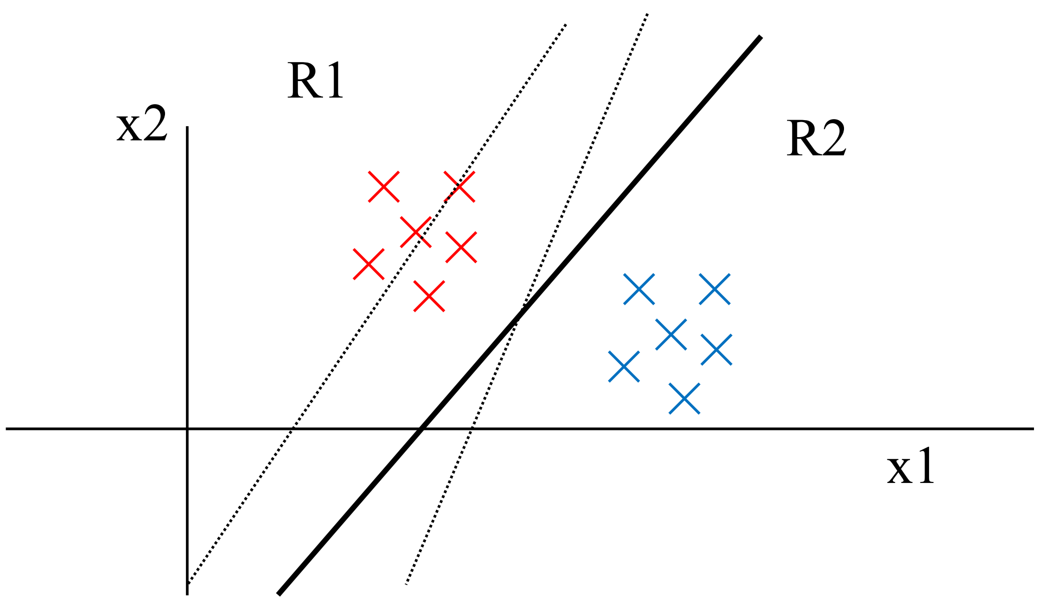
Lecture 3
Linear Classifiers

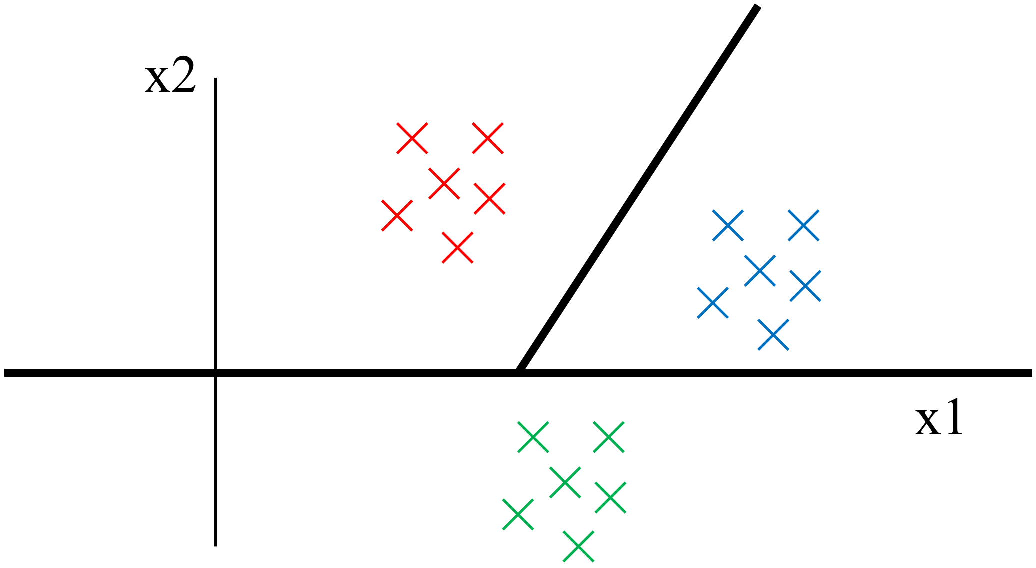
Linear Classifiers

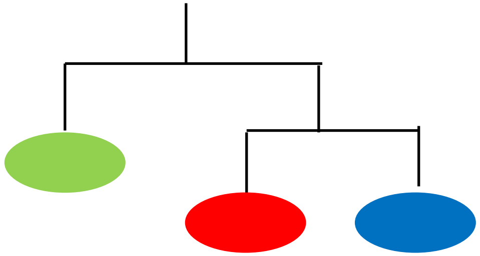
- **Basic idea**: for a set of exemplary feature vectors belonging to each class, find a decision boundary in terms of linear functions of the features
- **Training:** find the decision boundary
  - in general, multiple solutions are possible



Generalizations

- ➔ **A linear classifier has region boundaries described by lines/plans/hyperplanes**
  - ⇨ Decision trees can combine multiple line





Linear Classifier Definition

- ➔ **Let $\underline{x}=(x_1,x_2,..x_l)$ a vector of features in a generic l-dimensional space**
- ➔ **Can we decide to which class it belongs to by means of a linear function g(x)?**
  - ⇨ $g(\underline{x})= w_1x_1+w_2x_2+\ldots w_lx_l +w_0$ linear combination of $\underline{x}$
- ➔ **Assuming two classes only are possible, classification could work as:**
  - ⇨ $g(\underline{x})>0$ if $x \in \omega 1$, $g(\underline{x}) <0$ if $x \in \omega 2$
- ➔ **Decision hyperplane:**

$$\text{Assume } \underline{x}_1, \underline{x}_2 \text{ on the decision hyperplane:}$$

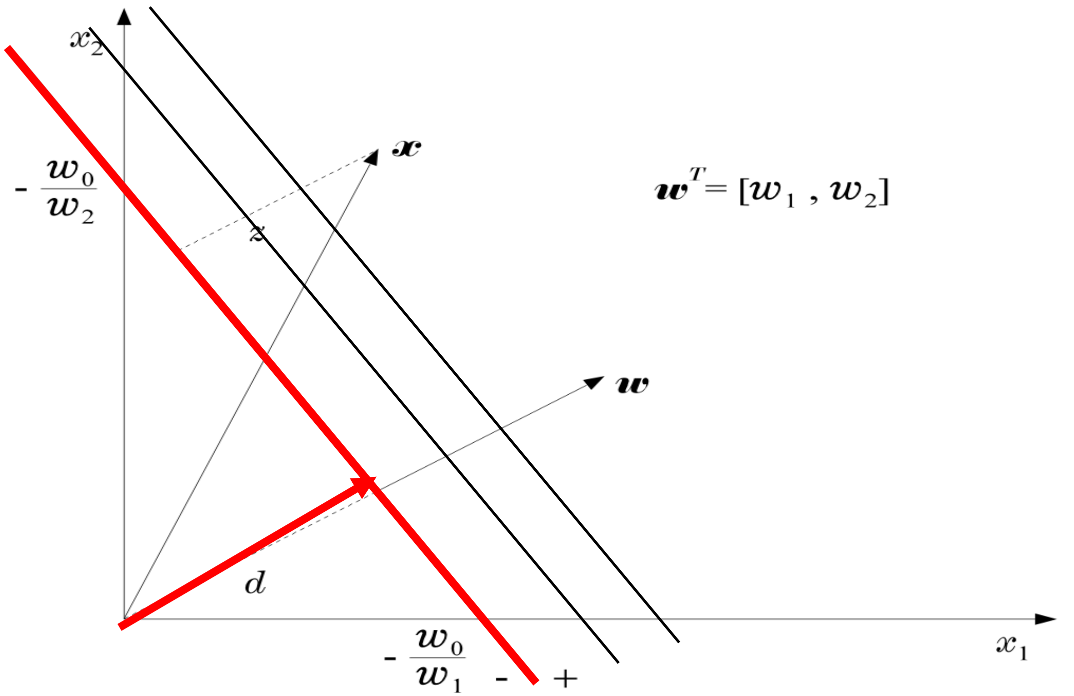
$$0 = \underline{w}^T \underline{x}_1 + w_0 = \underline{w}^T \underline{x}_2 + w_0 \Rightarrow$$

$$\underline{w}^T (\underline{x}_1 - \underline{x}_2) = 0 \;\; \forall \underline{x}_1, \underline{x}_2$$

Hence:

$$\boxed{\underline{w} \perp \text{on the decision hyperplane}}$$

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



$$d = \frac{|w_0|}{\sqrt{w_1^2 + w_2^2}}, \quad z = \frac{|g(\underline{x})|}{\sqrt{w_1^2 + w_2^2}}$$

The Perceptron Algorithm

➔ **How do we define the $\underline{w}^*$ coefficients starting from a set of available test points?**

⇨ Assume linearly separable classes, i.e.,

$$\exists\ \underline{w}^*:\ \underline{w}^{*T}\,\underline{x} + w_0^* > 0\ \ \forall \underline{x} \in \omega_1$$
$$\underline{w}^{*T}\,\underline{x} + w_0^* < 0\ \ \forall \underline{x} \in \omega_2$$

⇨ We can express the boundary conditions as $\underline{w}'^T\underline{x}' = 0$, being

$$\rightarrow\quad \underline{w}' \equiv \begin{bmatrix} \underline{w}^* \\ w_0^* \end{bmatrix},\ \underline{x}' = \begin{bmatrix} \underline{x} \\ 1 \end{bmatrix}$$

$$\rightarrow\quad \underline{w}^{*T}\,\underline{x} + w_0^* = \underline{w}'^T\,\underline{x}' = 0$$

- Our goal: Compute a solution, i.e., a hyperplane <u>*w*</u>, so that

$$\underline{w}^T \underline{x} (><) 0 \quad \underline{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

- The steps
  - →Define a cost function to be minimized
  - →Choose an algorithm to minimize the cost function
  - →The minimum corresponds to a solution

Cost Function

- ➔ **In general, not all the test points can be correctly classified for a given $\underline{w}$**
  - ⇨ For each $\underline{w}$ vector we can quantify the classification error as

$$J(\underline{w}) = \sum_{\underline{x} \in Y} (\delta_x \underline{w}^T \underline{x}) \qquad J(\underline{w}) \geq 0$$

    - →Where Y is the subset of the vectors wrongly classified by $\underline{w}$ (i.e. $\underline{w}^T \underline{x} < 0$ if x is in $\omega 1$ and $\underline{w}^T \underline{x} > 0$ if x is in $\omega 2$), and

$$\delta_x = -1 \text{ if } \underline{x} \in Y \text{ and } \underline{x} \in \omega_1$$

$$\delta_x = +1 \text{ if } \underline{x} \in Y \text{ and } \underline{x} \in \omega_2$$

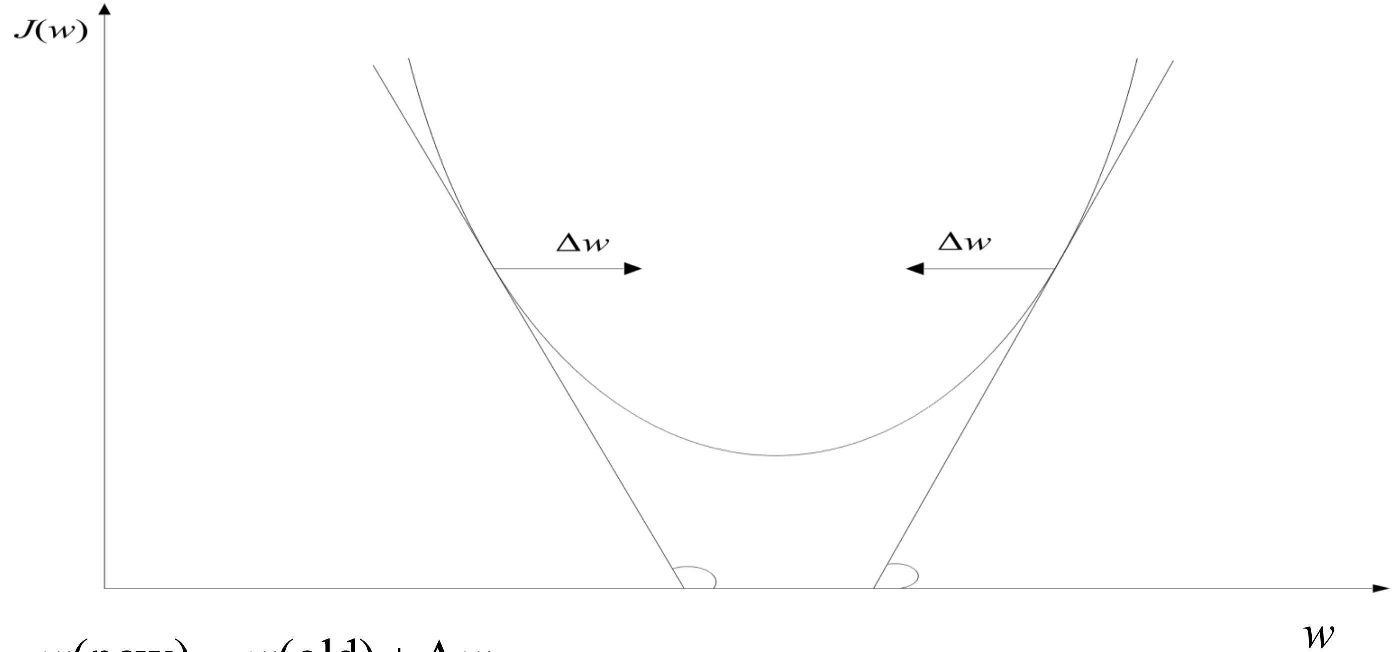
  - ⇨ When Y=(empty set) a solution is achieved and

$$J(\underline{w}) = 0$$

Cost Function

- ➔ **For each component w_i, $J(\underline{w})$ is piecewise linear (WHY?)**
  - ⇨ If Y does not change, varying w_i with continuity implies varying J(w) linearly
  - ⇨ Es. Two points $\underline{x1}$ and $\underline{x2}$ in Y:
    - → $J(\underline{w}) = w1\ (\delta 1\ x11 + \delta 2\ x21) + w2\ (\delta 1\ x12 + \delta 2\ x22) + \ldots wl\ (\delta 1\ x1l + \delta 2\ x2l)$;
    - → Linear function of w_i, as long as Y does not change
      - » Increasing or decreasing according to $\delta 1\ x1i + \delta 2\ x2i$

- ➔ **How to find the vector $\underline{w}$?**
  - ⇨ The Algorithm
    - → The philosophy of the gradient descent is adopted.



$$\underline{w}(\text{new}) = \underline{w}(\text{old}) + \Delta \underline{w}$$

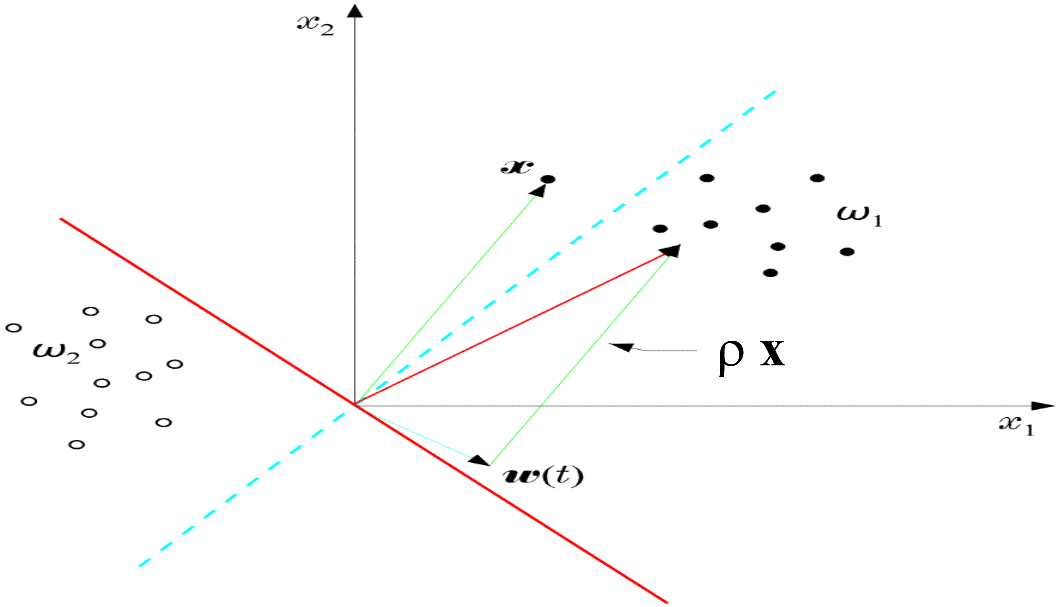
$$\Delta \underline{w} = -\mu \frac{\partial J(\underline{w})}{\partial \underline{w}} \Big| \underline{w} = \underline{w}(\text{old})$$

→Wherever valid

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \Big(\sum_{\underline{x} \in Y} \delta_x \underline{w}^T \underline{x}\Big) = \sum_{\underline{x} \in Y} \delta_x \underline{x}$$

→ $$\boxed{\underline{w}(t+1) = \underline{w}(t) - \rho_t \sum_{\underline{x} \in Y} \delta_x \underline{x}}$$

⇨ A geometrical summary:



$$\underline{w}(t+1) = \underline{w}(t) + \rho_t \underline{x}$$
$$= \underline{w}(t) - \rho_t \delta_x \underline{x} \quad (\delta_x = -1)$$

⇨ The perceptron algorithm **converges** in a **finite** number of iteration steps to a solution if

$$\lim_{t\to\infty} \sum_{k=0}^{t} \rho_k \to \infty, \qquad \lim_{t\to\infty} \sum_{k=0}^{t} \rho_k^{\;2} < +\infty$$

$$\text{e.g.,}: \ \rho_t = \frac{c}{t}$$

➔ A useful variant of the perceptron algorithm

$$\underline{w}(t+1) = \underline{w}(t) + \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \le 0 \\ \underline{x}_{(t)} \in \omega_1 \end{array}$$

$$\underline{w}(t+1) = \underline{w}(t) - \rho \underline{x}_{(t)}, \quad \begin{array}{c} \underline{w}^T(t)\underline{x}_{(t)} \ge 0 \\ \underline{x}_{(t)} \in \omega_2 \end{array}$$

$$\underline{w}(t+1) = \underline{w}(t) \quad \text{otherwise}$$

- ⇨ It is a reward and punishment type of algorithm
- ⇨ It can be proved that converges in a finite number of steps, with a costant ρ

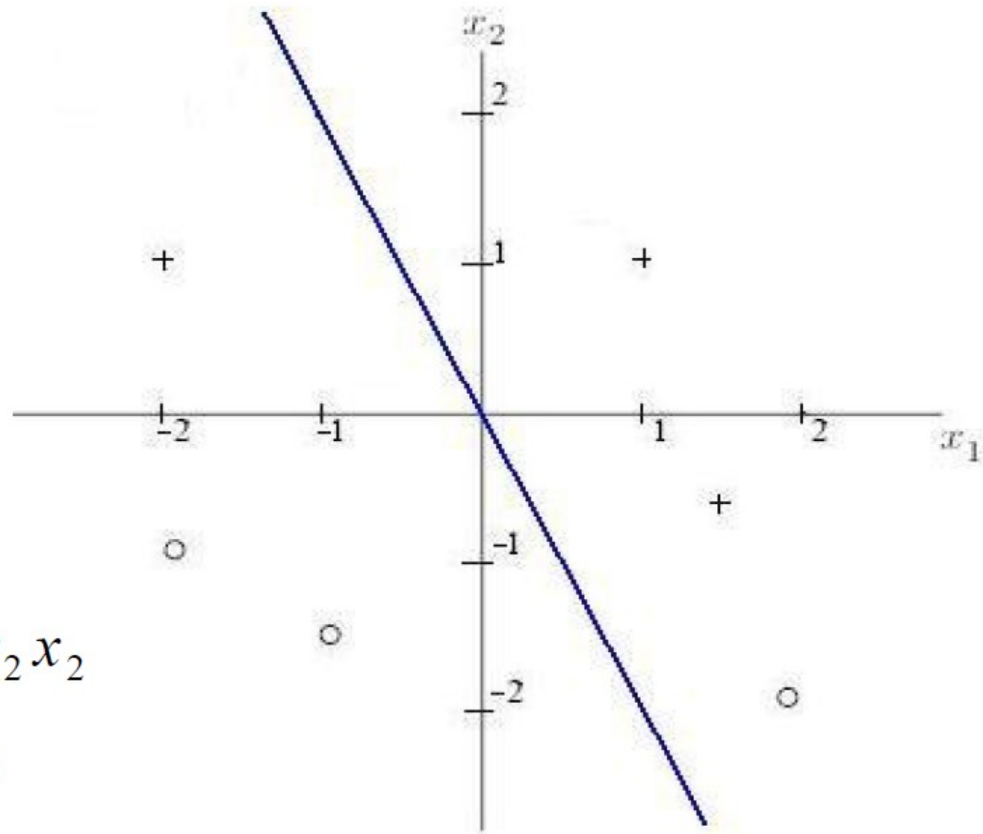
Learning example

Initial Values:

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$0 = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0 + x_1 + 0.5\mathrm{x}_2$$
$$\Rightarrow \; x_2 = -2\mathrm{x}_1$$



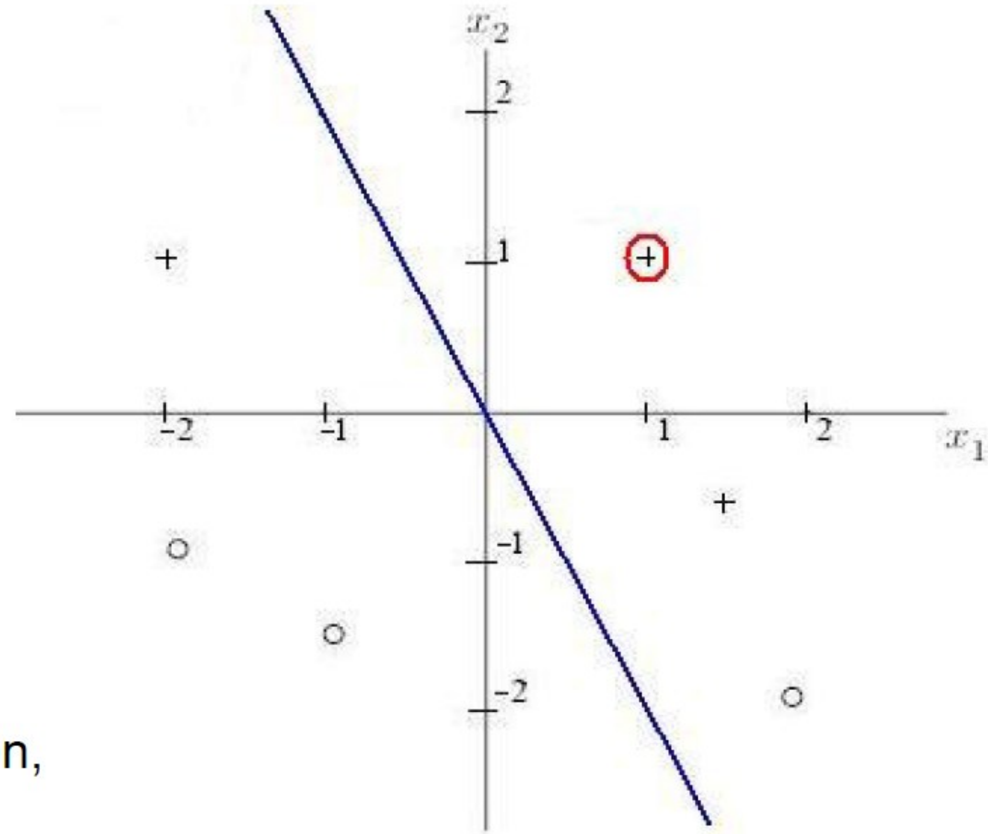
Learning example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 1$, $x_2 = 1$
$w^T x > 0$

Correct classification, no action



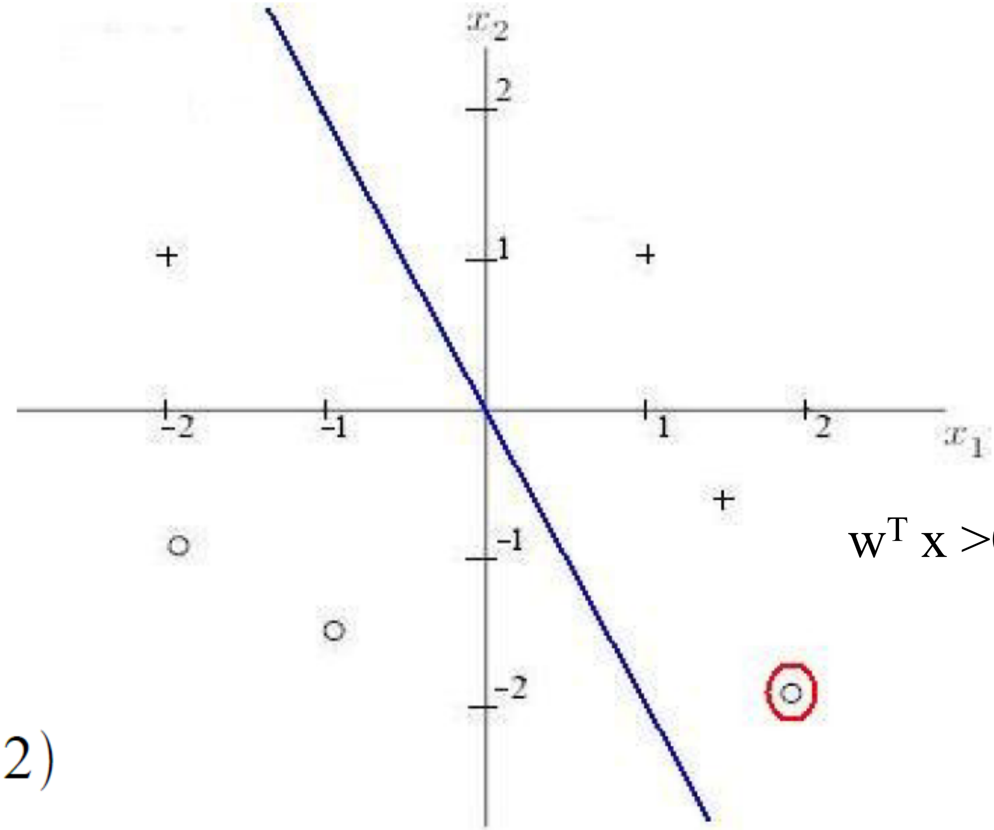
Learning example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 2,\ x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$
$$w_1 = w_1 - 0.2 * 2$$
$$w_2 = w_2 - 0.2 * (-2)$$



$w^T x > 0$

*J(w)=wo+w1*x1+w2*x2*

Learning example

$\eta = 0.2$

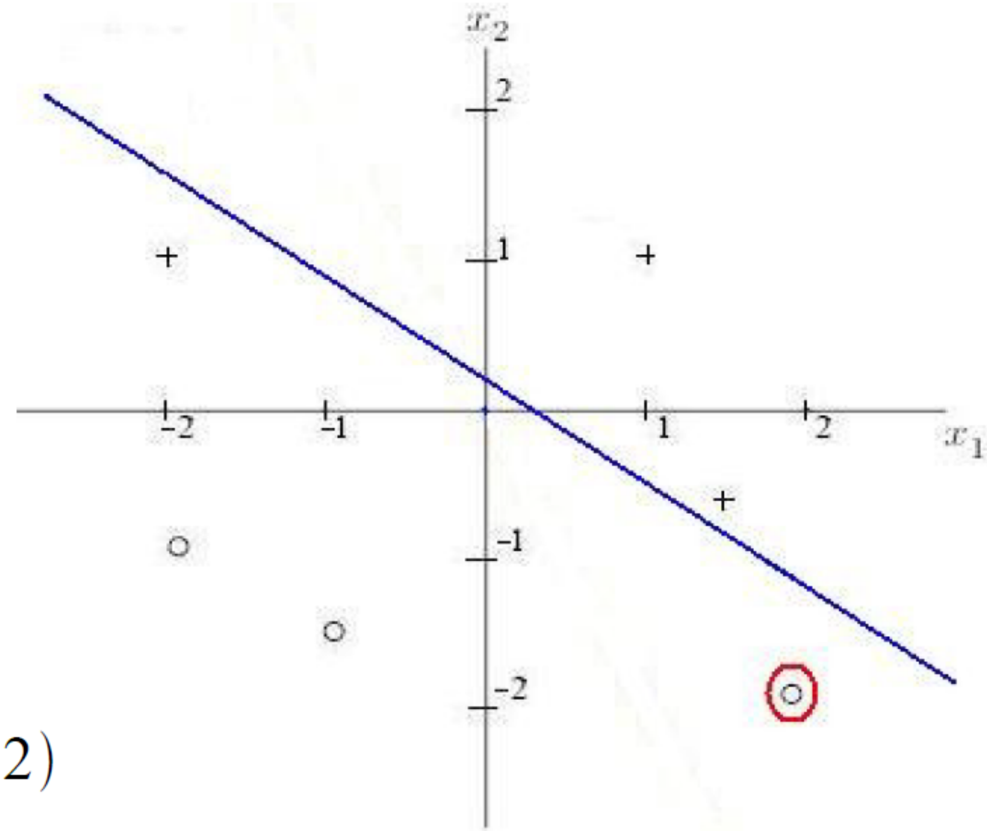
$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = 2,\ x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

$$w_2 = w_2 - 0.2 * (-2)$$



Learning example

$\eta = 0.2$

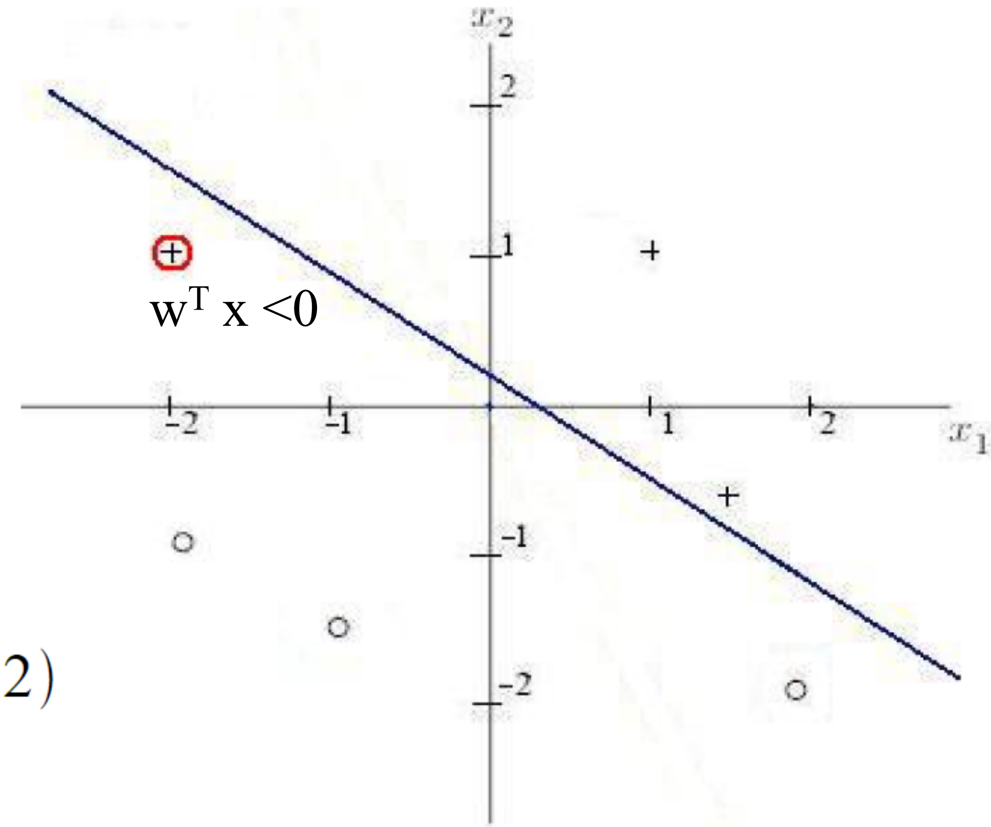
$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -2,\ x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$



$J(w) = -(wo + w1 * x1 + w2 * x2)$

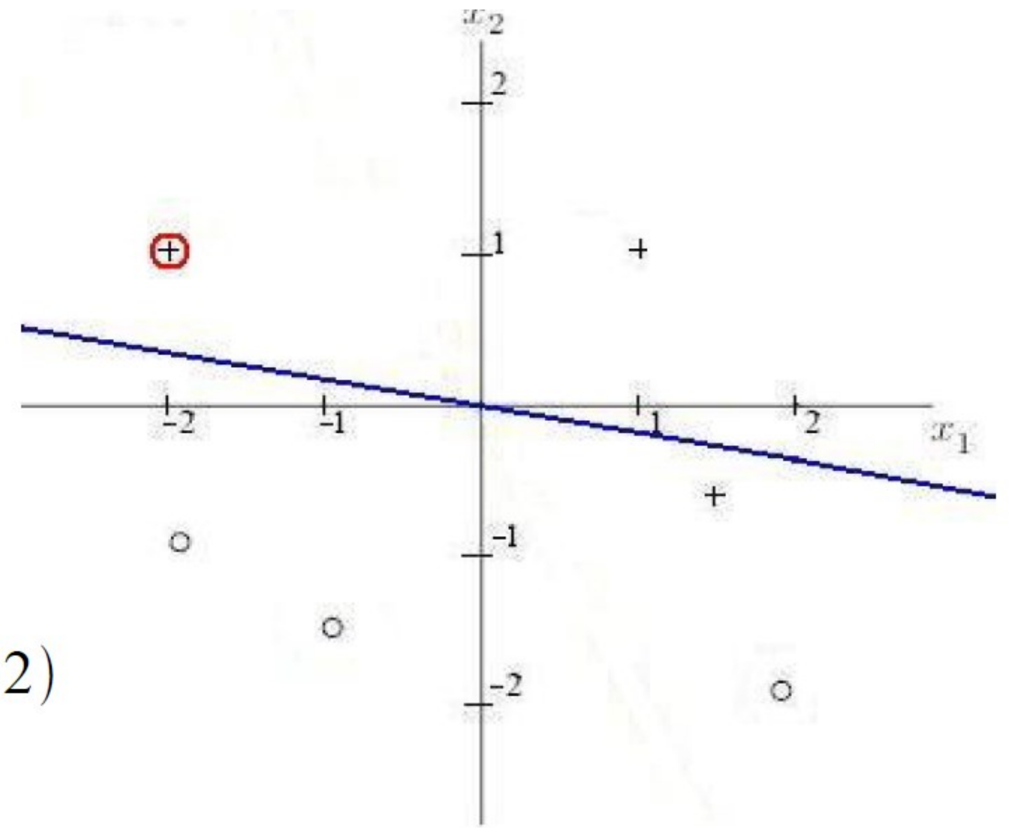
Learning example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

x_1 = -2, x_2 = 1

$$w_0 = w_0 + 0.2 * 1$$
$$w_1 = w_1 + 0.2 * (-2)$$
$$w_2 = w_2 + 0.2 * 1$$



Convergence Proof

w(j+1)=w(j)+ρx(j) if w(j)x(j)<0 and x(k) $\in \omega 1$
w(j+1)=w(j)-ρx(j) if w(j)x(j)>0 and x(k) $\in \omega 2$

Assume **w(0)=0** and **k** training points belonging to $\omega 1$ misclassified

w(k)=w(k-1)+ ρx(k-1) =w(k-2)+ ρx(k-2)+ ρx(k-1) =...
w(0)+ ρx(0)+ ρx(1)+... ρx(k-1)

If classes are separable, $\exists$ w* : w*x(k)>0 for each training point in $\omega 1$; multiplying previous equation for w* we get:

$$w^* w(k)=w^*w(0)+ \rho \underbrace{w^*x(0)+\rho w^*x(1)+..\ \rho w^*}_{\text{all positive >= |a|=max w*x}}x(k-1)$$

0

w* w(k) >= k a

Convergence Proof

From cauchy-scharz inequality:

$$|w^*|^2 \, |w(k)|^2 >= |w^* \, w(k)|^2 >= (ka)^2$$

Therefore:

$$|w(k)|^2 >= (ka)^2 / |w^*|^2$$

Going back to a generic step:

$$w(j+1) = w(j) + \rho \, x(j) \text{ for } j=0, 2, \ldots k-1$$

Taking the modules:

$$|w(j+1)|^2 = |w(j) + \rho \, x(j)|^2 = |w(j)|^2 + \rho^2 |x(j)|^2 + 2 \rho \, w(j) x(j)$$

Being x(j) in $\omega 1$ misclassified, i.e. $w(j)x(j)<0$:

$$|w(j+1)|^2 - |w(j)|^2 <= \rho^2 \, |x(j)|^2$$

$$|w(j)|^2 - |w(j-1)|^2 <= \rho^2 |x(j-1)|^2 \ldots$$

$$|w(1)|^2 - |w(0)|^2 <= \rho^2 |x(0)|^2$$

Convergence Proof

We get:

$$|w(k)|^2 <= \rho^2 (|x(0)|^2 + |x(1)|^2 +.. |x(k-1)|^2) <= k |b|$$

Being b the maximum module of the misclassified points.

Finally:

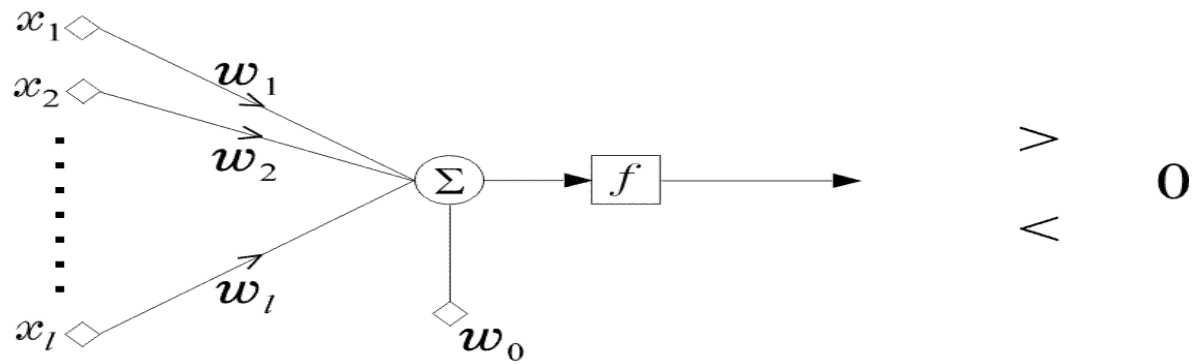
$$(ka)^2/|w^*|^2 <= |w(k)|^2 <= k |b|$$

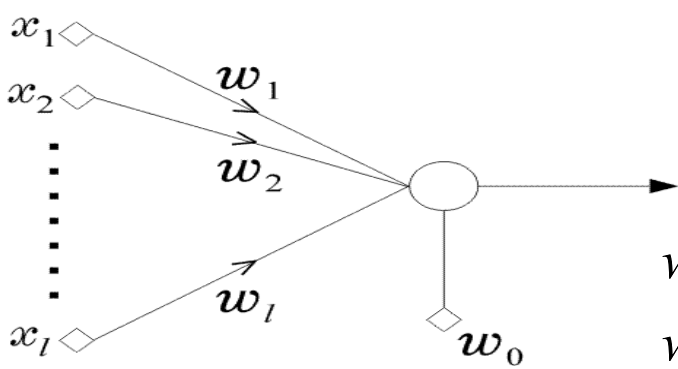
To satisfy both the equations k cannot be greater than

$$k=|b| \, |w^*|^2/a^2$$

Therefore, in a finite number of steps the algorithm converges.

The preceptron





$w_i's$ synapses or synaptic weights

w_0 threshold

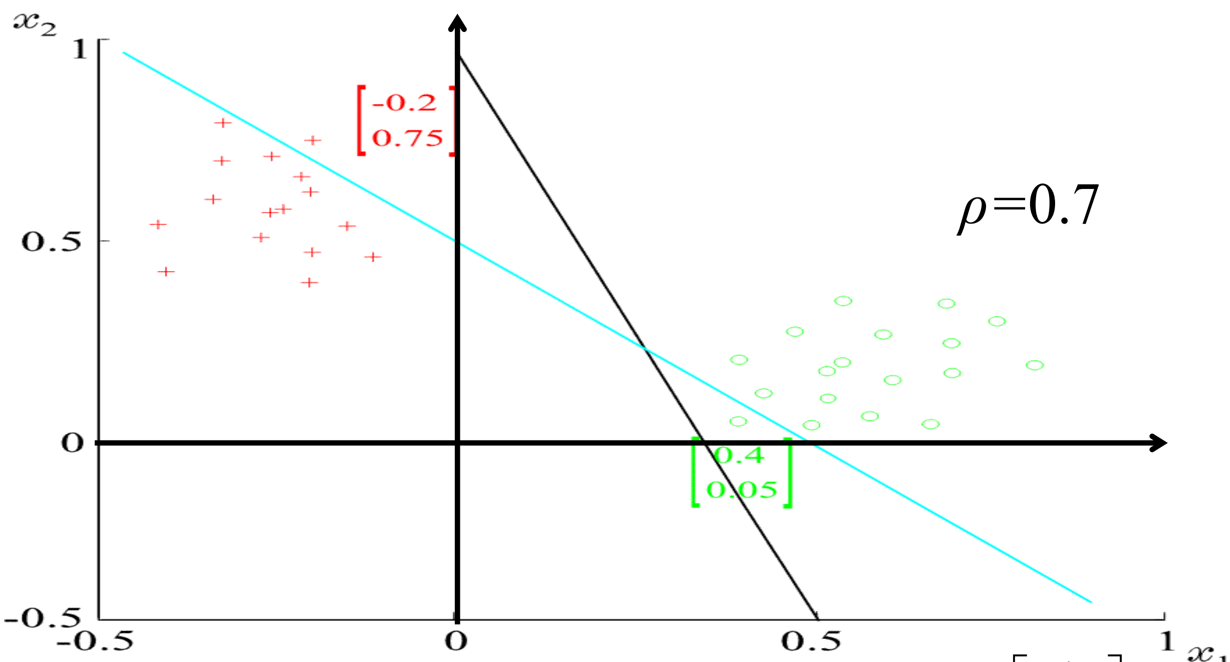
It is a **learning machine** that learns from the **training vectors** via the perceptron algorithm

⇨ Example: At some stage t the perceptron algorithm results in

$$w_1 = 1,\ \ w_2 = 1,\ \ w_0 = -0.5$$

$$x_1 + x_2 - 0.5 = 0$$

The corresponding hyperplane is



ρ=0.7

Perceptron algorithm:

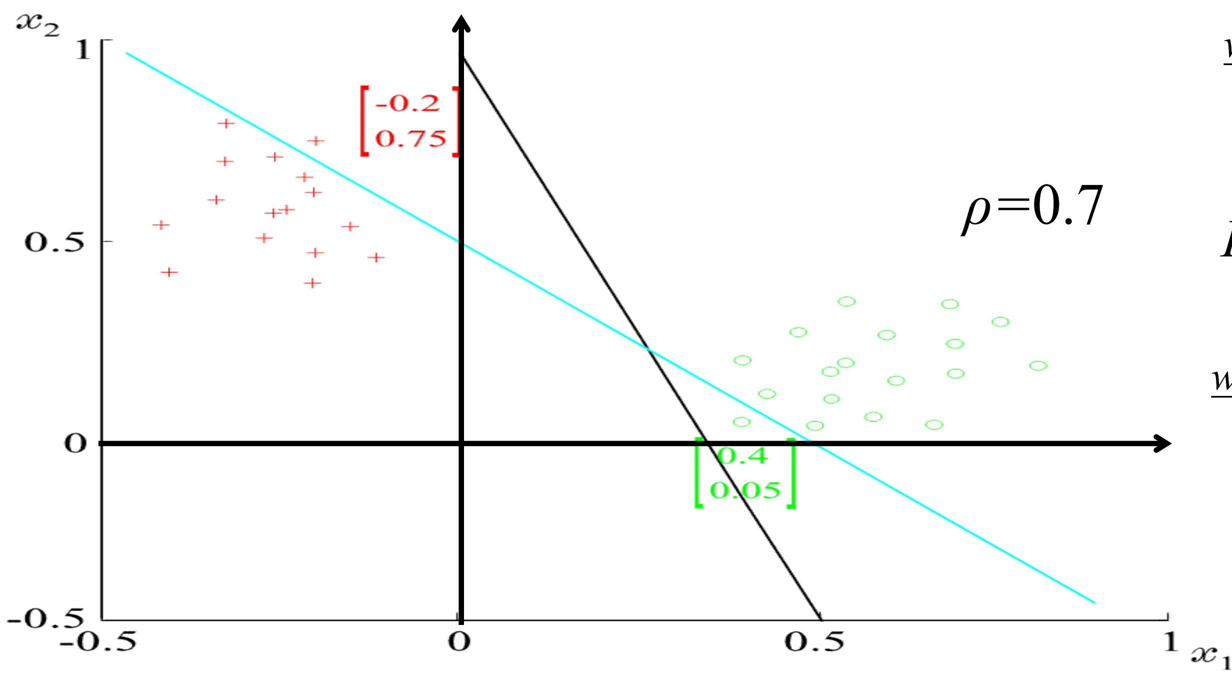
$$\underline{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1)\begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} - 0.7(+1)\begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.42 \\ 0.51 \\ -0.5 \end{bmatrix}$$

⇨ Example: At some stage t the perceptron algorithm results in

$$w_1 = 1,\ w_2 = 1,\ w_0 = -0.5$$

$$x_1 + x_2 - 0.5 = 0$$

The corresponding hyperplane is



Per-point variant:

P=[0.4, 0.05]; $w^T x<0$ wrong!

$$\underline{w}(t+1) = \begin{bmatrix} 1 \\ 1 \\ -0.5 \end{bmatrix} - 0.7(-1)\begin{bmatrix} 0.4 \\ 0.05 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.28 \\ 1.035. \\ 0.2 \end{bmatrix}$$

P=[-0.2, 0.75]; $w^T x>0$ wrong!

$$\underline{w}(t+1) = \begin{bmatrix} 1.28 \\ 1.035 \\ 0.2 \end{bmatrix} - 0.7(+1)\begin{bmatrix} -0.2 \\ 0.75 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.42 \\ 0.51. \\ -0.5 \end{bmatrix}$$

The order for processing points can be randomized

Python Exercise

- ➔Implement the perceptron algorithm in the punishement-reward form and test it for some exemplary bi-dimensional points.
- ➔Try to use the perceptron class of the sklearn module for finding the decision boundary for the same points considered in the previous case.

Python Exercise

Generate four 2-dimensional data sets X_i , $i = 1, . . . ,4$, each containing data vectors from two classes. In all X_i's the first class (denoted -1) contains 100 vectors uniformly distributed in the square $[0, 2]\times[0, 2]$. The second class (denoted $+1$) contains another 100 vectors uniformly distributed in the squares $[3, 5]\times[3, 5]$, $[2, 4]\times[2, 4]$, $[0, 2]\times[2, 4]$, and $[1, 3]\times[1, 3]$ for X1, X2, X3, and X4, respectively. Each data vector is augmented with a third coordinate that equals 1.

Perform the following steps:

- ➔ Plot the four data sets and notice that as we move from X1 to X3 the classes approach each other but remain linearly separable. In X4 the two classes overlap.
- ➔ Run the perceptron algorithm for each X_i , $i = 1, . . . ,4$, with learning rate parameters 0.01 and 0.05 and initial estimate for the parameter vector $[1, 1, -0.5]$.
- ➔ Run the perceptron algorithm for X3 with learning rate 0.05 using as initial estimates for w $[1, 1, -0.5]$ and $[1, 1, 0.5]$.
- ➔ Comment on the results.

Conclusions from previous exercise

1) First, for a fixed learning parameter, the number of iterations (in general) increases as the classes move closer to each other (i.e., as the problem becomes more difficult).

2) Second, the algorithm fails to converge for the data set X4, where the classes are not linearly separable (it runs for the maximum allowable number of iterations that we have set).

3) Third, different initial estimates for w may lead to different final estimates for it (although all of them are optimal in the sense that they separate the training data of the two classes).

Least Squares Methods

- **If classes are linearly separable, the perceptron output results in**

$$w_1, w_2, \ldots, w_0$$

- **If classes are NOT linearly separable, we shall compute the weights**

so that the difference between

- The actual output of the classifier, $\underline{w}^T \underline{x}$, and
- The desired outputs, e.g. ± 1 $\quad \begin{array}{l} +1 \text{ if } \underline{x} \in \omega_1 \\ -1 \text{ if } \underline{x} \in \omega_2 \end{array}$

to be **SMALL**

⇨ **SMALL**, in the **mean square** error sense, means to choose $\underline{w}$ so that the cost function

→ $J(\underline{w}) \equiv E[(y - \underline{w}^T \underline{x})^2]$ is minimum

→ $\hat{\underline{w}} = \arg\min_{\underline{w}} J(\underline{w})$

→ y the corresponding desired responses

⇨ Minimizing

$J(\underline{w})$ w.r. to $\underline{w}$ results in :

$$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} E[(y - \underline{w}^T x)^2] = 0$$

$$= 2E[\underline{x}(y - \underline{x}^T \underline{w})] \Rightarrow$$

$$E[\underline{x}\underline{x}^T]\underline{w} = E[\underline{x}y] \Rightarrow$$

$$\boxed{\hat{\underline{w}} = R_x^{-1} E[\underline{x}y]}$$

But who knows statistical distribution of feature vectors???

where R_x is the **autocorrelation matrix**

$$R_x \equiv E[\underline{x}\underline{x}^T] = \begin{bmatrix} E[x_1x_1] & E[x_1x_2]\ldots & E[x_1x_l] \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ E[x_lx_1] & E[x_lx_2]\ldots & E[x_lx_l] \end{bmatrix}$$

and $E[\underline{x}y] = \begin{bmatrix} E[x_1 y] \\ \ldots \\ E[x_l y] \end{bmatrix}$ the **crosscorrelation vector**

Other cost functions

➔ **<u>SMALL</u> in the <u>sum of error</u> squares sense means**

⇨ $$J(\underline{w}) = \sum_{i=1}^{N} (y_i - \underline{w}^T \underline{x}_i)^2$$

$(y_i, \underline{x}_i)$: training pairs that is, the input $\underline{x}_i$ and its corresponding class label $y_i (\pm 1)$.

⇨ $$\frac{\partial J(\underline{w})}{\partial \underline{w}} = \frac{\partial}{\partial \underline{w}} \sum_{i=1}^{N} (y_i - \underline{w}^T \underline{x}_i)^2 = 0 \Rightarrow$$

$$\boxed{(\sum_{i=1}^{N} \underline{x}_i \underline{x}_i^T)\underline{w} = \sum_{i=1}^{N} \underline{x}_i y_i}$$

➔Pseudoinverse Matrix

⇨ Define

$$X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \ldots \\ \underline{x}_N^T \end{bmatrix} \quad \text{(an } N x l \text{ matrix)}$$

$$\underline{y} = \begin{bmatrix} y_1 \\ \ldots \\ y_N \end{bmatrix} \text{ corresponding desired responses}$$

⇨ $$X^T = [\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N] \quad \text{(an } l x N \text{ matrix)}$$

⇨ $$X^T X = \sum_{i=1}^{N} \underline{x}_i \underline{x}_i^T \qquad X^T \underline{y} = \sum_{i=1}^{N} \underline{x}_i y_i$$

Thus $(\sum_{i=1}^{N} \underline{x}_i^T \underline{x}_i)\underline{\hat{w}} = (\sum_{i=1}^{N} \underline{x}_i y_i)$

$$(X^T X)\underline{\hat{w}} = X^T \underline{y} \Rightarrow$$

$$\underline{\hat{w}} = (X^T X)^{-1} X^T \underline{y}$$

$$= X^{\neq} \underline{y}$$

$\boxed{X^{\neq} \equiv (X^T X)^{-1} X^T}$ **Pseudoinverse of *X***

⇨ Assume $N=l$ $\Rightarrow$ X square and invertible. Then

$$(X^T X)^{-1} X^T = X^{-1} X^{-T} X^T = X^{-1} \Rightarrow$$

$$\boxed{X^{\neq} = X^{-1}}$$

⇨ Assume $N>l$. Then, in general, there is no solution to satisfy all equations simultaneously:

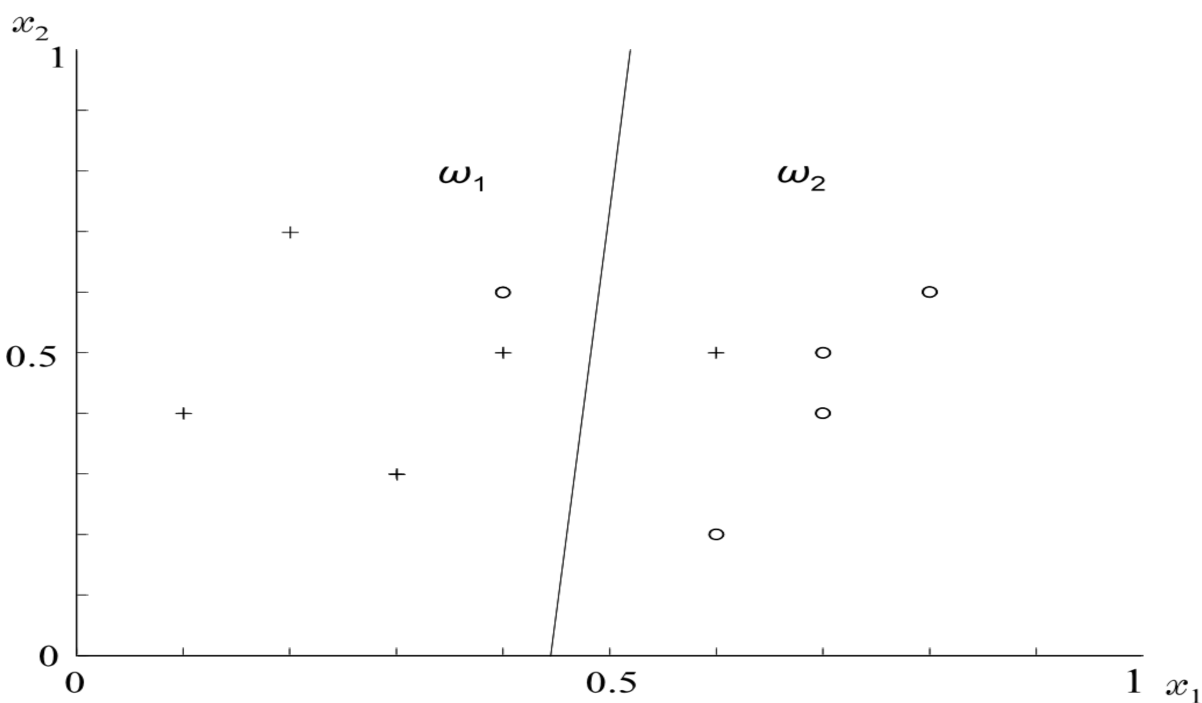
$$X\underline{w}=\underline{y}: \quad \begin{array}{c} \underline{x}_1^T\underline{w}=y_1 \\ \underline{x}_2^T\underline{w}=y_2 \\ \cdots\cdot \\ \underline{x}_N^T\underline{w}=y_N \end{array} \qquad N \text{ equations} > l \text{ unknowns}$$

⇨ The "solution" $\underline{w}=X^{\neq}\underline{y}$ corresponds to the minimum sum of squares solution

⇨ Example:

$$\omega_1 : \begin{bmatrix} 0.4 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.2 \\ 0.7 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$$

$$\omega_2 : \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}$$



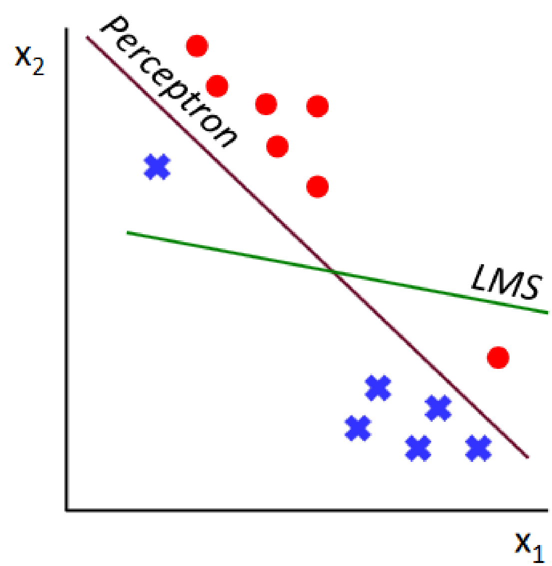
$$X = \begin{bmatrix} 0.4 & 0.5 & 1 \\ 0.6 & 0.5 & 1 \\ 0.1 & 0.4 & 1 \\ 0.2 & 0.7 & 1 \\ 0.3 & 0.3 & 1 \\ 0.4 & 0.6 & 1 \\ 0.6 & 0.2 & 1 \\ 0.7 & 0.4 & 1 \\ 0.8 & 0.6 & 1 \\ 0.7 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} = \underline{y}$$

$$\Rightarrow \quad X^T X = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix}, X^T \underline{y} = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

$$\underline{w} = (X^T X)^{-1} X^T \underline{y} = \begin{bmatrix} -3.13 \\ 0.24 \\ 1.34 \end{bmatrix}$$

Error minimization vs. gradient

- ➔ **Perceptron rule**
  - ⇨ Always finds a solution if classes are separable, but does not converge if not
- ➔ **Error minimization**
  - ⇨ LSE solution has guaranteed convergence, but it may not find a separating hyperplane if classes are separable
    - → It minimizes the sum of the distances from the separating hyperplane



Multi Class Generalization

- ➔ **Idea: using M parallel binary classifiers for each possible class (does $\underline{x}$ belong to class i? yes/not)**
- ➔ **The goal is to compute M linear discriminant functions:**

$$g_i(\underline{x}) = \underline{w}_i^T \underline{x}$$

$$y_i = 1 \quad \text{if} \quad \underline{x} \in \omega_i$$
$$y_i = 0 \quad \text{otherwise}$$

⇨ according to the MSE.

→Adopt as desired responses y_i:

→Let $\underline{y} = \left[y_1, y_2, \ldots, y_M\right]^T$

→And the matrix $W = \left[\underline{w}_1, \underline{w}_2, \ldots, \underline{w}_M\right]$

How to compute W?

➔ **We need to solve a number M of MSE minimization problems. That is:**

$$\hat{W} = \arg\min_{W} E\left[\left\| \underline{y} - W^T \underline{x} \right\|^2\right] = \arg\min_{W} E\left[\sum_{i=1}^{M} \left(y_i - \underline{w}_i^T \cdot \underline{x}\right)^2\right]$$

Design each $\underline{w}_i$ so that its desired output is 1 for $\underline{x} \in \omega_i$ and 0 for any other class.

⇨ **Remark:** The MSE criterion belongs to a more general class of cost function with the following **important** property:

→The value of $g_i(\underline{x})$ is an estimate, in the MSE sense, of the a-posteriori probability $P(\omega_i \mid \underline{x})$, provided that the desired responses used during training are $y_i = 1, \underline{x} \in \omega_i$ and 0 otherwise.

Other solutions?

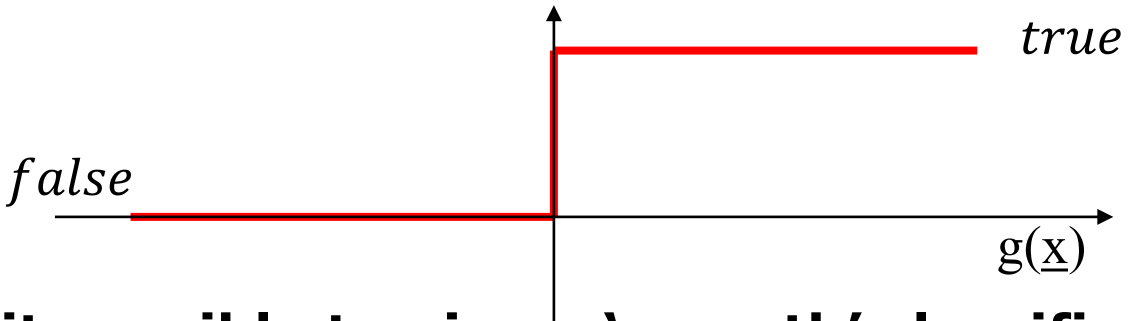
- ➔ Our current approach: **one-vs-rest**
  - ⇨ For each class ω, make a binary classifier to distinguish from other classes
  - ⇨ If there are M classes, there are M classifiers
  - ⇨ At test time, we select the class giving the highest g_i(x)
- ➔ **One-vs-one**
  - ⇨ For each pair of classes ω1 and ω2, make a binary classifier
  - ⇨ If there are M classes, there are M(M-1)/2 classifiers
  - ⇨ At test time, we select the class that has most wins!

In Scikit-learn

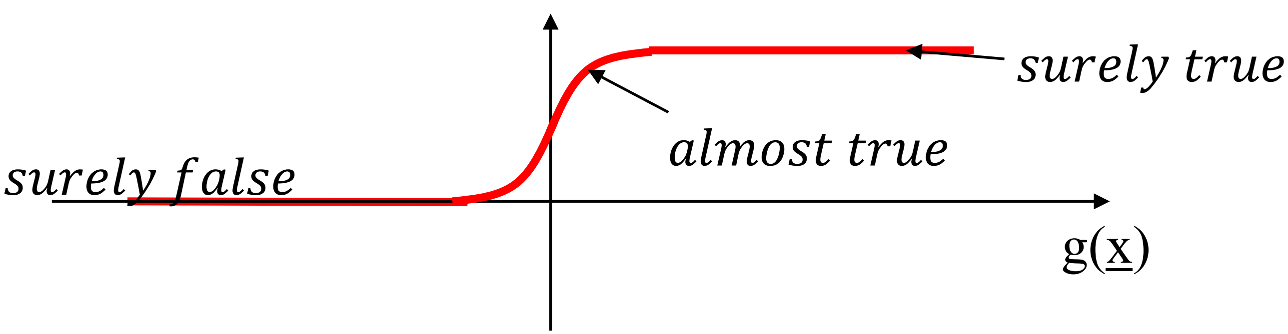
- ➔ **It includes implementations of both the methods**
  - ⇨ OneVsRestClassifier
  - ⇨ OneVsOneClassifier
- ➔ **However, the built-in algorithms (e.g. Perceptron) will automatically work on the one-vs-rest approach**
  - ⇨ Try the Perceptron classifier on the iris dataset, which includes three classes!!!

Is classification always YES/NOT?

- **Until now we have considered a deterministic classification output whose output is true/false**
  - $g(\underline{x})>0 \rightarrow \underline{x} \in \omega 1$, $g(\underline{x})<0 \rightarrow \underline{x} \notin \omega 1$



- **Is it possible to give a 'smooth' classification, measuring the uncertainness of the output?**
  - Or to use some training data with heterogeneous 'confidence'?



Logistic function

➔ **A possible S function mapping real numbers to [0,1] (to be associated to linear classifiers) is the logistic function:**

⇨ $$\frac{1}{1+e^{-g(\underline{x})}} = \frac{1}{1+e^{-(w_0+w_1x_1+..w_lx_l)}}$$

→ when g(x) goes to minus infinity, the classifier output is almost 0, when it goes to plus infinity the classifier output is almost 1

➔ **In this case the output of the classifier is not binary (TRUE/FALSE) but it is a real value**

⇨ Can we interpret this value as a probability?

→ $P(\omega 1/x)=1/(1+e^{-g(\underline{x})})$ and $P(!\omega 1/x)=e^{-g(x)}/(1+e^{-g(x)})=1/(1+e^{g(\underline{x})})$

⇨ It follows that g(x) meaning is:

→ $g(x)=\ln(P(\omega 1/x)/(1-P(\omega 1/x))=\ln(P(\omega 1/x)/P(\omega 2/x))$

⇨ And in a more compact form: $P(\omega\_i/x)=1/(1+e^{-y\_i\, g(\underline{x})})$

Generalization

➔ **Let an M-class task, $\omega_1, \omega_2 .. \omega_M$. In logistic discrimination, the logarithm of <u>the likelihood ratios</u> towards a reference class ω_M are modeled via linear functions, i.e.,**

$$\ln\left(\frac{P(\omega_i \mid \underline{x})}{P(\omega_M \mid \underline{x})}\right) = w_{i,0} + \underline{w}_i^T \underline{x}, \;\; i = 1,\ 2,\ \ldots,\ M\text{-}1$$

⇨ Taking into account that $\sum_{i=1}^{M} P(\omega_i \mid \underline{x}) = 1$

it can be easily shown that the above is equivalent with modeling posterior probabilities as:

$$P(\omega_M \mid \underline{x}) = \frac{1}{1+\sum_{i=1}^{M-1}\exp\left(w_{i,0}+\underline{w}_i^T \underline{x}\right)} = \frac{1}{1+e^{g_1(\underline{x})}+e^{g_2(\underline{x})}+\ldots e^{g_{M-1}(\underline{x})}}$$

$$P(\omega_i \mid \underline{x}) = \frac{\exp\left(w_{i,0}+\underline{w}_i^T \underline{x}\right)}{1+\sum_{i=1}^{M-1}\exp\left(w_{i,0}+\underline{w}_i^T \underline{x}\right)} = \frac{e^{g_i(\underline{x})}}{1+e^{g_1(\underline{x})}+e^{g_2(\underline{x})}+..e^{g_{M-1}(\underline{x})}}, i = 1,2,\ldots \mathrm{M}-1$$

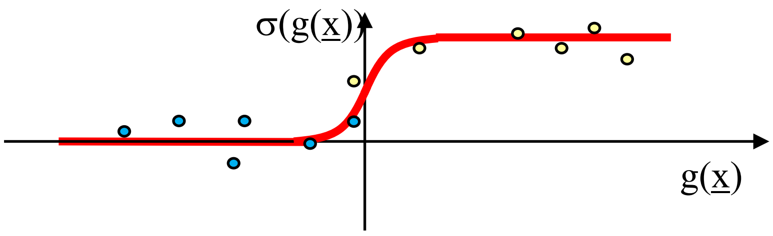
⇨ For the two-class case we find our previous results

$$P(\omega_2 \mid \underline{x}) = \frac{1}{1+\exp\left(w_0+\underline{w}^T \underline{x}\right)}$$

$$P(\omega_1 \mid \underline{x}) = \frac{\exp\left(w_0+\underline{w}^T \underline{x}\right)}{1+\exp\left(w_0+\underline{w}^T \underline{x}\right)}$$

Can we train a logistic classifier?

- ➔ **Data sets for training are not in the form (x, y), but in the form (x, Pr(y))**
  - ⇨ Classifier assigns class y, if Pr(y) is higher than 0.5
- ➔ **Classifier goal: finding the w vector for which g(x) fits the logistic function σ(g(x)) for the training data**
  - ⇨ How? Different approaches based on maximum likehood arguments for minimizing a cost function J(w) with usual gradient methods
  - ⇨ e.g. $-\sum_{i}^{N} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))$



  - ⇨ Logistic discrimination is a useful tool, since it allows linear modeling and at the same time ensures posterior probabilities to add to one.

Logist Regression Loss Function

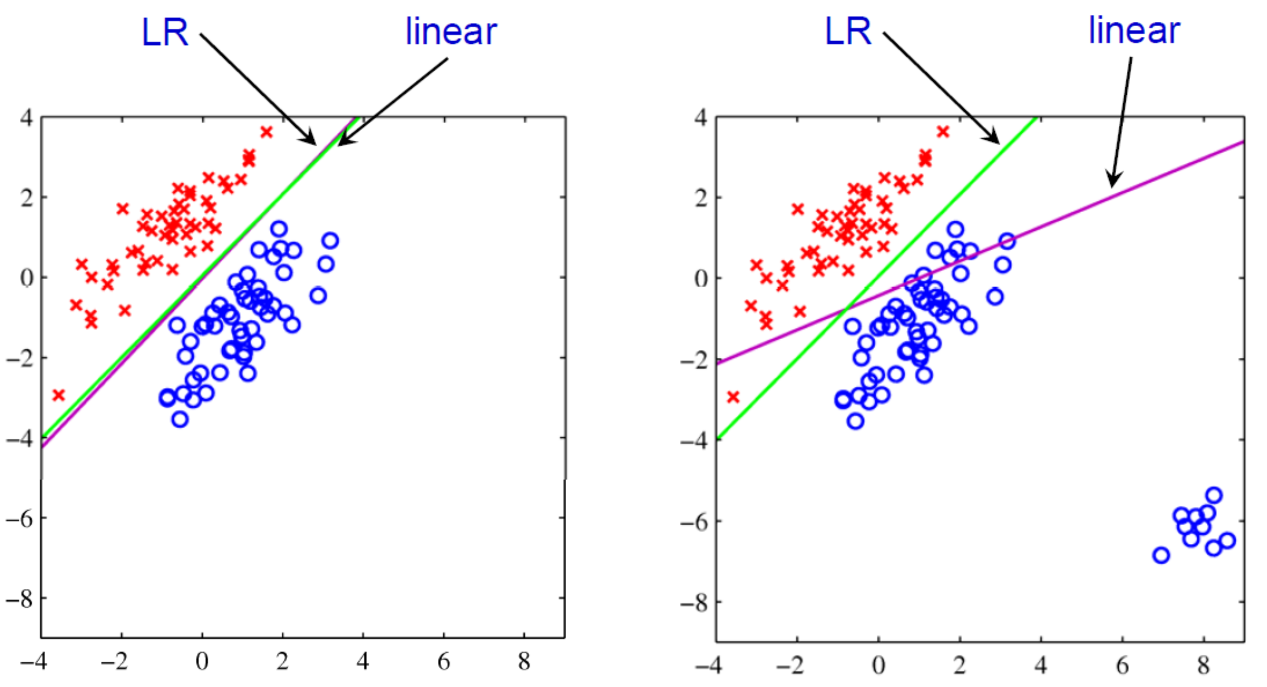
- **Assume**
  - $p(y=1|\underline{x}, \underline{w})=\sigma(\underline{w}^T\underline{x})$
  - $p(y=0|\underline{x}, \underline{w})=1-\sigma(\underline{w}^T\underline{x})$
- **In compact form**
  - $p(y|\underline{x}, \underline{w})=(\sigma(\underline{w}^T\underline{x}))^y(1-\sigma(\underline{w}^T\underline{x}))^{(1-y)}$
- **Assuming to have N independent data**
  - $p(y|\underline{x}, \underline{w})=\prod_i^N (\sigma(\underline{w}^T\underline{x}_i))^y(1-\sigma(\underline{w}^T\underline{x}_i))^{(1-y)}$
- **The negative log likehood is:**
  - $J(w)=-\sum_i^N y_i \log\sigma(w^T x_i) + (1-y_i)\log(1-\sigma(w^T x_i))$
  - $J(w)=\sum_i^N \log(1+e^{-y_i f(x_i)})$
  - Training : $$\min_{\mathbf{w}\in\mathbb{R}^d} \sum_i^N \underbrace{\log\left(1+e^{-y_i f(\mathbf{x}_i)}\right)}_{\text{loss function}}$$

Python exercise

- ➔ **Load and run the regression example on the Iris Dataset**
  - ⇨ Rewrite it for two classes only, by filtering the training data
  - ⇨ Compare the results with other models based on perceptrons

Python Exercise

- ➔ **Generates two sets of random points clustered into two areas; add a far cluster to one of the two sets**
  - ⇨ Compare linear regression and logistic regression

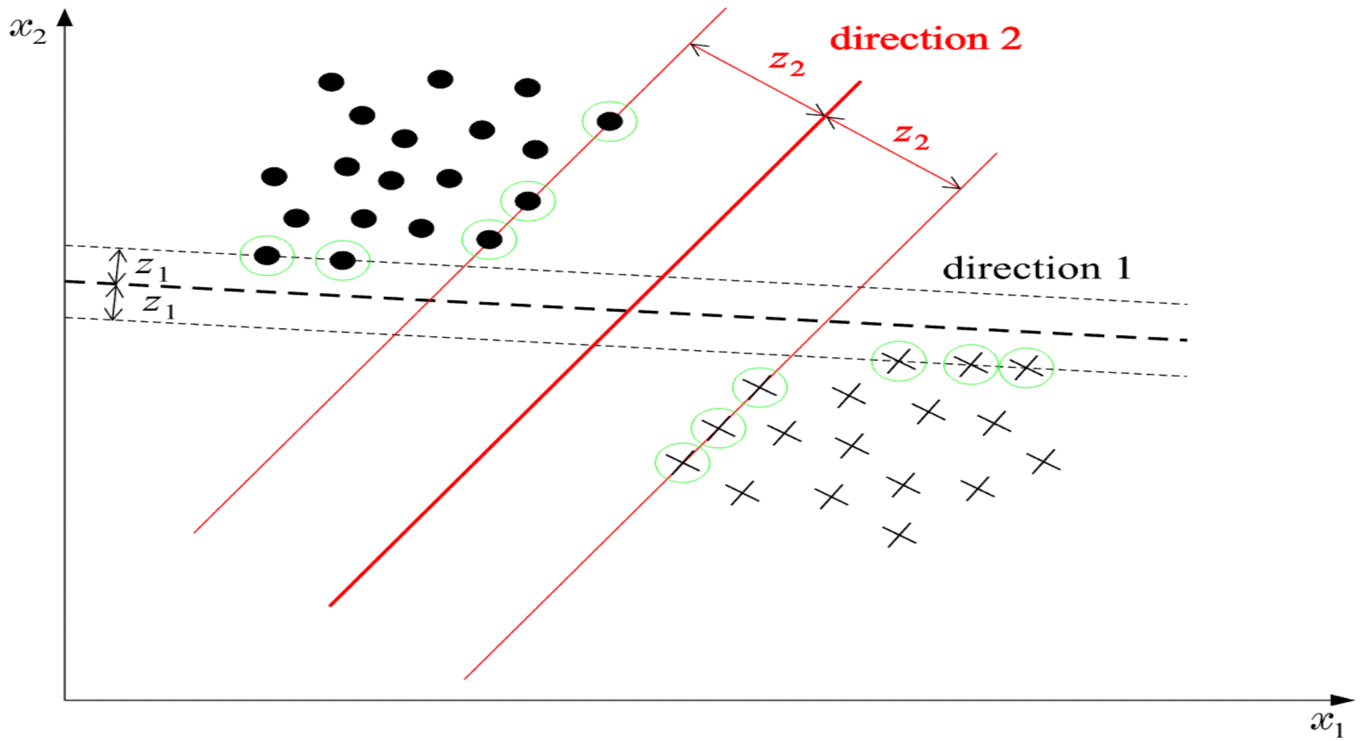


Support Vector Machines

➔ **The goal: Given two linearly separable classes, design the classifier**

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

that leaves the **maximum margin** **from both classes**



⇨ **Margin**: Each hyperplane is characterized by

→ Its direction in space, i.e., $\underline{w}$

→ Its position in space, i.e., w_0

→ For EACH direction, $\underline{w}$, choose the hyperplane that leaves the SAME distance from the nearest points from each class. The margin is twice this distance.

⇨ The distance of a point $\hat{\underline{x}}$ from a hyperplane is given by

$$z_{\hat{x}} = \frac{g(\hat{\underline{x}})}{\|\underline{w}\|}$$

⇨ Scale, $\underline{w}$, $\underline{w}_0$, so that at the nearest points from each class the discriminant function is ±1:

$$|g(\underline{x})| = 1 \ \{g(\underline{\mathrm{x}}) = +1 \text{ for } \omega_1 \text{ and } g(\underline{x}) = -1 \text{ for } \omega_2\}$$

⇨ Thus the margin is given by

$$\frac{1}{\|\underline{w}\|} + \frac{1}{\|\underline{w}\|} = \frac{2}{\|\underline{w}\|}$$

⇨ Also, the following is valid

$$\underline{w}^T \underline{x} + w_0 \geq 1 \ \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} + w_0 \leq -1 \ \forall \underline{x} \in \omega_2$$

⇨ SVM (linear) classifier

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

⇨ Minimize

$$J(\underline{w}) = \frac{1}{2} \|\underline{w}\|^2$$

⇨ Subject to

$$y_i(\underline{w}^T \underline{x}_i + w_0) \geq 1, \ i = 1,2,\ldots,N$$

$$y_i = 1, \text{for } \underline{x}_i \in \omega_i,$$

$$y_i = -1, \text{for } \underline{x}_i \in \omega_2$$

⇨ The above is justified since by minimizing $\|\underline{w}\|$

the margin $\dfrac{2}{\|w\|}$ is maximised

⇨ The above is a quadratic optimization task, subject to a set of linear inequality constraints. The Karush-Kuhh-Tucker conditions state that the minimizer satisfies:

→(1) $\frac{\partial}{\partial \underline{w}} \mathrm{L}(\underline{w}, w_0, \underline{\lambda}) = \underline{0}$

→(2) $\frac{\partial}{\partial w_0} L(\underline{w}, w_0, \underline{\lambda}) = 0$

→(3) $\lambda_i \geq 0, i = 1,2,\ldots, N$

→(4) $\lambda_i \left[y_i (\underline{w}^T \underline{x}_i + w_0) - 1 \right] = 0, i = 1,2,\ldots, N$

→Where $L(\bullet,\bullet,\bullet)$ is the Lagrangian

$$L(\underline{w}, w_0, \underline{\lambda}) \equiv \frac{1}{2} \underline{w}^T \underline{w} - \sum_{i=1}^{N} \lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0)]$$

⇨ The solution: from the above, it turns out that

→ $$\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

→ $$\sum_{i=1}^{N} \lambda_i y_i = 0$$

⇨ Remarks:

→ The Lagrange multipliers can be either zero or positive. Thus,

» $$\underline{w} = \sum_{i=1}^{N_s} \lambda_i y_i \underline{x}_i$$

where $N_s \leq N$, corresponding to positive Lagrange multipliers

» From constraint (4) above, i.e.,

$$\lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0) - 1] = 0, \quad i = 1, 2, \ldots, N$$

the vectors contributing to satisfy $\underline{w}$

$$\underline{w}^T \underline{x}_i + w_0 = \pm 1$$

» These vectors are known as SUPPORT VECTORS and are the closest vectors, from each class, to the classifier.

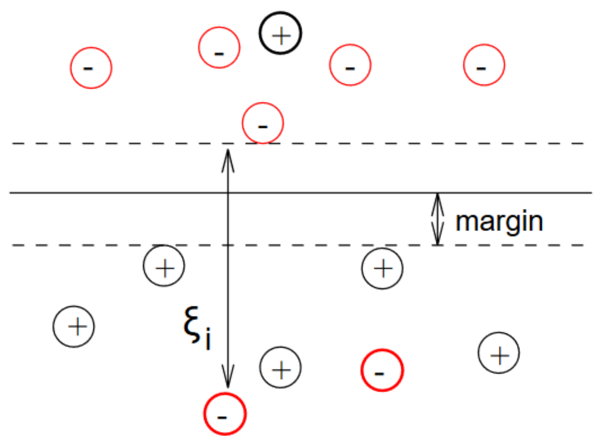
» Once $\underline{w}$ is computed, w_0 is determined from conditions (4).

» The optimal hyperplane classifier of a support vector machine is UNIQUE.

» Although the solution is unique, the resulting Lagrange multipliers are not unique.

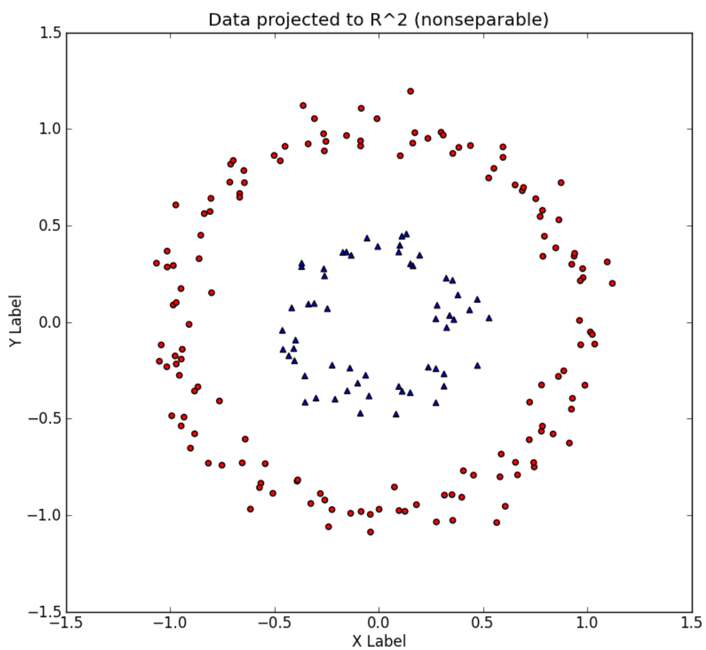
Allowing for errors

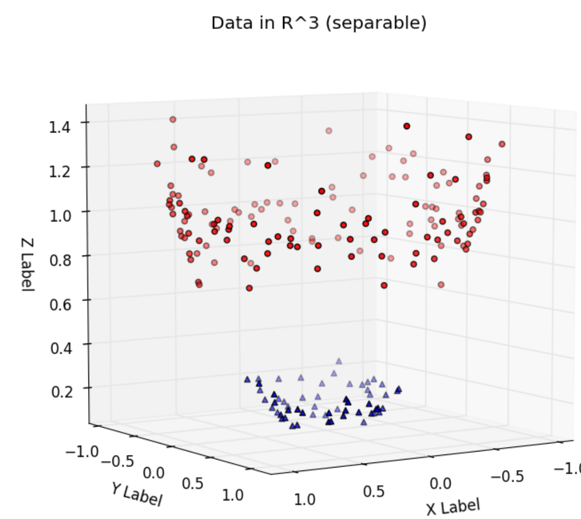
- ➔ **To deal with the non-separable case, one can rewrite the problem as:**
  - ⇨ Minimize $||w||^2 + C \sum_{i=1}^{N} \xi_i$
  - ⇨ Subject to: $y_i(w \cdot x_i + w_o) \geq 1 - \xi_i, \quad \xi_i > 0$



Dealing with non-separable classes

➔ **Using kernel functions to add other features for increasing the feature space dimension and making the problem linearly separable**





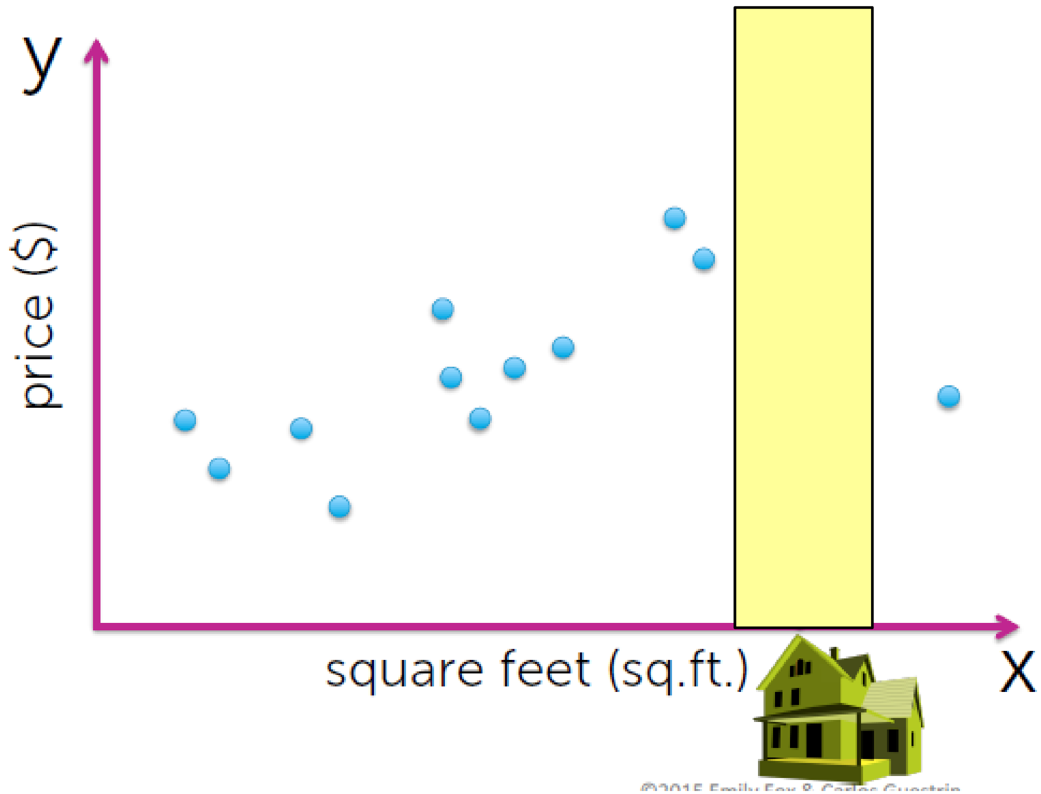
Python example

- ➔ **Look at the SVM.py example.**
- ➔ **What is the effect of the C parameter used in the SVM model provided by scikit?**
  - ⇨ Softing the margins! Why??

Linear model generalizations

- ➔ **Is it possible to fit $\underline{w}^T\underline{x}=\underline{y}$, where $\underline{y}$ is not a binary vectore of +/-1, but may assume general values?**
  - ⇨ Yes! In this case y can predict not a discrete class but a value associated to the data point
  - ⇨ e.g. build a predictor for house prices, given some observations of [(place, size, #rooms), cost] sets

An illustrative example

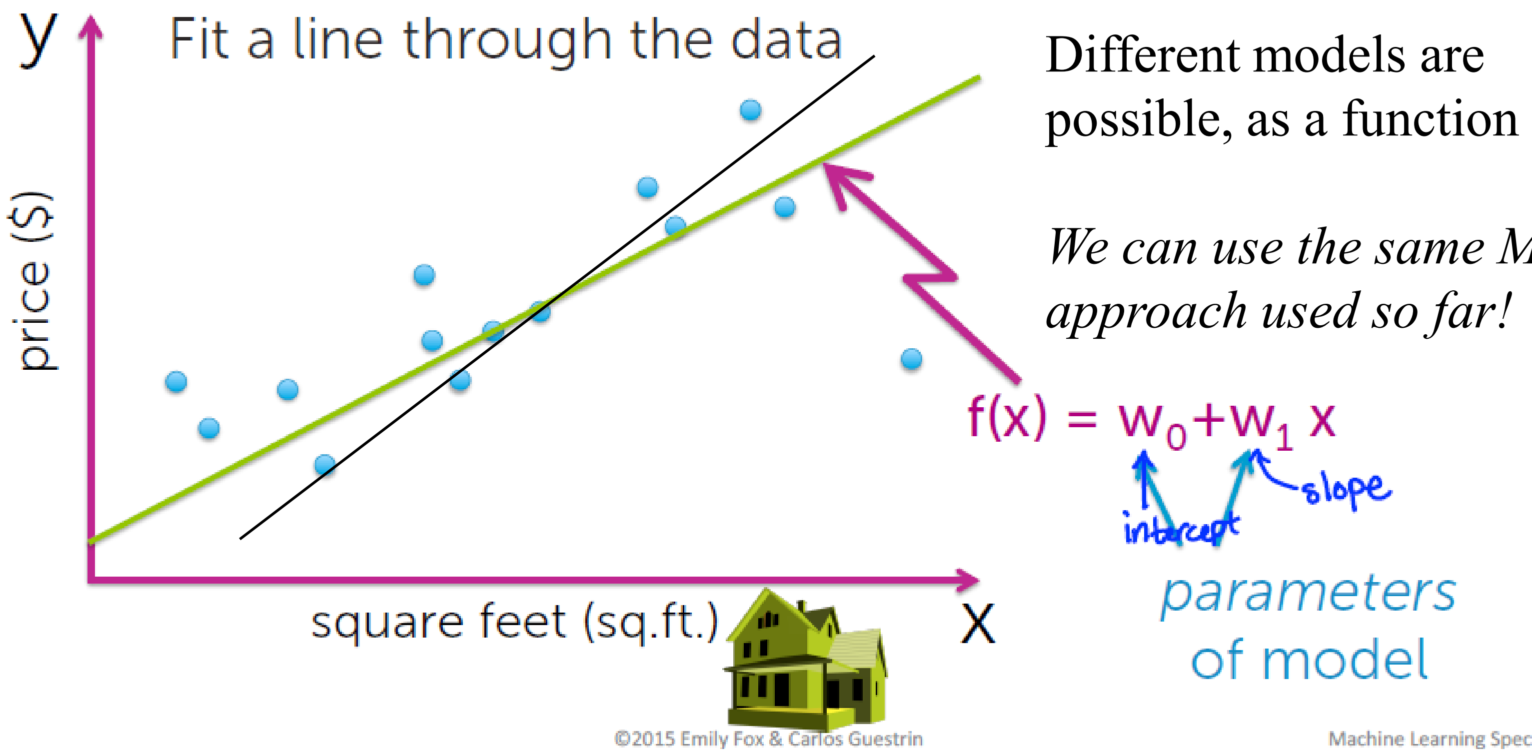


No house sold recently had *exactly* the same sq.ft.

- Look at average price in range
- **Still only 2 houses!**
- Throwing out info from all other sales

7

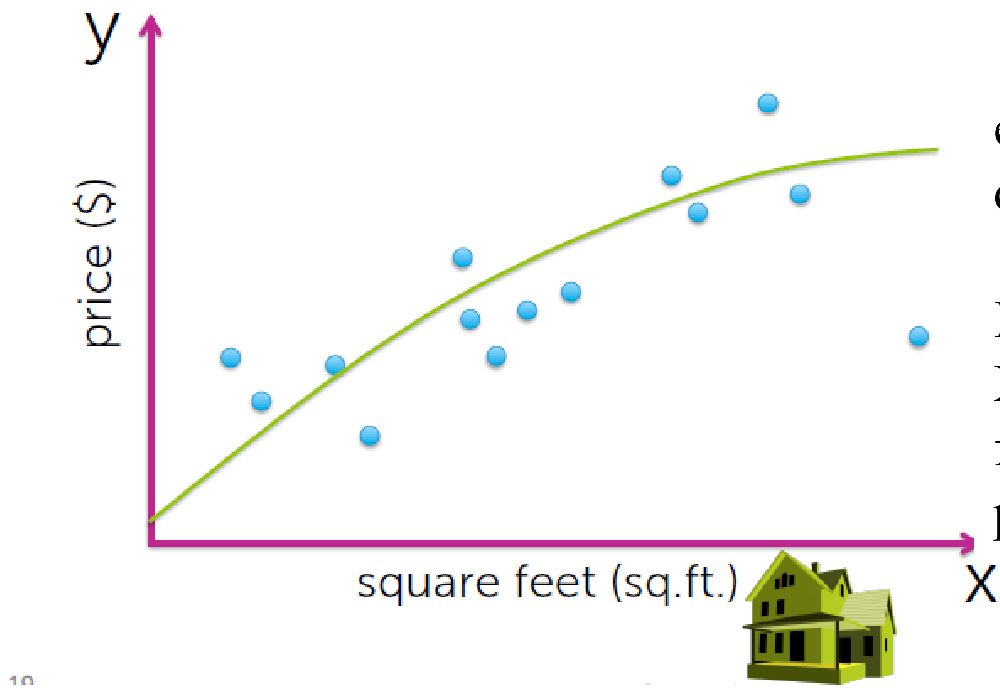
Can we do better than comparing only close points?



Different models are possible, as a function of <u>w;</u>

We can use the same MSE approach used so far!

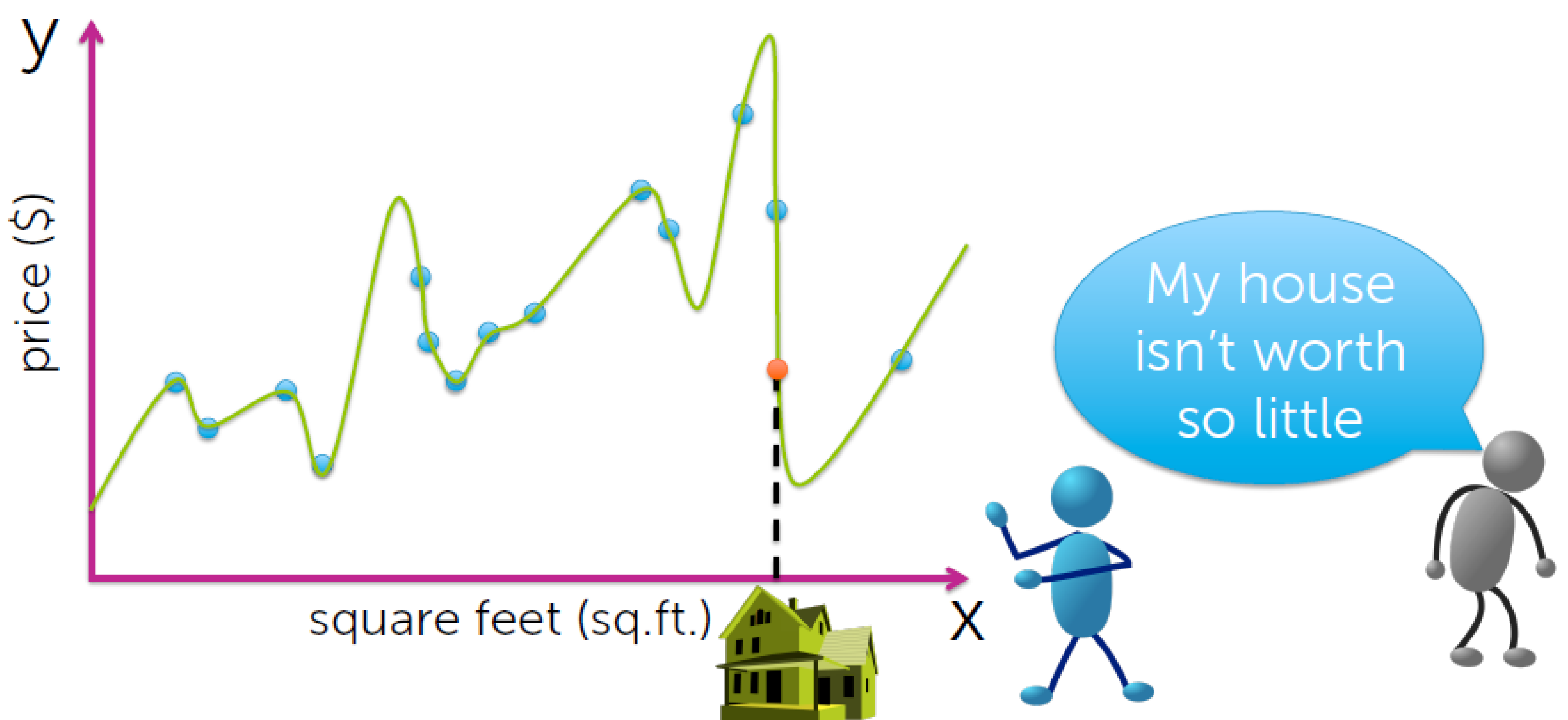
More complex models are possible!



e.g. $f(x)=w0+w1\ x1 + w2\ x1^2$
quadratic of polinomial models!

In all the cases, we can solve the $X\underline{w}=\underline{y}$ MSE problem, by adding features corresponding to powers of each component x_i

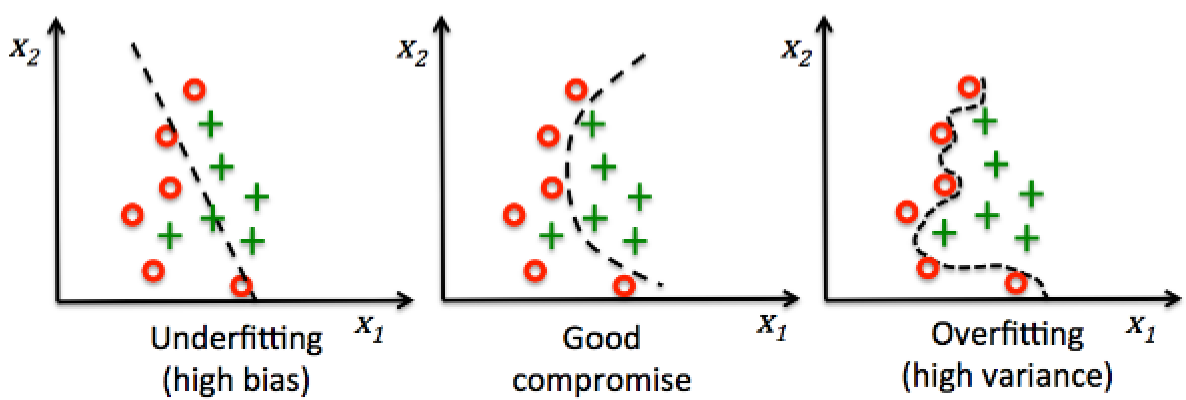
More complex, more accurate?



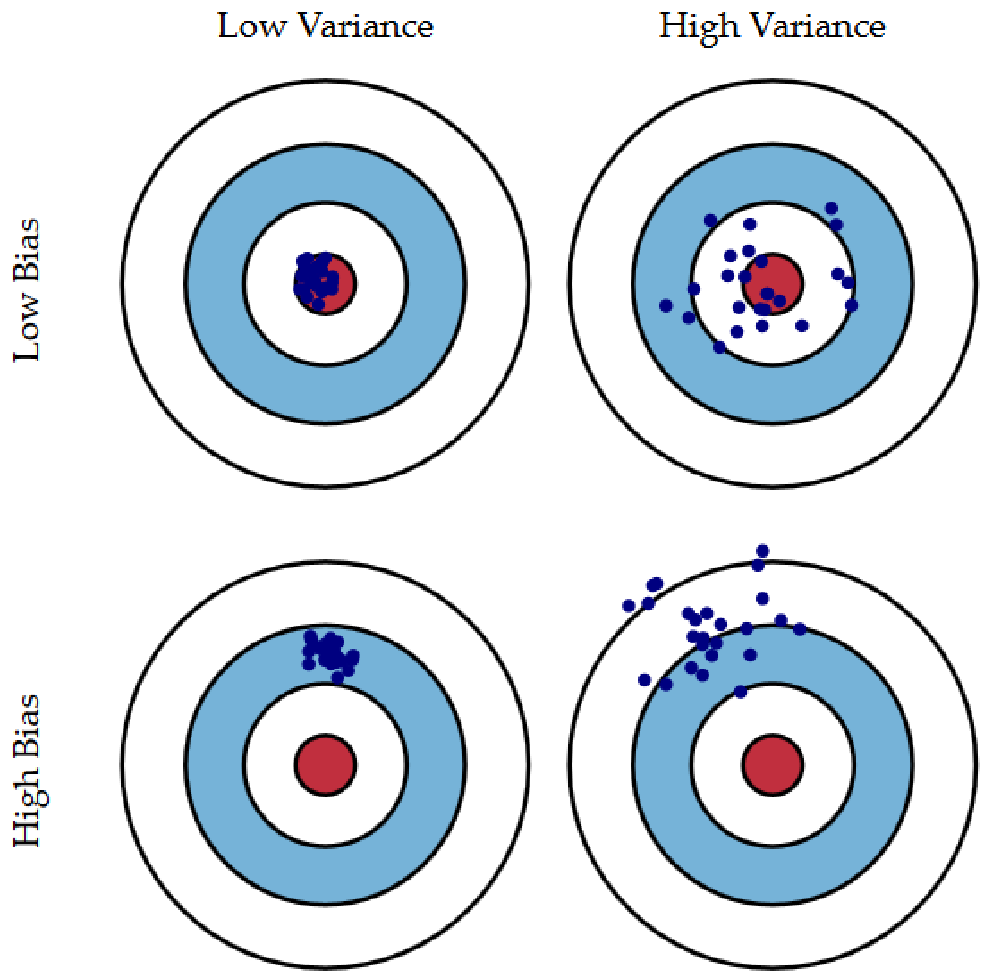
Bias Variance Dilemma:

what does it mean?

- ➔ **How much our model depend on the training data? And how complex should be?**
- ➔ **An illustrative example:**
  - ⇨ Is it fine to use a linear classifier for these points? Or am I losing some data behaviors?
  - ⇨ How does perform an extreme complex model which strongly depends on the training data? Which generalization capabilities?
    - → Polinomial functions g(x) can be obtained as linear functions of feature powers



Variance and Bias



Bias-Variance Dilemma

- ➔ **A classifier g($\underline{x}$) is a learning machine that tries to predict the class label y of $\underline{x}$.**
- ➔ **In practice, a finite data set D is used for its training. Let us write g($\underline{x}$, D). Observe that:**

  - ⇨ For some training sets, $D=\{(y_i,\underline{x}_i),i=1,2,...,N\}$ the training may result to good estimates, for some others the result may be worse.

  - ⇨ The average performance of the classifier can be tested against the MSE optimal value, in the mean squares sense, that is:

$$E_D\left[\left(g(\underline{x};D)-E[y\,|\,\underline{x}]\right)^2\right]$$

where E_D is the mean over all possible data sets D.

→The above is written as:

$$E_D\left[\left(g(\underline{x};D)-E[y\mid \underline{x}]\right)^2\right]=$$

$$\left(E_D\left[g(\underline{x};D)\right]-E[y\mid \underline{x}]\right)^2+E_D\left[\left(g(\underline{x};D)-E_D\left[g(\underline{x};D)\right]\right)^2\right]$$

→In the above, the first term is the contribution of the bias and the second term is the contribution of the variance.

→For a finite *D*, there is a trade-off between the two terms. Increasing bias it reduces variance and vice verse. This is known as the bias-variance dilemma.

→Using a complex model results in low-bias but a high variance, as one changes from one training set to another. Using a simple model results in high bias but low variance.

Complexity and Generalization