1x_1 + \cdots + w_nx_n = \vec{w} \cdot \vec{x}$$

- Learning = find the weights \vec{w} that define the function
- The decision boundary is a hyperplane in n dimensions
- Will work if we have **linearly separable concepts**

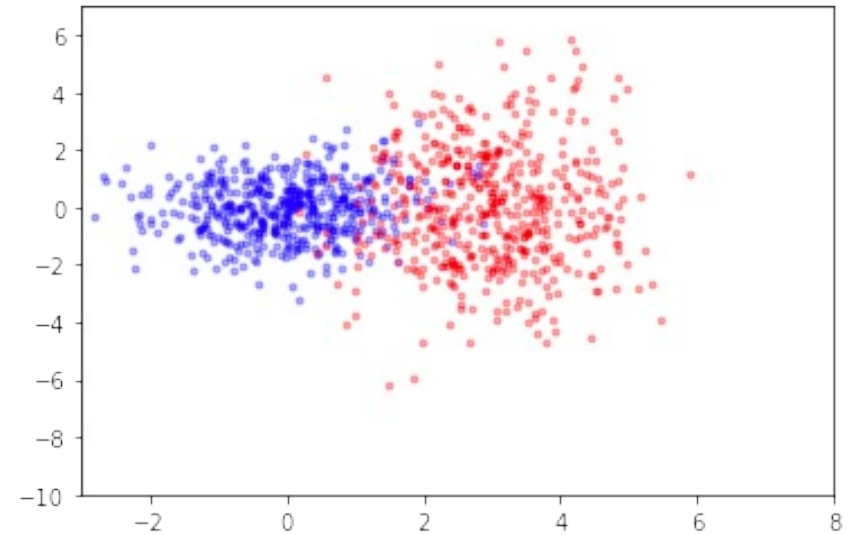
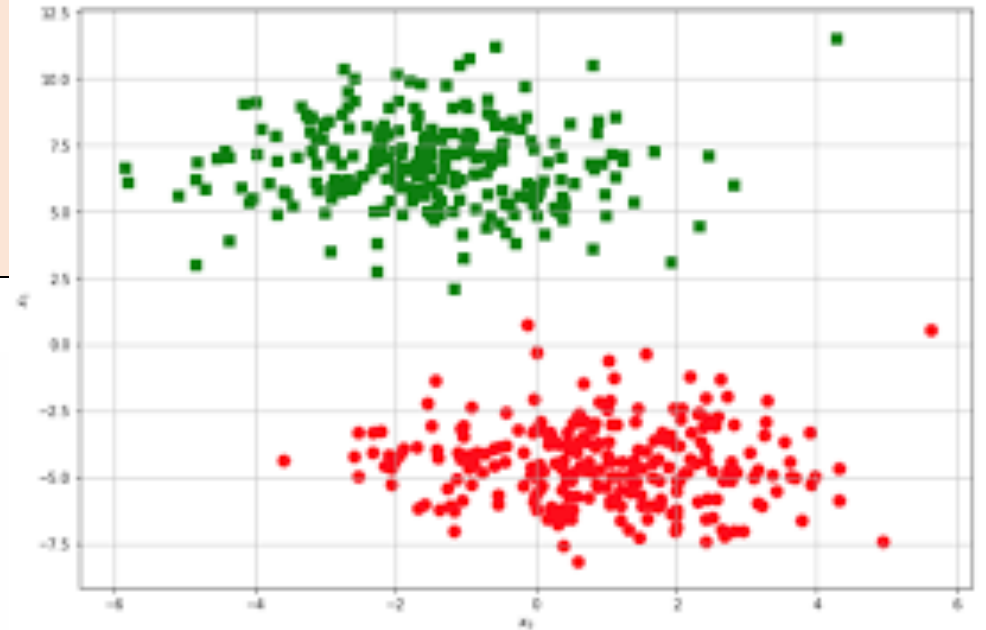
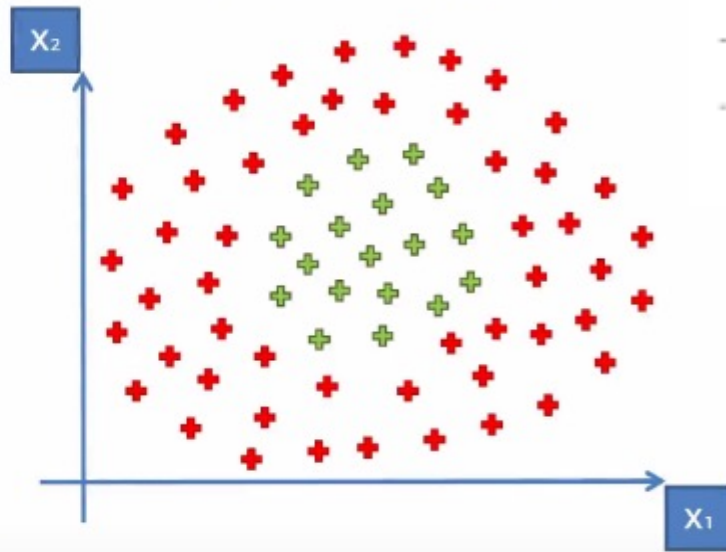


Examples

Linearly Separable



Not Linearly Separable

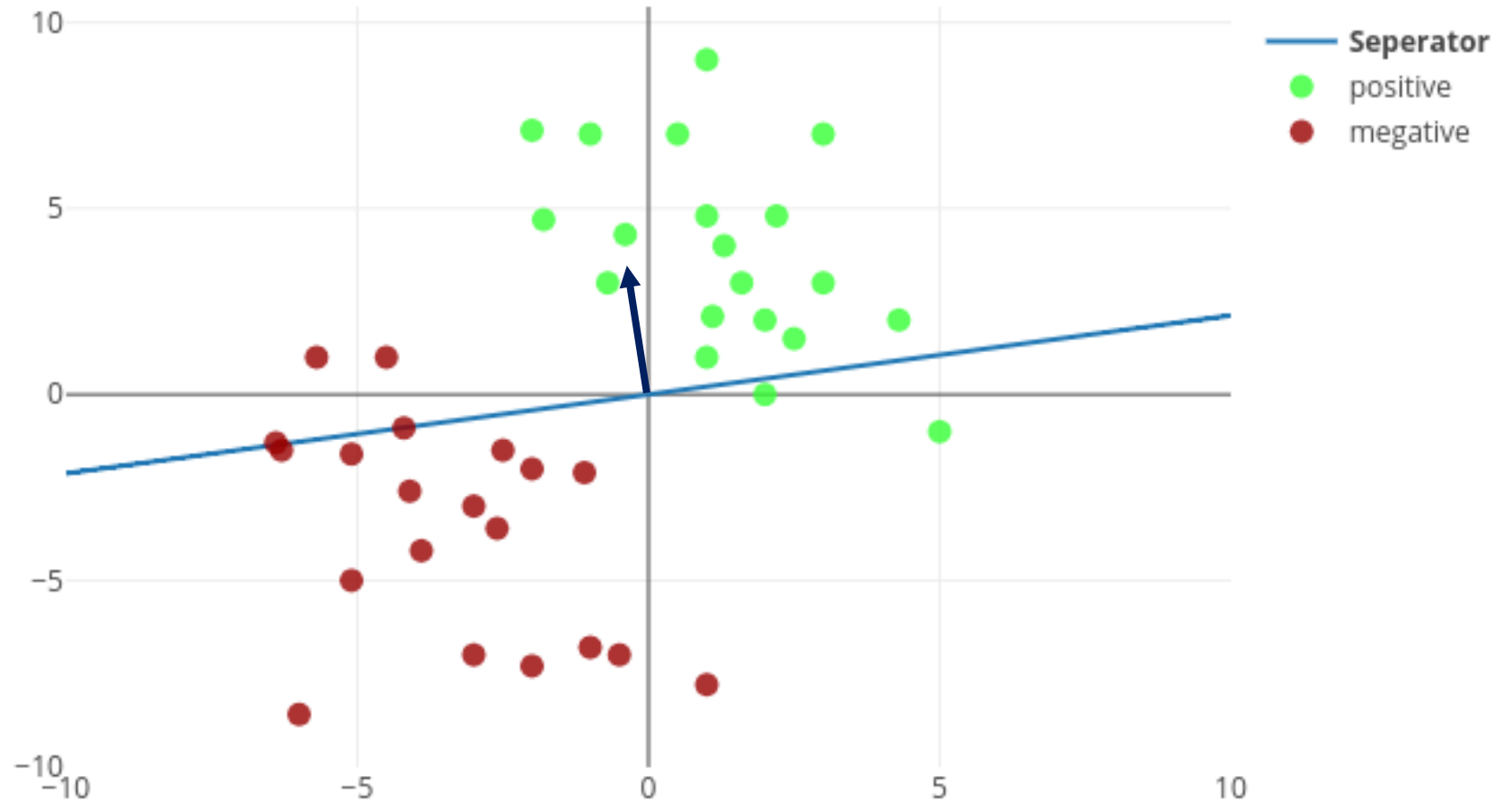


A hyperplane as a linear separator

Consider the decision function

$$h(\vec{x}) = \text{sgn}(\vec{x} \cdot \vec{w} + w_0)$$

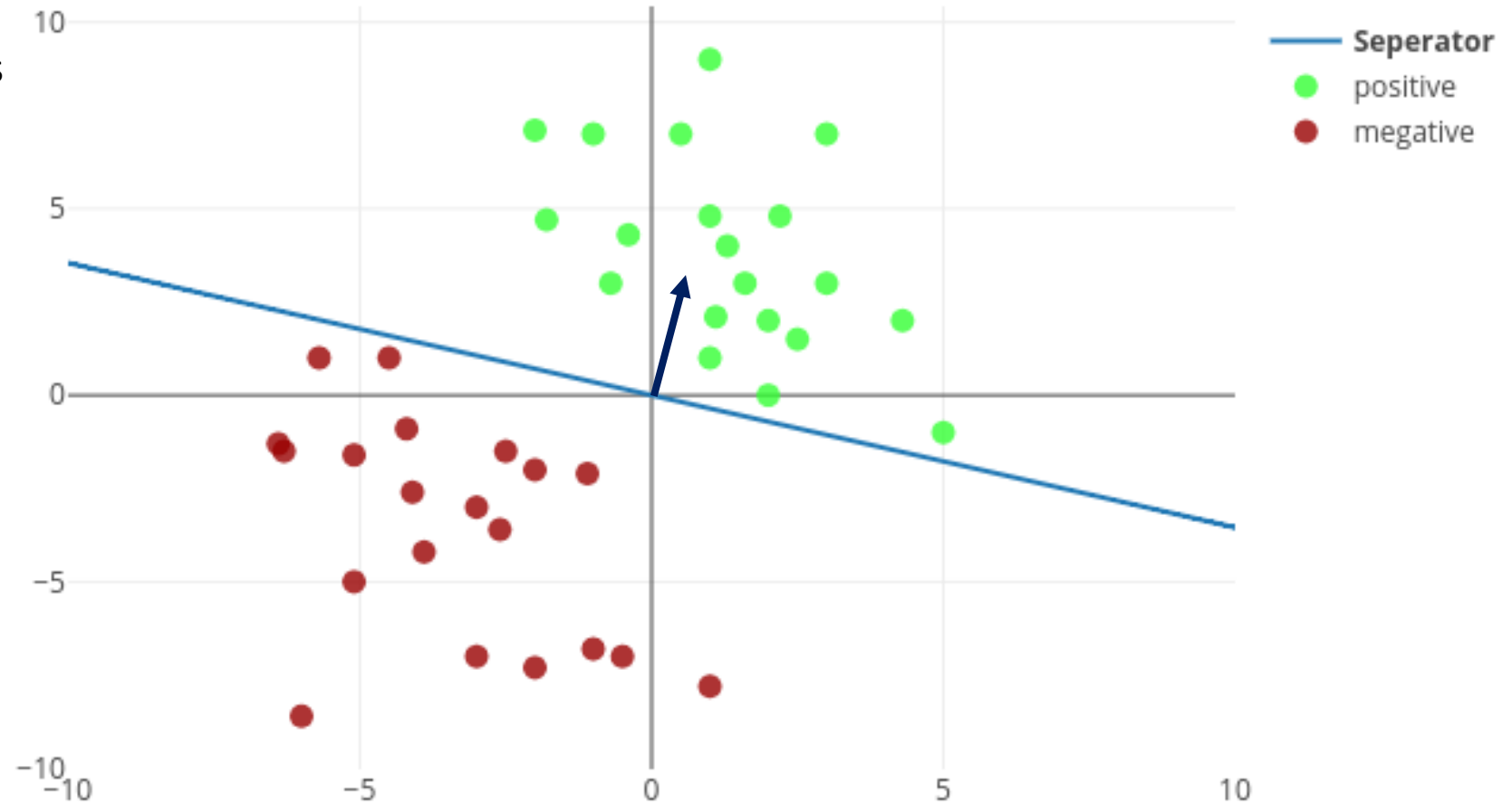
In this figure, who is w ?
what is b ? what values does
 h take for various points?



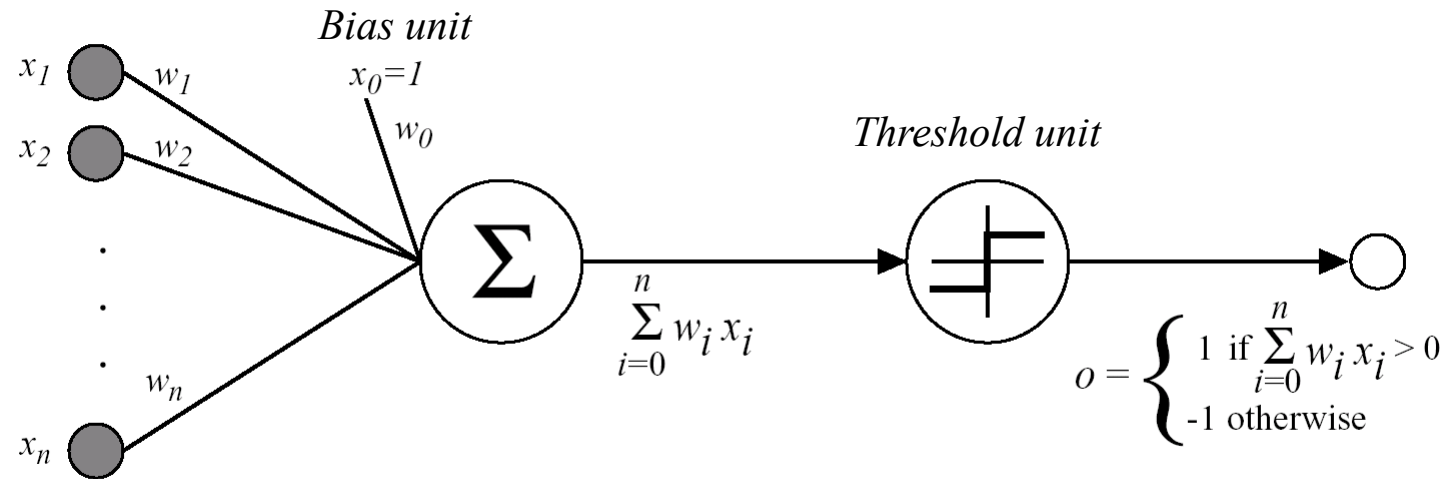
Consider the feature vectors \vec{x}_i , $i = 1 \dots m$,
and the matching labels y_i .

What can we say about the expressions

$$y_i(\vec{x}_i \cdot \vec{w} + w_0)$$



The Perceptron



$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

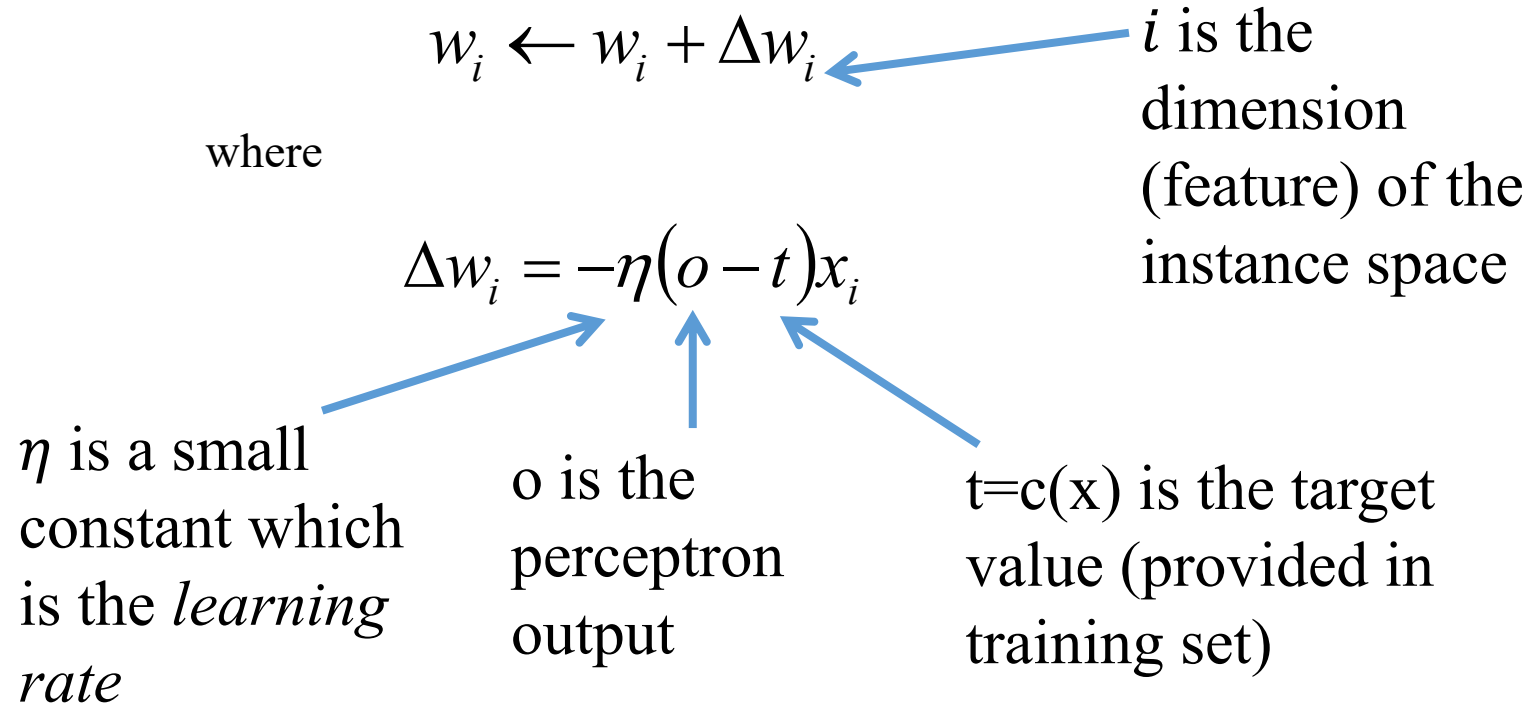
Or, in vector notations:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}\vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Learning Procedure

- Given a set of sample points and their classification (-1 or 1) we need to adjust the **weights** \vec{w} of the perceptron so that we will have the correct output for each sample.
- What is the simplest algorithm?
 - Start with random weights and improve
- How to improve?
 - Measure the error!
- What is the simplest error function?
 - $\text{output}(x) - \text{target}(x)$

Perceptron Training Rule



Updating the weights

$$\Delta w_i = -\eta(o - t)x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$

- If the instance x is classified correctly then no update!
- For any training instance x that is misclassified at the present iteration:
 - if $C(x) = +1$ we update the weights as:
$$w = w + 2\eta x$$
 - if $C(x) = -1$ we update the weights as:
$$w = w - 2\eta x$$

Justification of the Rule

$$\Delta w_i = -\eta x_i (o - t)$$

If $(o_i - t_i) = 0$ then Δw_i is 0 and there is no update

Otherwise: $w_i \leftarrow w_i + \Delta w_i$

o	t	o-t	x_i	Δw_i	$w_i \cdot x_i$
-1	+1	<0	>0	>0	increased
-1	+1	<0	<0	<0	increased
+1	-1	>0	>0	<0	decreased
+1	-1	>0	<0	>0	decreased

The Perceptron Learning Algorithm

- Assume the target value for classification are $t \in \{-1, +1\}$
- The perceptron seeks to find a linear separator with NO ERRORS.
- Is this always possible?

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in D compute

$$o_d = \text{sgn}(\vec{w} \cdot \vec{x}_d)$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta(o_d - t_d)x_{id}$$

$$w_i = w_i + \Delta w_i$$

n+1 weights to be updated
in the normal vector

Why does $E[\vec{w}]$ improve in every iteration?

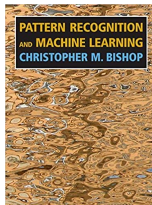
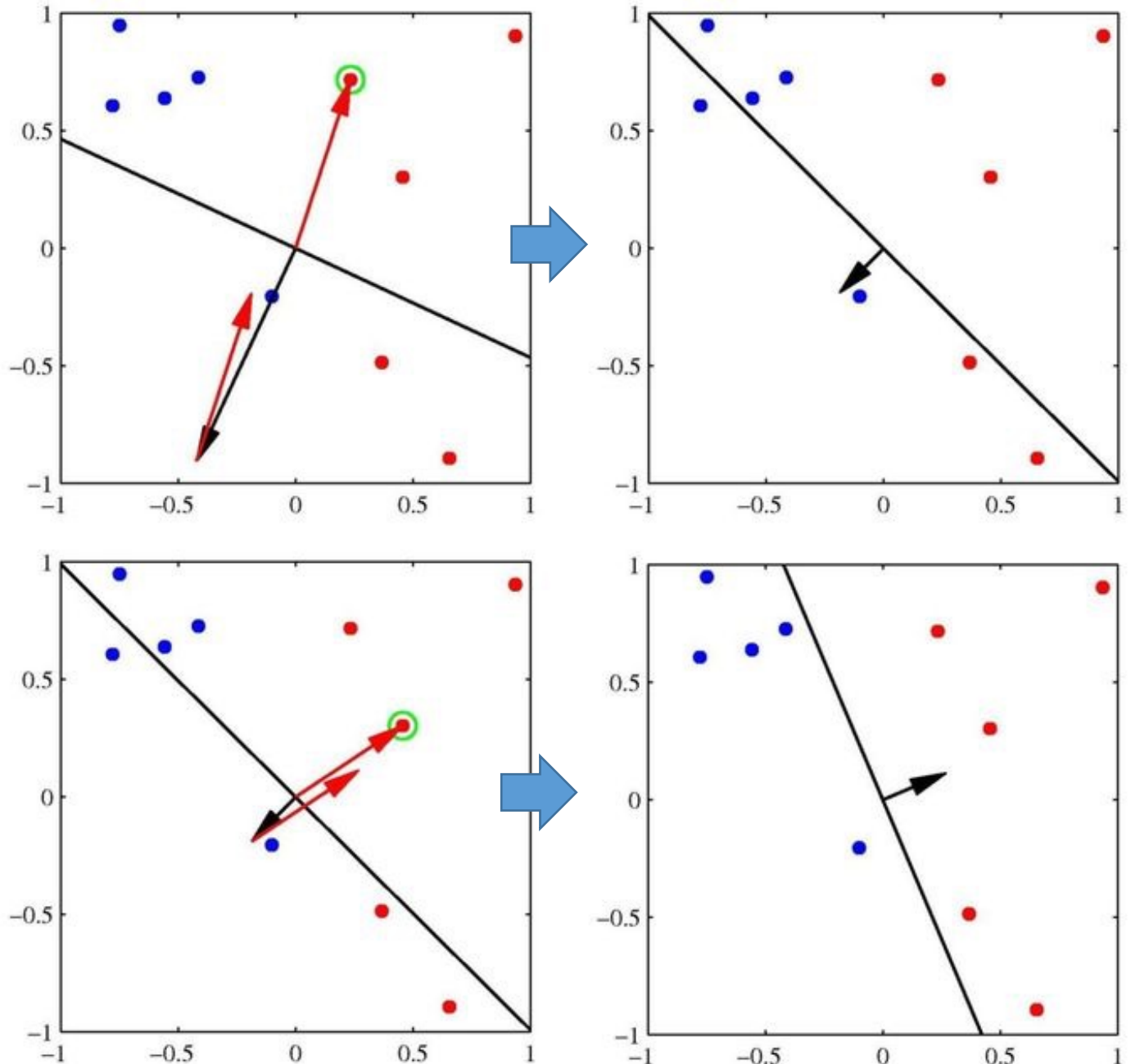
Red dots are positive ($t = +1$)
The marked one is initially misclassified

Recall that we then update by setting:

$$w^{(j+1)} = w^{(j)} + 2\eta x$$

We therefore get, for a misclassified x with $C(x) = +1$:

$$w^{(j+1)} \cdot x > w^{(j)} \cdot x$$



Figures from Bishop's book

Rosenblatt's Perceptron Theorem

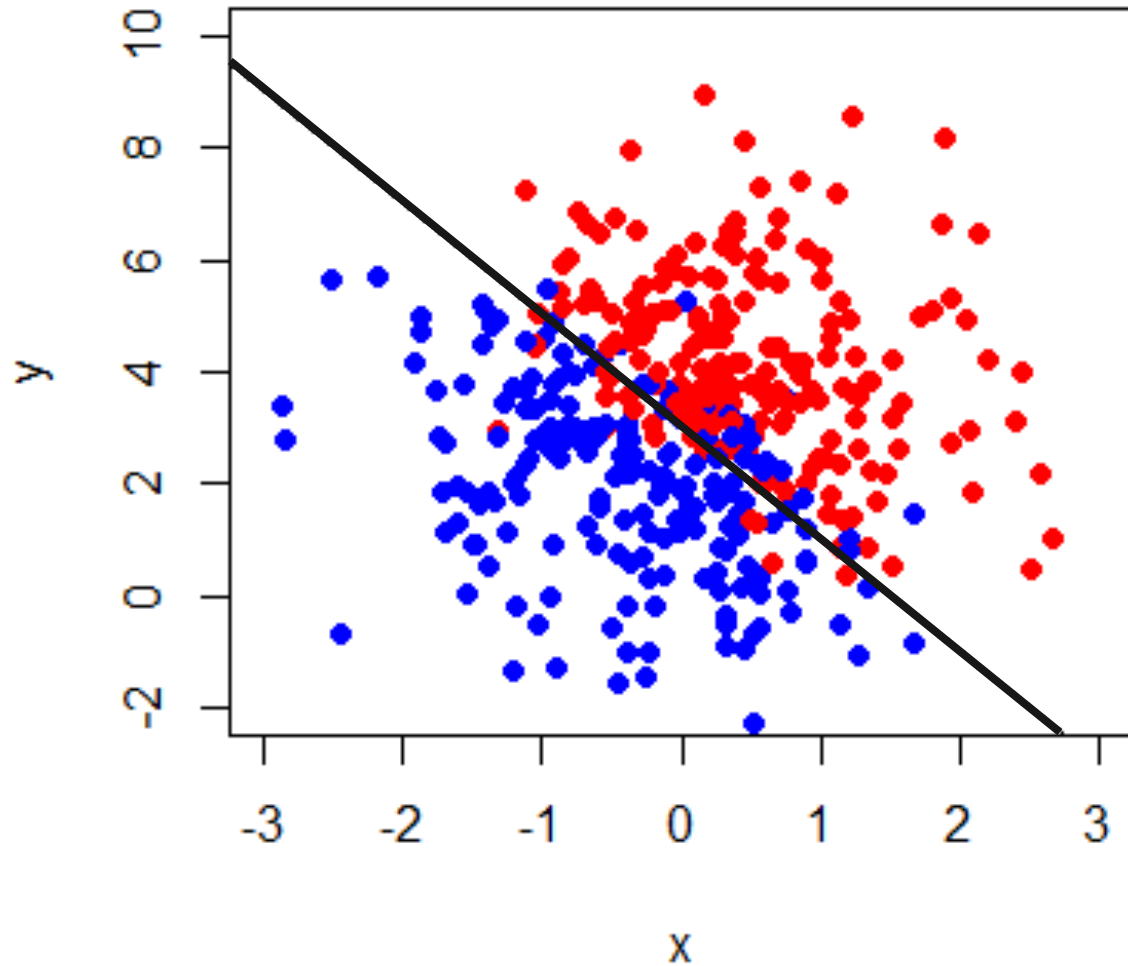
The Perceptron learning algorithm converges to a perfect classifier (no errors on the training data) iff the training data, D , is linearly separable.

- Note: we also need to control η to really guarantee convergence
(if its too big we may overshoot the perfect classifier)
- Some results on the rate of convergence were proven and can be useful in the context of ANNs (and deep learning)
- The Perceptron itself is not a practical learning approach but is an important component of many modern learning approaches.



Frank Rosenblatt
Cornell Univ, NY, US
1928-1971

However - data is not always linearly separable

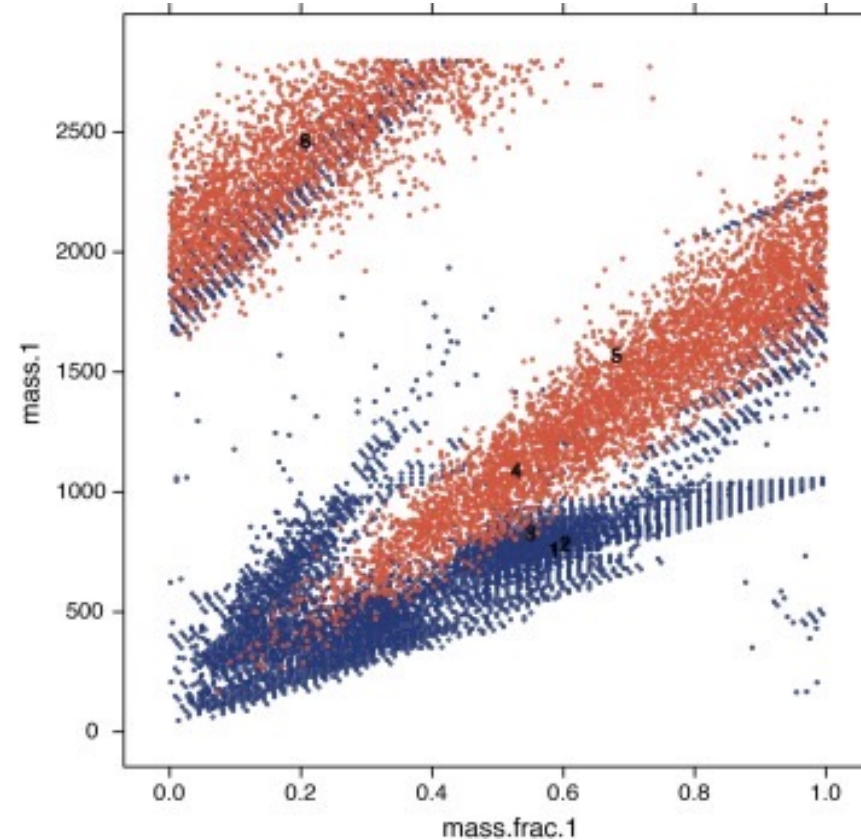


Data is not always linearly separable

London walking commuters



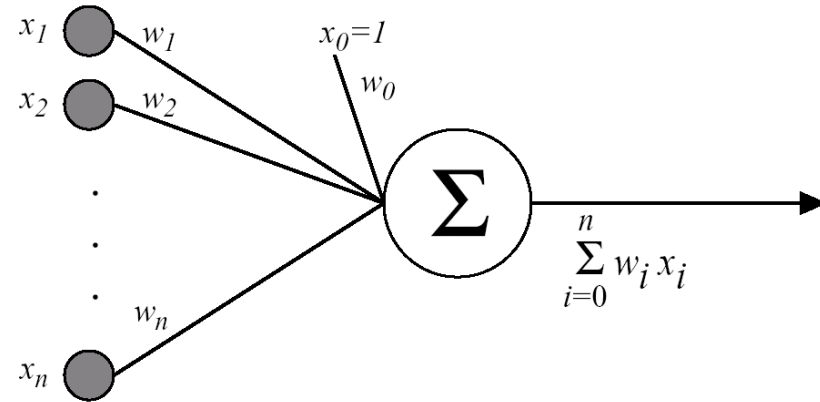
Lipids vs peptides (Dittwald et al)



What to do if the data is NOT linearly separable?

- Later... we map it to higher dimension where it is more likely to be linearly separable.
- Today... we still use a linear classifier that will be a (good?) approximation for the real discriminant function!
- We want to reduce the classification errors as much as possible.
- We can't use the perceptron rule since it may not converge. We are looking for a better rule.

A Linear Unit



Consider simple *linear unit* (with no threshold), where

$$o = w_0 + w_1 x_1 + \dots + w_n x_n = \vec{w} \cdot \vec{x}$$

Learning?

- What error function can we use?

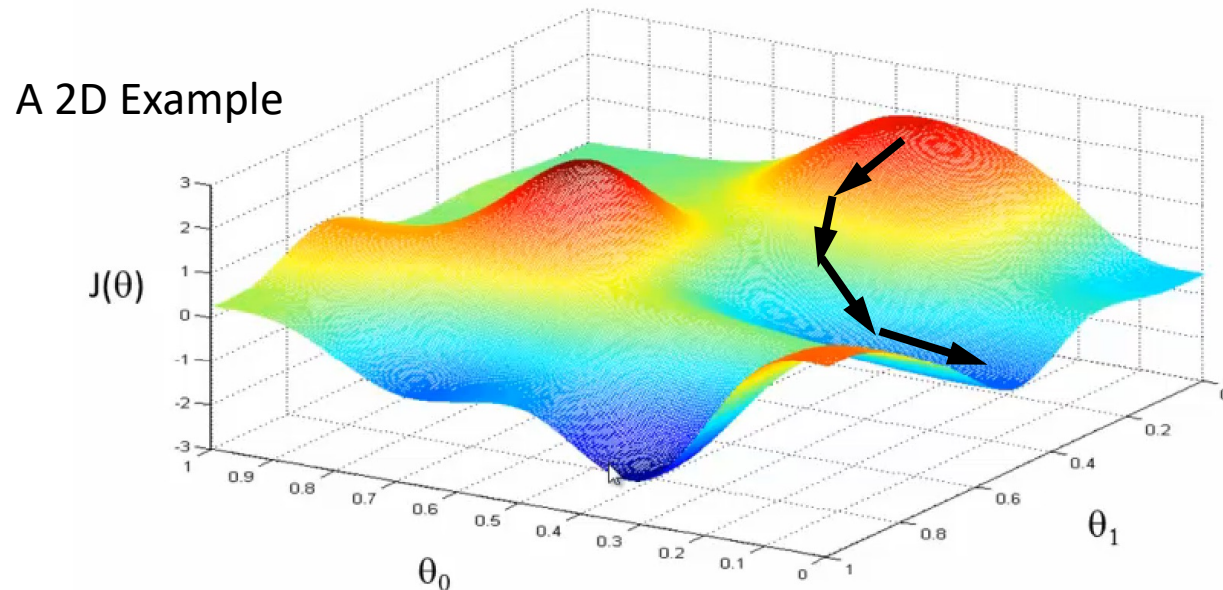
Let's try to learn w_i 's that minimize
the squared error over all sample points :

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (o_d - t_d)^2 = \frac{1}{2} \sum_{d \in D} (\vec{w} \cdot \vec{x}_d - t_d)^2$$

Where D is a set of training examples

Error in the Weights Space

- The error $E(w)$ is a function $E: \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ of the set of weights.
- Minimizing this function is like searching for a minimum on the functional high dimensional surface.
- For a smooth function, the direction of maximum increase/decrease at each point is the gradient/negative gradient:



Gradient Descent Rule

The gradient of E is $\nabla E[\vec{w}] = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right)$

Hence, the training rule will be $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$ or $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (\vec{w} \vec{x}_d - t_d)^2 = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (\vec{w} \vec{x}_d - t_d)^2 = \\ &= \frac{1}{2} \sum_{d \in D} 2(\vec{w} \vec{x}_d - t_d) \frac{\partial}{\partial w_i} (\vec{w} \vec{x}_d - t_d) = \sum_{d \in D} (o_d - t_d) x_{id} \end{aligned}$$

The training rule $\Delta \vec{w} = -\eta \nabla E[\vec{w}]$ becomes for each weight i :

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \sum_{d \in D} (o_d - t_d) x_{id}$$

LMS Algorithm

Input: η , a set of training examples D

η is the learning rate (e.g., 0.05)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where:

\vec{x} is the vector of input values, t is the target output value.

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in D compute

$$o_d = \vec{w} \cdot \vec{x}_d$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta \sum_{d \in D} (o_d - t_d) x_{id}$$

$$w_i = w_i + \Delta w_i$$

Reminder: Perceptron Algorithm

Input: η , a set of training examples D

η is the learning rate (e.g., 0.05)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where:

\vec{x} is the vector of input values, t is the target output value.

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in D compute

$$o_d = \text{sgn}(\vec{w} \cdot \vec{x}_d)$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta \sum_{d \in D} (o_d - t_d) x_{id}$$

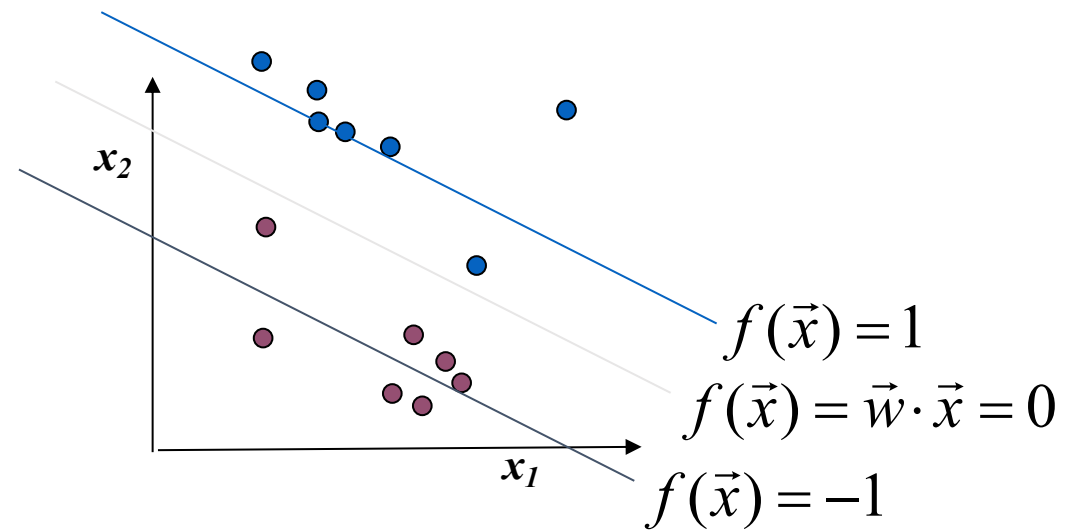
$$w_i = w_i + \Delta w_i$$

What Does LMS Minimize?

- The target numerical output of the linear unit on all training examples are their classification values (+1,-1)
- This means we will get a linear hyper-plane that **minimizes the sum of square distances** between the training samples and the **offset** (either +1 or -1) to this hyper-plane

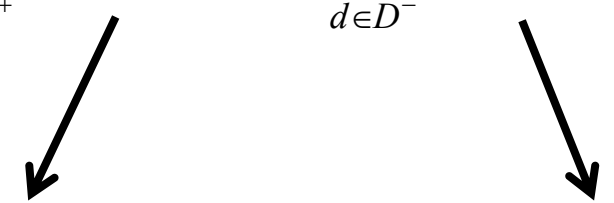
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (o_d - t_d)^2 = \frac{1}{2} \sum_{d \in D} (\vec{w} \cdot \vec{x}_d - t_d)^2$$

Minimize the SSE



35

LMS Problem

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \left[\sum_{d \in D^+} (\vec{w} \cdot \vec{x}_d - 1)^2 + \sum_{d \in D^-} (\vec{w} \cdot \vec{x}_d + 1)^2 \right]$$


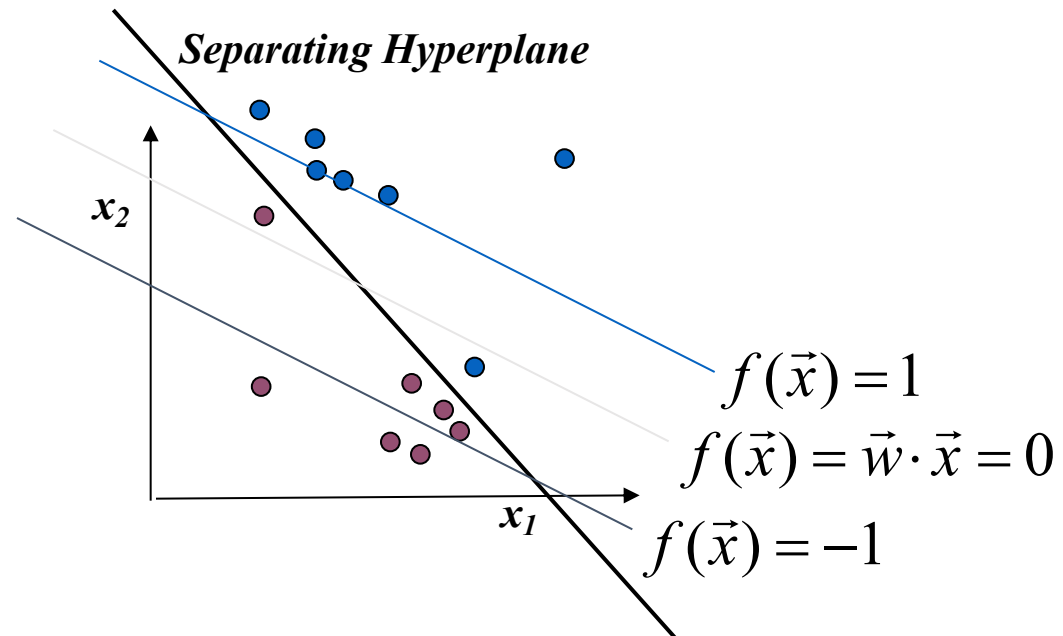
*Minimize the distance
between the positive
instances and the +1
iso-line of the function*

*Minimize the distance
between the negative
instances and the -1
iso-line of the function*

- This is different than minimizing the **number of misclassified samples**, and in fact may lead to some error even if the samples are linearly separable.

LMS Problem

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \left[\sum_{d \in D^+} (\vec{w} \cdot \vec{x}_d - 1)^2 + \sum_{d \in D^-} (\vec{w} \cdot \vec{x}_d + 1)^2 \right]$$



Perceptron

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in D compute

$$o_d = \text{sgn}(\vec{w} \cdot \vec{x}_d)$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta \sum_{d \in D} (o_d - t_d) x_{id}$$

$$w_i = w_i + \Delta w_i$$

- Optimizes number of errors
- Finds the separating plane if class is linearly separable.
- If not, it may not converge!

LMS

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in D compute

$$o_d = \vec{w} \cdot \vec{x}_d$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta \sum_{d \in D} (o_d - t_d) x_{id}$$

$$w_i = w_i + \Delta w_i$$

- Optimizes the distance to +1 and -1 lines

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \left[\sum_{d \in D^+} (\vec{w} \cdot \vec{x}_d - 1)^2 + \sum_{d \in D^-} (\vec{w} \cdot \vec{x}_d + 1)^2 \right]$$

- Uses gradient descent

LMS Algorithm (Stochastic Gradient Descent)

Input: η , a set of training examples D

η is the learning rate (e.g., 0.05)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where:

\vec{x} is the vector of input values, t is the target output value.

Initialize each w_i to some small random number.

Until termination do

For each \vec{x}_d in S compute

$$o_d = \vec{w} \cdot \vec{x}_d$$

For each linear unit weight w_i , Do

$$\Delta w_i = -\eta \sum_{d \in S} (o_d - t_d) x_{id}$$

$$w_i = w_i + \Delta w_i$$

Choose a random subset $S \subset D$

Linear Classification Revisited

- Perceptron used $\text{sgn}(\vec{w} \cdot \vec{x}_d)$ to choose between -1 or 1, but cannot use gradient descent
- LMS solution using $(\vec{w} \cdot \vec{x}_d)$ had problems as it did not find the global solution and was optimizing to offsets of +1 and -1
- We want to use **probabilistic** formulation
- We also look for a continuous domain that will represent the sign function so that we would be able to derive and use gradient descent algorithm

Logistic Regression

- A classification algorithm.
- Yields a family of linear separators
- The training data, in the binary case, consists of feature vectors and associated labels: $\{(\vec{x}^{(i)}, y^{(i)})\}_{i=1..m}$
- We learn a predictor hypothesis $h(\vec{x})$ $h: \mathbb{R}^k \rightarrow \mathbb{R}$
- Smoothly generalizes to any finite number of classes
- LoR is particularly important because it is a basic building block of artificial neural networks and of deep learning.

Interpretation as probabilities

- In LoR we want the values of h to be interpreted as probabilities.
- This means that $0 \leq h(\vec{x}) \leq 1$.
- We will also interpret values closer to 1 as probably coming from C1
and numbers closer to 0 as probably coming from C0.

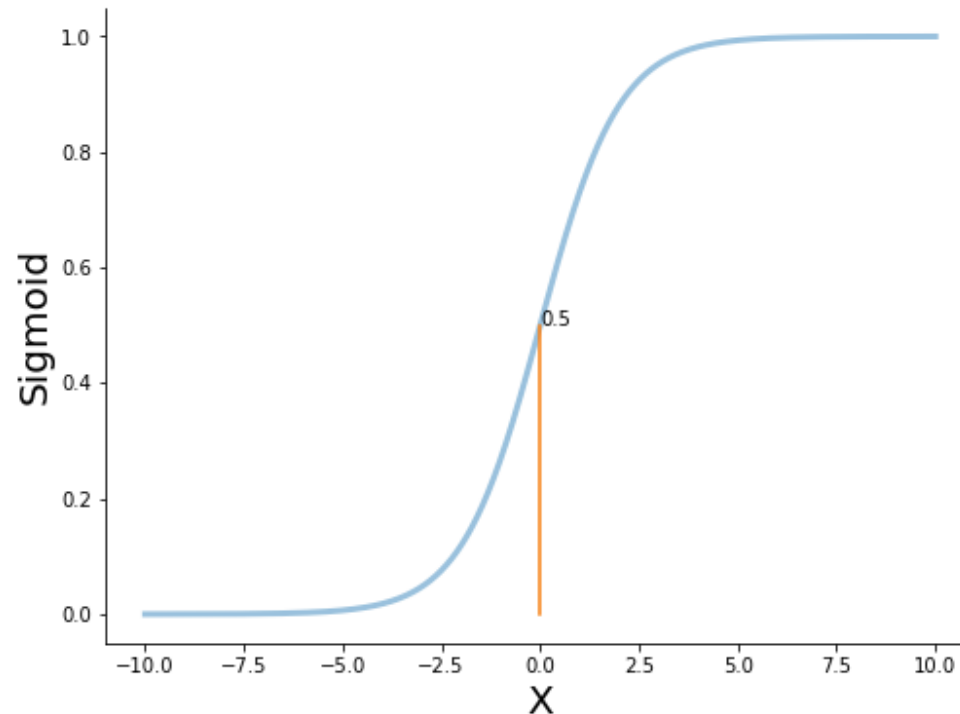
The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Some properties:

- Smooth
- Monotone
- Range? Limits?
- Derivative:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Derivative of Sigmoid

$$\begin{aligned}\frac{\partial}{\partial x} \sigma(x) &= \frac{\partial}{\partial x} \frac{1}{(1 + e^{-x})} = \frac{\partial}{\partial x} (1 + e^{-x})^{-1} = \\ &= -(1 + e^{-x})^{-2} \frac{\partial}{\partial x} e^{-x} = \\ &= -(1 + e^{-x})^{-2} e^{-x} (-1) = \\ &= \frac{1}{(1 + e^{-x})^2} = \\ &= \frac{1}{(1 + e^{-x})} \frac{1}{1 + e^{-x}} = \\ &= \frac{1}{(1 + e^{-x})} \left(1 - \frac{1}{1 + e^{-x}} \right) = \sigma(x)(1 - \sigma(x))\end{aligned}$$

Our Central Assumption

- $P(X | Y)$ is a sigmoid function applied to some linear combination of the input features.
- Mathematically, logistic regression assumes a vector $\vec{w} \in \mathbb{R}^{k+1}$ so that for points $\vec{x} \in \mathbb{R}^k$ we have:

$$P(X = \vec{x} | Y = 1) = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}} = \frac{e^{w^T x}}{1 + e^{w^T x}}$$

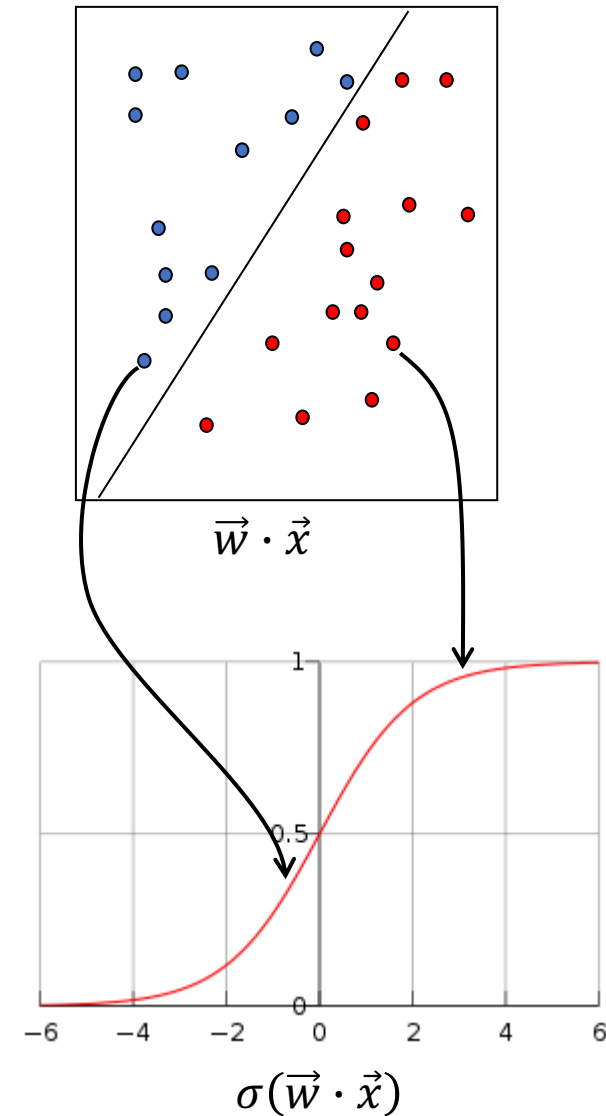
Where $w^T x = w_0 + \sum_{i=1}^k w_i x_i$

ML Classifier

- Now, assuming priors are the same (or we don't know them), using Bayes we get:
- $P(y = 1|x) = P(x|y = 1) = h_w(\vec{x})$
- $P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_w(\vec{x})$
- So:
 - Choose 1 if $P(y = 1|\vec{x}) > P(y = 0|\vec{x})$ else choose 0 \rightarrow
 - Choose 1 if $h_w(\vec{x}) > 1 - h_w(\vec{x})$ else choose 0 \rightarrow
 - Choose 1 if $h_w(\vec{x}) > 0.5$ else choose 0

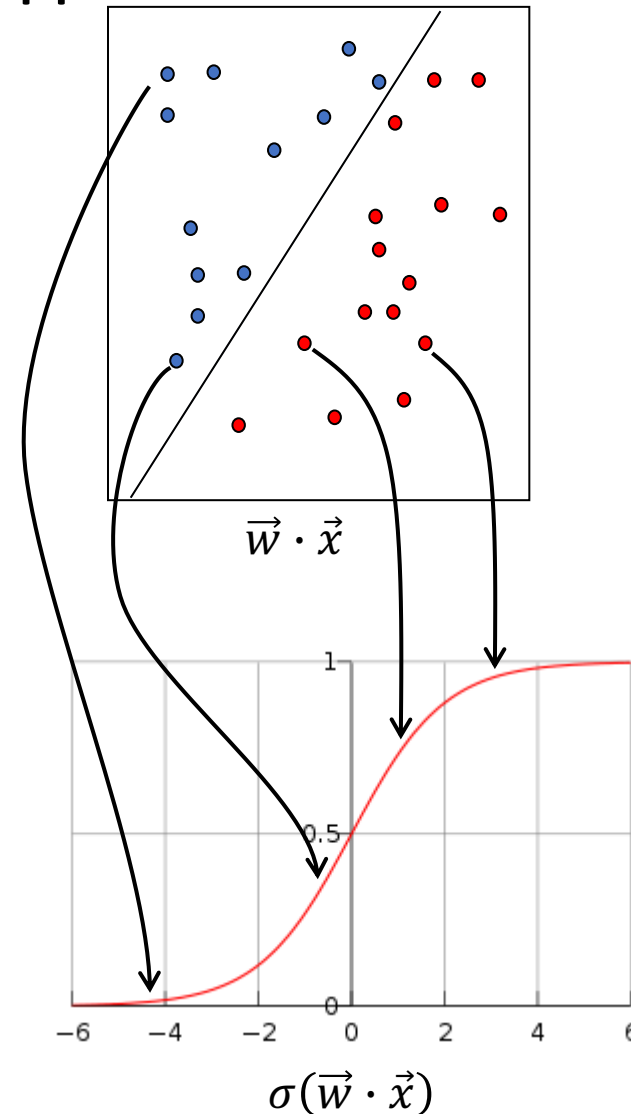
Example

- For a given instance \vec{x} ,
- if $\vec{w} \cdot \vec{x} > 0$ then we get $\sigma(\vec{w} \cdot \vec{x}) > 0.5$ and we will classify \vec{x} as 1
- if $\vec{w} \cdot \vec{x} < 0$ then we get $\sigma(\vec{w} \cdot \vec{x}) < 0.5$ and we will classify \vec{x} as 0



Probabilistic Interpretation

- We interpret $h_w(\vec{x}) = \sigma(\vec{w} \cdot \vec{x})$ as the (conditional) probability of x being classified to 1:
- $h_w(\vec{x}) = P(x|y = 1)$
- The furthest \vec{x} is from the boundary the highest its probability to be classified correctly!



Minimizing the Risk (under 0/1 cost)

- Given \vec{x} we classify it either to 0 or 1
- The risk of choosing 0 is $P(y = 1|x) = h_w(\vec{x})$
- The risk of choosing 1 is $P(y = 0|x) = 1 - h_w(\vec{x})$
- Hence, the **loss** that we may pay if we choose wrongly is:

$h_w(\vec{x})$ if $y=1$, and $1 - h_w(\vec{x})$ if $y=0$

- We can combine both by using a simple trick:

$$\text{loss}(\vec{x}, \vec{w}) = (h_w(\vec{x}))^y (1 - h_w(\vec{x}))^{1-y}$$

- We can use the -log function (monotonic decreasing) to get an equivalent loss:

$$\text{loss}(\vec{x}, \vec{w}) = -y \ln(h_w(\vec{x})) - (1 - y) \ln(1 - h_w(\vec{x}))$$

Note: this is also called Cross Entropy Loss

- Reminder: binary entropy is:

$$-p \log(p) - (1 - p) \log(1 - p)$$

- Cross entropy – between two binary distributions p and q

$$-p \log(q) - (1 - p) \log(1 - q)$$

- More generally:

$$-\sum_{x \in X} p(x) \log(q(x))$$

- Used as loss function in neural networks

Loss on All Data

- Assume we have m training examples, the total loss would be

$$\begin{aligned}\text{Loss}(\vec{w}) &= \\ &= -\sum_{d=1}^m (y^{(d)} \ln(h_w(\vec{x}^{(d)})) + (1 - y^{(d)}) \ln(1 - h_w(\vec{x}^{(d)}))) = \\ &= -\sum_{d=1}^m (y^{(d)} \ln(\sigma(\vec{w} \cdot \vec{x}^{(d)})) + (1 - y^{(d)}) \ln(1 - \sigma(\vec{w} \cdot \vec{x}^{(d)})))\end{aligned}$$

- This can be minimized using gradient descent!
- Deriving for each w_i

$$\frac{\partial}{\partial w_i} \text{loss}(\vec{w}) = -\sum_{d=1}^m \frac{\partial}{\partial w_i} ((y^{(d)} \ln(\sigma(\vec{w} \cdot \vec{x}^{(d)})) + (1 - y^{(d)}) \ln(1 - \sigma(\vec{w} \cdot \vec{x}^{(d)}))))$$

Deriving for One Given Instance $(\mathbf{x}^{(d)}, y^{(d)})$

$$\frac{\partial}{\partial w_i} \left((y \ln(\sigma(\vec{w} \cdot \vec{x})) - (1 - y) \ln(1 - \sigma(\vec{w} \cdot \vec{x}))) \right) =$$

$$\left(y \frac{1}{\sigma(\vec{w} \cdot \vec{x})} - (1 - y) \frac{1}{1 - \sigma(\vec{w} \cdot \vec{x})} \right) \frac{\partial}{\partial w_i} \sigma(\vec{w} \cdot \vec{x}) =$$

$$\left(y \frac{1}{\sigma(\vec{w} \cdot \vec{x})} - (1 - y) \frac{1}{1 - \sigma(\vec{w} \cdot \vec{x})} \right) \sigma(\vec{w} \cdot \vec{x}) (1 - \sigma(\vec{w} \cdot \vec{x})) \frac{\partial}{\partial w_i} (\vec{w} \cdot \vec{x}) =$$

$$\left(\frac{y(1 - \sigma(\vec{w} \cdot \vec{x})) - (1 - y)\sigma(\vec{w} \cdot \vec{x})}{\sigma(\vec{w} \cdot \vec{x})(1 - \sigma(\vec{w} \cdot \vec{x}))} \right) \sigma(\vec{w} \cdot \vec{x}) (1 - \sigma(\vec{w} \cdot \vec{x})) \frac{\partial}{\partial w_i} (\vec{w} \cdot \vec{x}) =$$
$$\left(y(1 - \sigma(\vec{w} \cdot \vec{x})) - (1 - y)\sigma(\vec{w} \cdot \vec{x}) \right) x_i =$$
$$(\mathbf{y} - \boldsymbol{\sigma}(\vec{w} \cdot \vec{x})) x_i$$

Minimizing Total Loss

$$\begin{aligned}\frac{\partial}{\partial w_i} \text{loss}(\vec{w}) = & -\sum_{d=1}^m \frac{\partial}{\partial w_i} \left((y^{(d)} \ln(\sigma(\vec{w} \cdot \vec{x}^{(d)})) + (1 - y^{(d)}) \ln(1 - \sigma(\vec{w} \cdot \vec{x}^{(d)}))) \right) = \\ & -\sum_{d=1}^m \left(y^{(d)} - \sigma(\vec{w} \cdot \vec{x}^{(d)}) \right) x_i^{(d)} = \sum_{j=1}^m \left(\sigma(\vec{w} \cdot \vec{x}^{(d)}) - y^{(d)} \right) x_i^{(d)}\end{aligned}$$

Now we can define the gradient:

$$\nabla \text{cost}(\vec{w}) = \left(\frac{\partial}{\partial w_1} \text{cost}(\vec{w}), \dots, \frac{\partial}{\partial w_n} \text{cost}(\vec{w}) \right)$$

and use gradient descent

Gradient Descent Rule

- $\Delta w_i = -\eta \sum_{d=1}^m \left((\sigma(\vec{w} \cdot \vec{x}^{(d)}) - y^{(d)}) x_i^{(d)} \right)$

- $w_i = w_i + \Delta w_i$

- Or the stochastic version:

$$w_i = w_i - \eta (\sigma(\vec{w} \cdot \vec{x}^{(d)}) - y^{(d)}) x_i^{(d)}$$

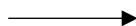
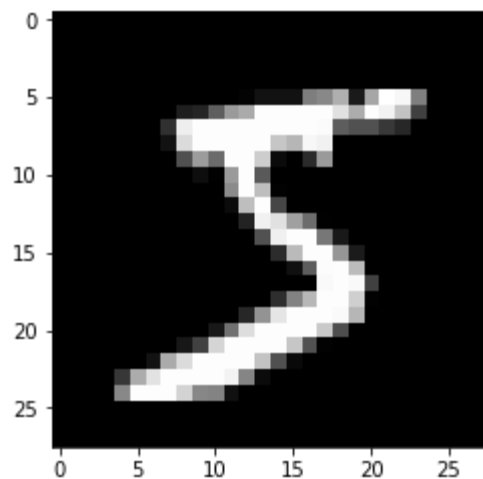
- This is called **Logistic Regression** although it is a **classification** algorithm not to be confused with regression
- Instead of using just linear relation, we map the linear relation with another function

Multiple classes

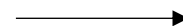
MNIST training data



Multi Class Classification



Model ?



0

1

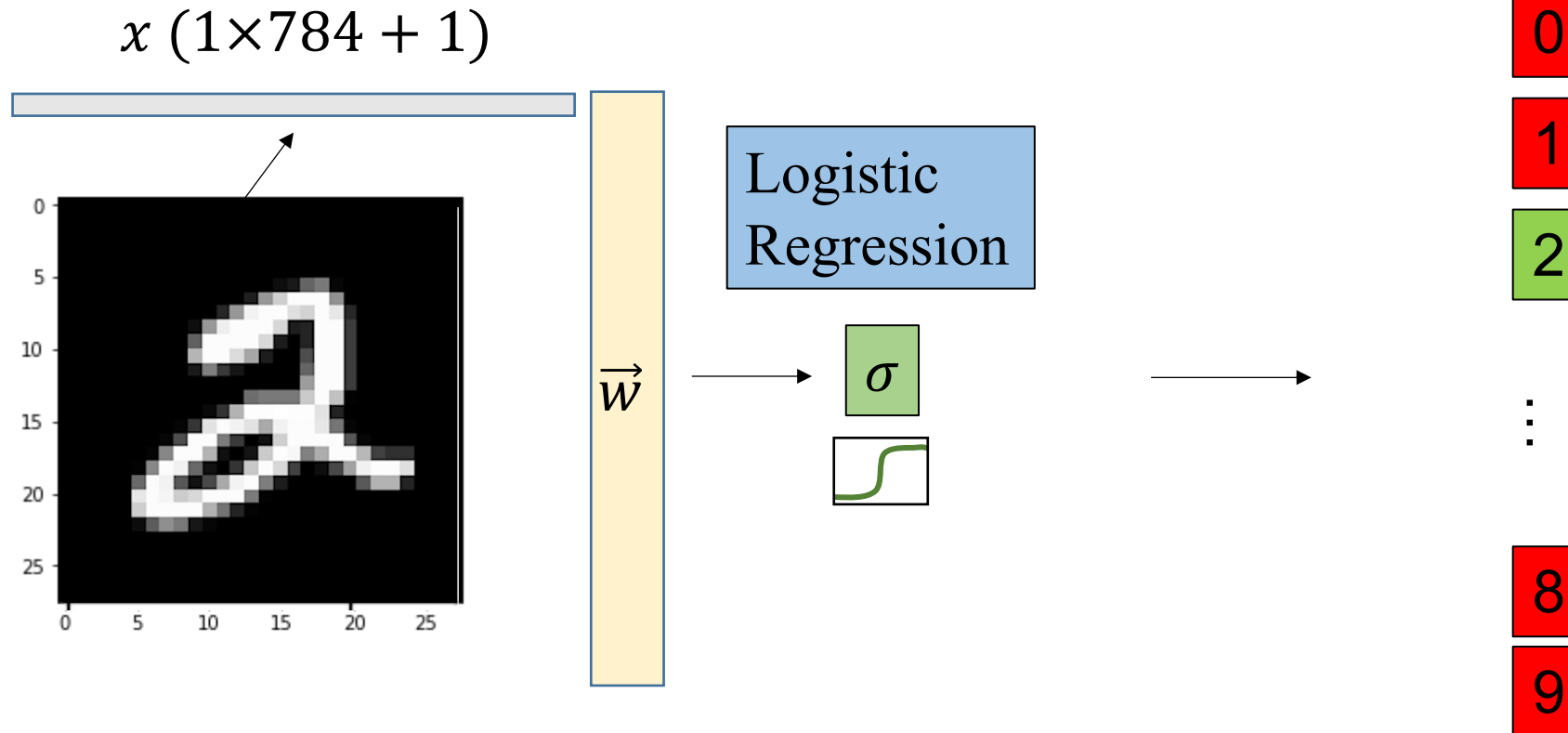
2

⋮

8

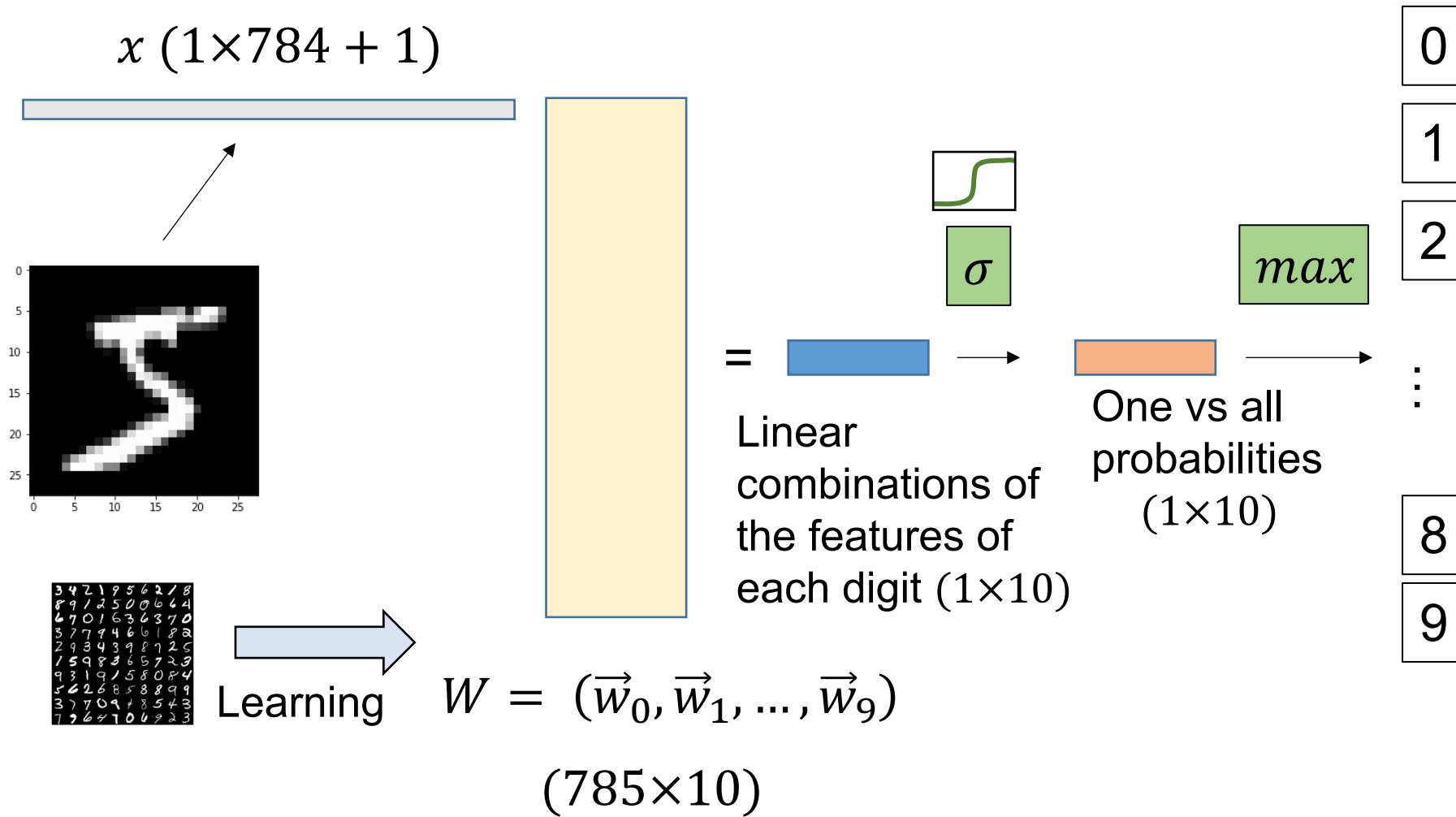
9

One vs. All Classifier for "2"



Positive : images of "2"
Negative: all others

Multi-class by using One vs All



Summary

- Logistic regression is an approach to classification. The name is misleading ...
- The execution algorithm (or the model) classifies based on a vector of numerical features
- The approach is based on the LoR assumption.
Namely $P(\vec{x}|y = 1) = \sigma(w^T \vec{x})$
- The learning process uses Gradient Descent to find w .
- No pseudo inverse solution for LoR
- Extends to multiple classes (OvA, softmax, others ...)

Comparison of Linear Classification Algorithms

- Perceptron:

$$w_i = w_i - \eta(\text{sgn}(\vec{w} \cdot \vec{x}^{(d)}) - y^{(d)})x_i^{(d)}$$

- Least Square:

$$w_i = w_i - \eta(\vec{w} \cdot \vec{x}^{(d)} - y^{(d)})x_i^{(d)}$$

- Logistic Regression:

$$w_i = w_i - \eta(\sigma(\vec{w} \cdot \vec{x}^{(d)}) - y^{(d)})x_i^{(d)}$$

Comparison of Loss functions

- Regression:
$$E[\vec{w}] = \frac{1}{|D|} \sum_{d \in D} (\vec{w} \cdot \vec{x}^{(d)} - t_d)^2$$

- Perceptron:
$$\frac{1}{|D|} \sum_{d \in D} (\text{sgn}(\vec{w} \cdot \vec{x}^{(d)}) - t_d)$$

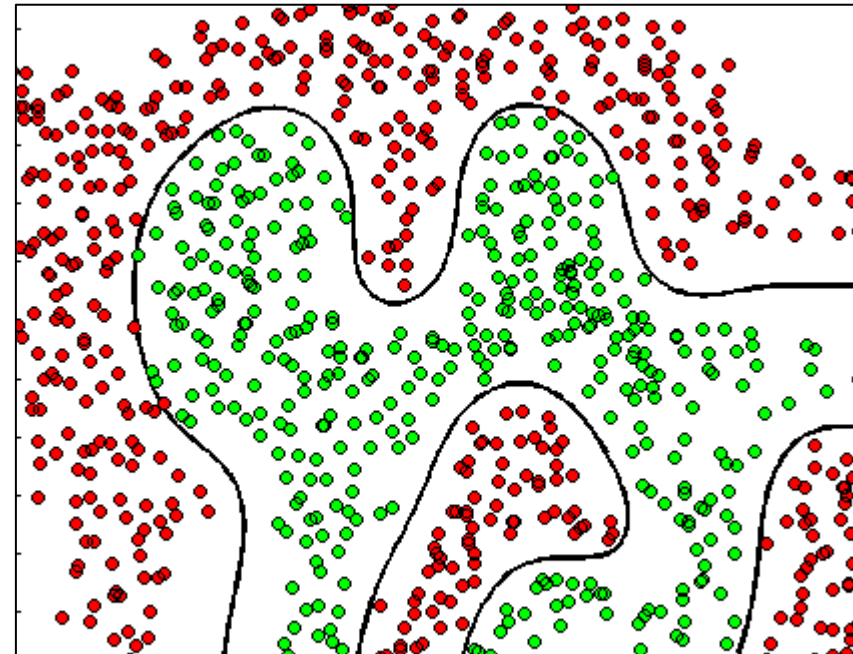
- LMS:
$$E[\vec{w}] = \frac{1}{|D|} \left[\sum_{d \in D^+} (\vec{w} \cdot \vec{x}^{(d)} - 1)^2 + \sum_{d \in D^-} (\vec{w} \cdot \vec{x}^{(d)} + 1)^2 \right]$$

- Logistic Regression

$$E[\vec{w}] = \frac{1}{|D|} \sum_{d=1}^m (-y^{(d)} \ln(\sigma(\vec{w} \cdot \vec{x}^{(d)})) - (1 - y^{(d)}) \ln(1 - \sigma(\vec{w} \cdot \vec{x}^{(d)})))$$

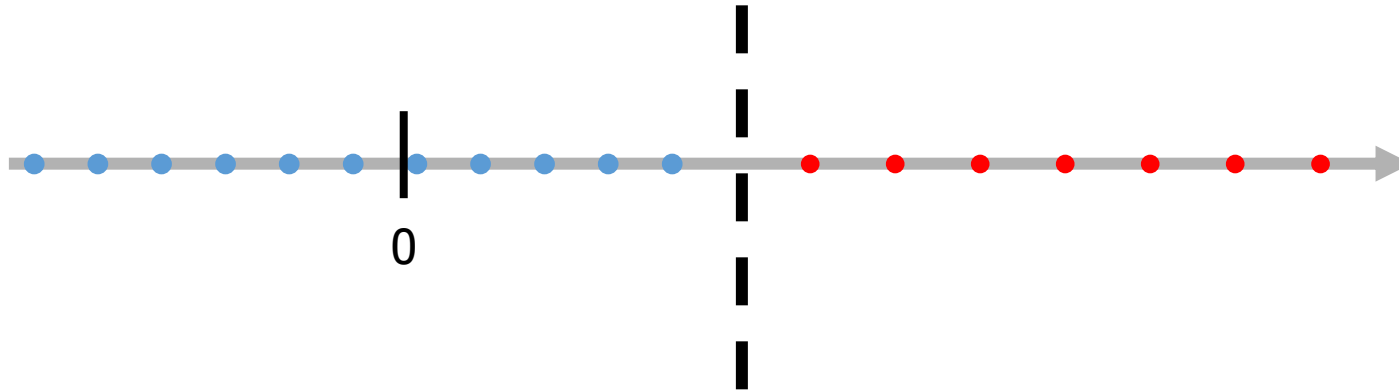
Non-Linear Decision Boundaries

- Decision boundaries which separate between classes may not always be linear
- In fact, they will sometimes be very complex boundaries and therefore may sometimes require the use of highly non-linear discriminant functions



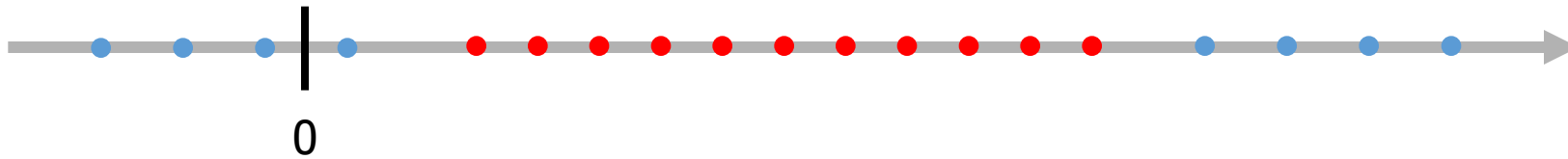
The Perceptron and Linear Separability

In 1D: $w_1 \cdot x + w_0 > 0$ OR $C(x) = \text{sgn}(w, x)$
 $w = (w_1, w_0)$

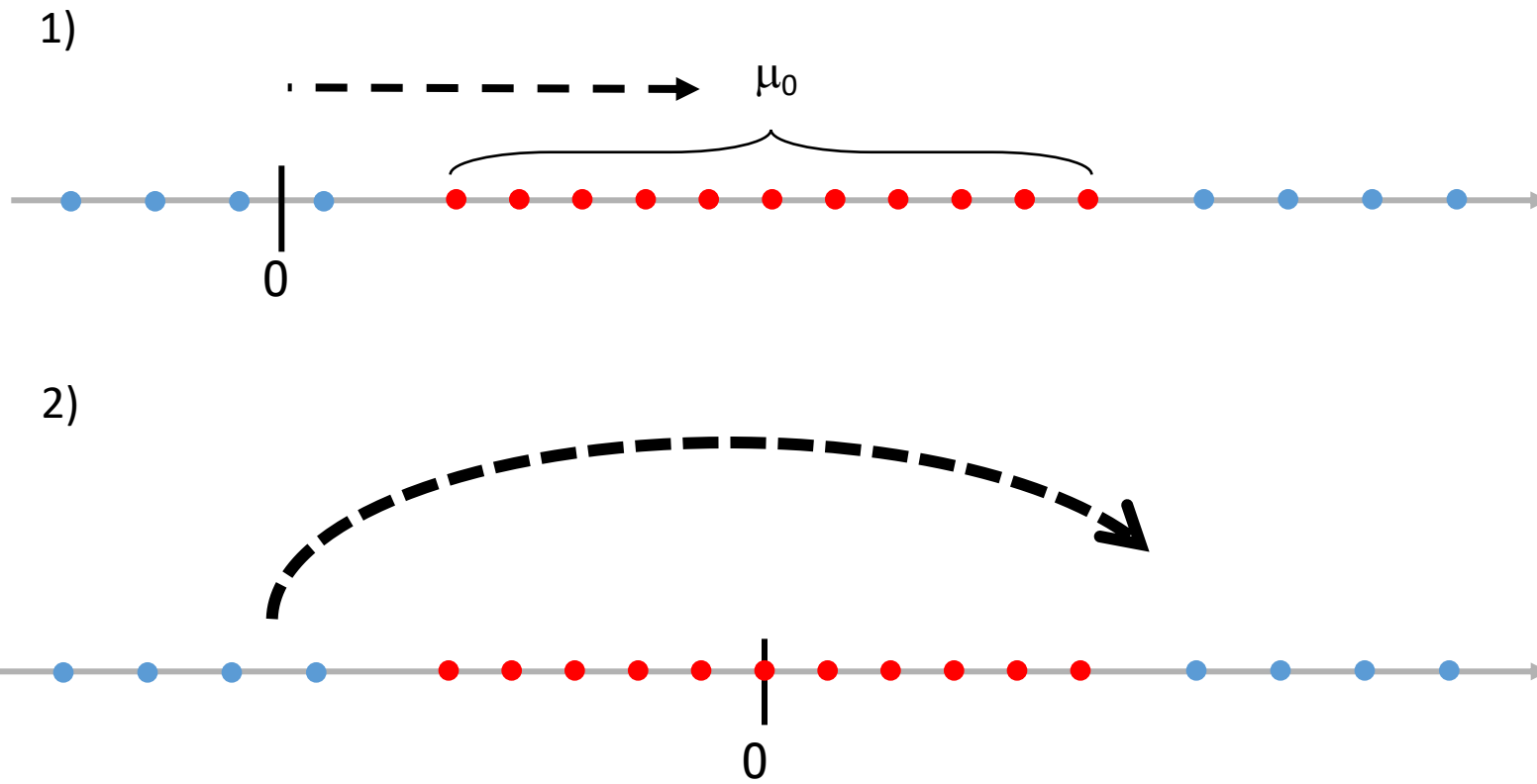


Can We Build a 1D Perceptron for This?

- Red: $C(x) = -1$
- Blue: $C(x) = +1$



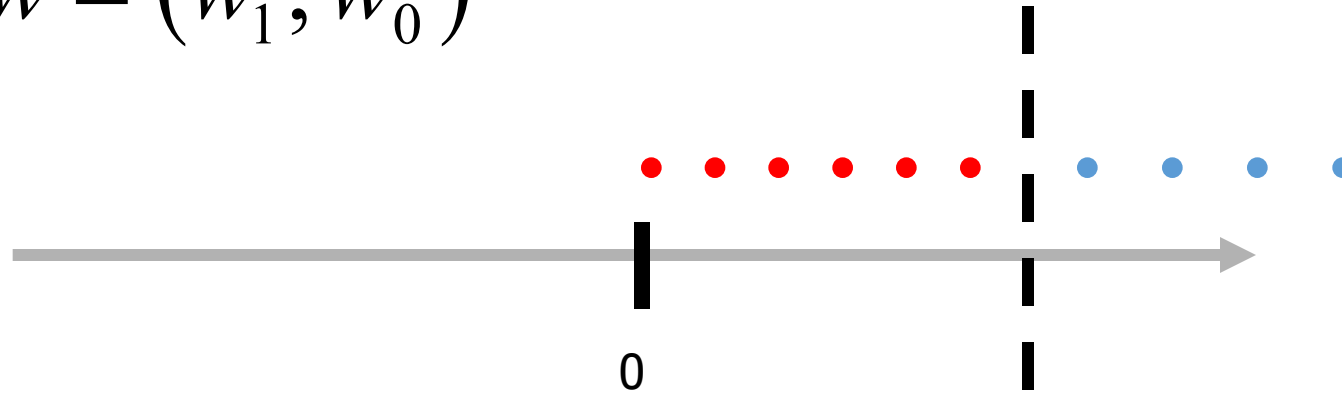
Example: Mapping x to $y=(x-\mu_0)^2$



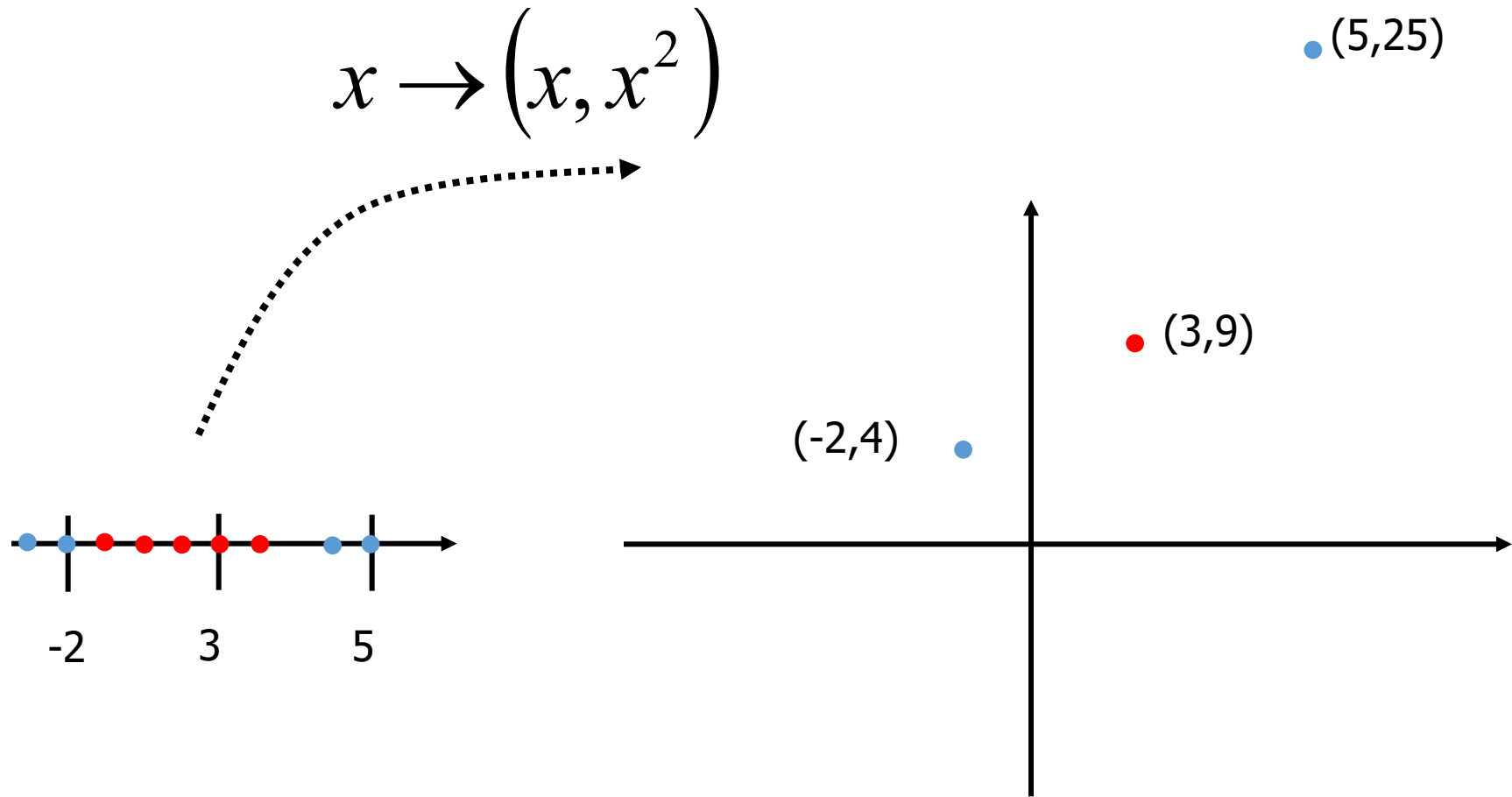
1D Perceptron After Mapping

$$w_1 \cdot y + w_0 = w_1 \cdot (x - \mu_0)^2 + w_0 > 0$$

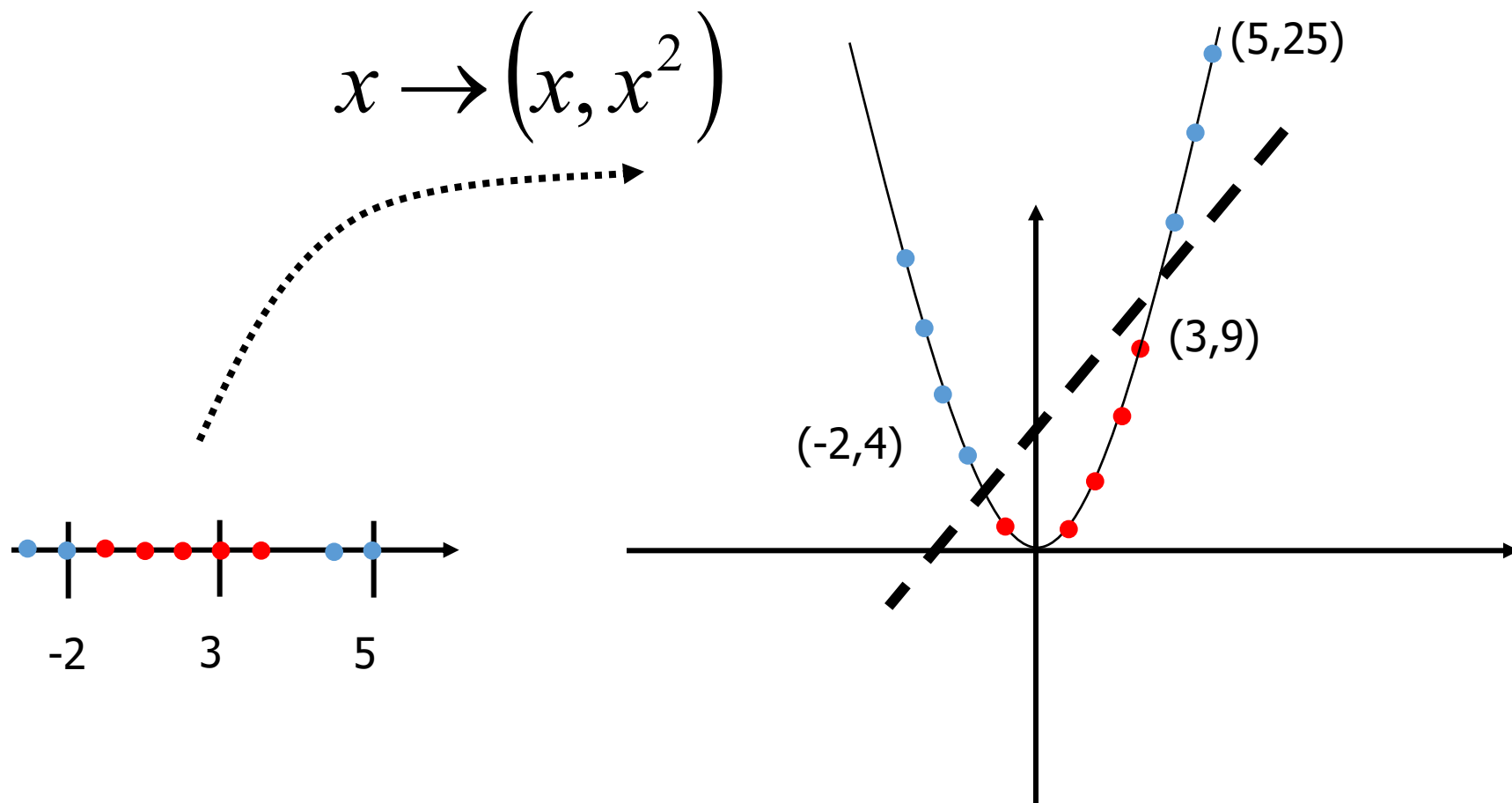
$$w = (w_1, w_0)$$



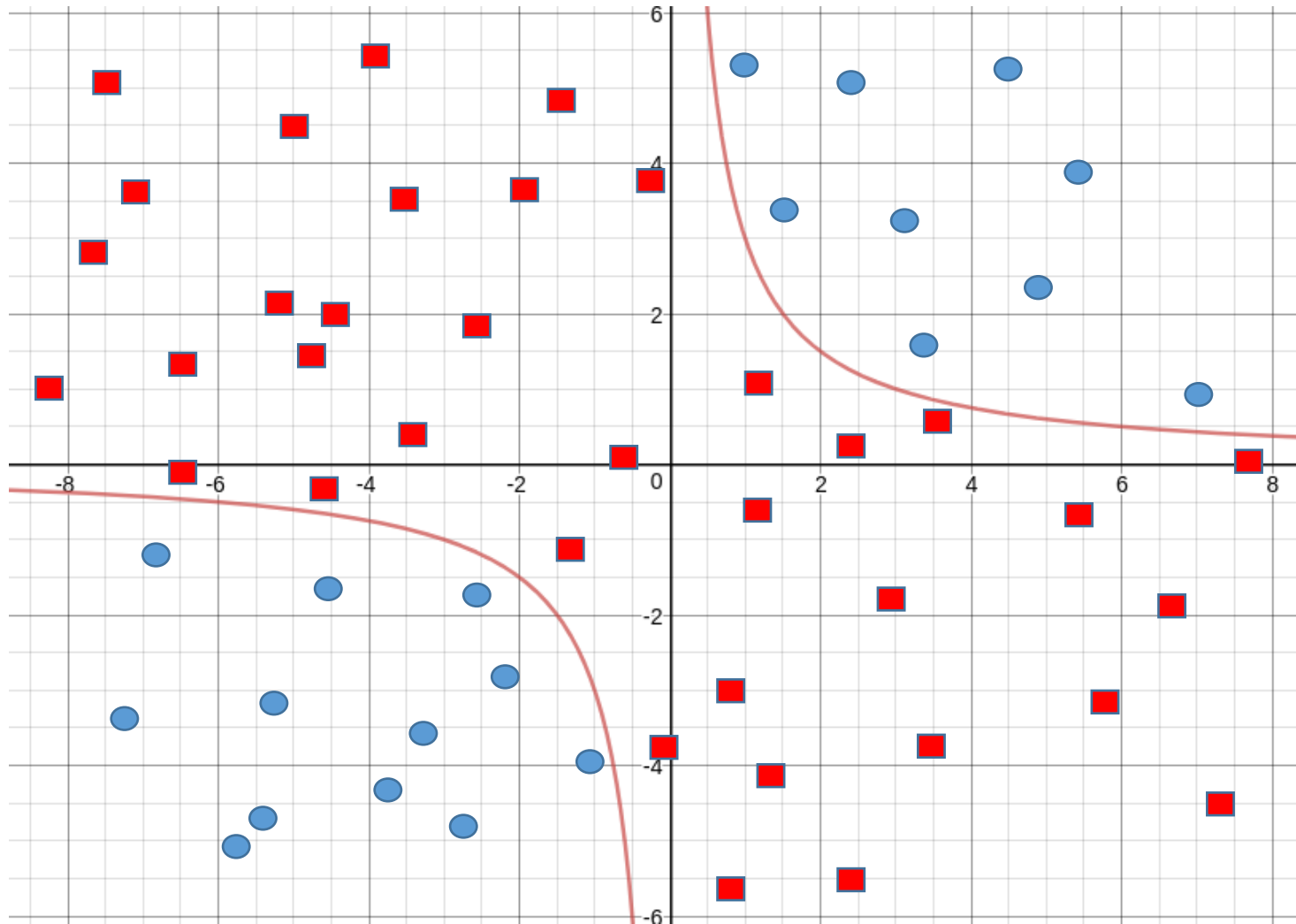
Mapping to Higher Dimension



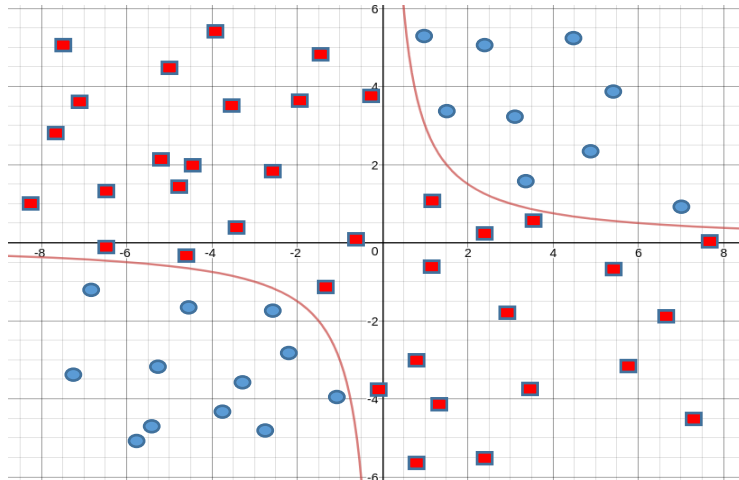
Linear Separability in the Target Space



Linear separation in 2D? 3D?



Mapping into 3D



-----> ??

$$(x, y) \rightarrow (x, y, xy)$$

Are the classes linearly separable now?

Will pick up from here next time
and continue to SVMs