Lecture 7: Linear regression

Machine Learning 2025

Federico Chiariotti (federico.chiariotti@unipd.it)

Università degli Studi di Padova

Recap

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Affine functions

Class of affine functions:

$$L_d = \left\{ h_{\mathbf{w},b}, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

Each function h is:

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = b + \sum_{i=1}^{d} w_i x_i$$

Affine functions are linear functions with a sum. There are d+1 parameters (d weights, plus the bias)

Linear hypotheses

Binary classification:

Sign of the affine function

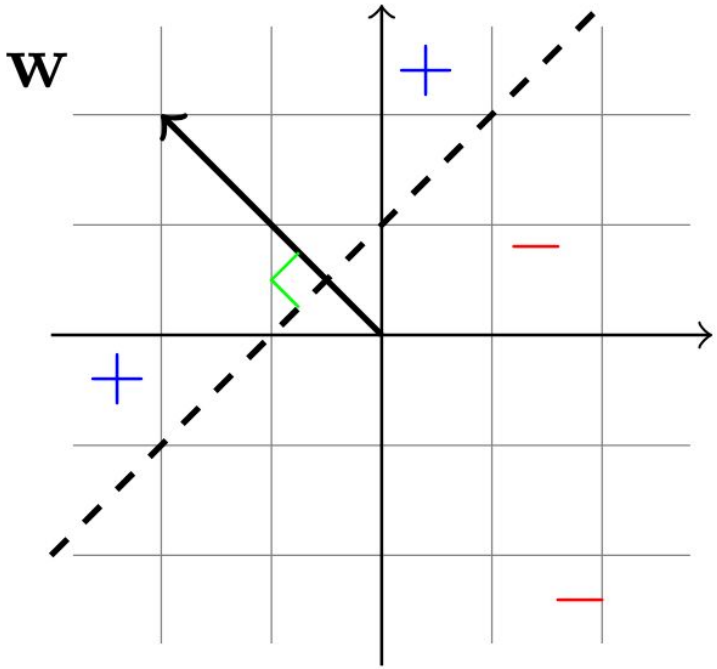
$$h_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

Classification:

Direct affine function

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

A 2D example



Separating (hyper)plane:

$$w_1 x_1 + w_2 x_2 + b = 0$$

Plane definition:

$$x_2 = m x_1 + h$$

Intercept:

$$h = -\frac{b}{w_2}$$

Slope:

$$m = \frac{w_1}{w_2}$$

The weight vector is orthogonal to the hyperplane

Homogeneous coordinates

We can incorporate the bias into the weights by adding one dimension

$$\mathbf{x}' = (1, x_1, x_2, \ldots, x_d) \qquad \mathbf{w}' = (b, w_1, w_2, \ldots, w_d)$$

In this way, the affine function becomes a linear function

$$h_{\mathbf{w},b}(\mathbf{x}) = b + \sum_{i=1}^{d} w_i x_i \qquad\qquad h_{\mathbf{w}'}(\mathbf{x}') = 1 \times b + \sum_{i=1}^{d} w_i x_i$$

The result is exactly the same, but the notation is much more compact

Linear programming as ERM

Linear programming (LP) is an optimization framework that can be solved efficiently. We can pose the ERM rule as a constraint, using a dummy objective

$$\begin{aligned} &\text{maximize } 1 \\ &\text{such that } \mathbf{A}\mathbf{w} \geq (1, \ldots, 1)^T \end{aligned}$$

Each row of matrix **A** is simply given by $y_i \mathbf{x}_i$

There are optimized solvers for LP (e.g., simplex), but we don't need to implement them here and now

The perceptron algorithm

Start: $\mathbf{w}^{(0)} = (0, \ldots, 0)$

Repeat until convergence:

1. Find a sample with $\text{sign}(\langle \mathbf{w}^{(i)}, \mathbf{x}_m \rangle) = -y_m$
2. Compute the new $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \alpha \mathbf{x}_m y_m$

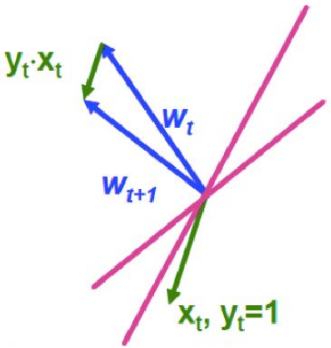
Parameter α is the learning rate (if omitted, it is set to 1)

Termination depends on realizability: if the training set is not linearly separable, the perceptron will go on forever

Pseudocode

Input: training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
initialize $\mathbf{w}^{(1)} = (0, \ldots, 0)$;
for $t = 1, 2, \ldots$ **do**
  if $\exists i$ *s.t.* $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ **then** $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$;
  else return $\mathbf{w}^{(t)}$;

Interpretation of update:



Note that:

$$y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle$$
$$= y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle + ||\mathbf{x}_i||^2$$

$\Rightarrow$ update guides $\mathbf{w}$ to be "more correct" on $(\mathbf{x}_i, y_i)$.

Part 1:
Linear regression

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

The linear regression problem

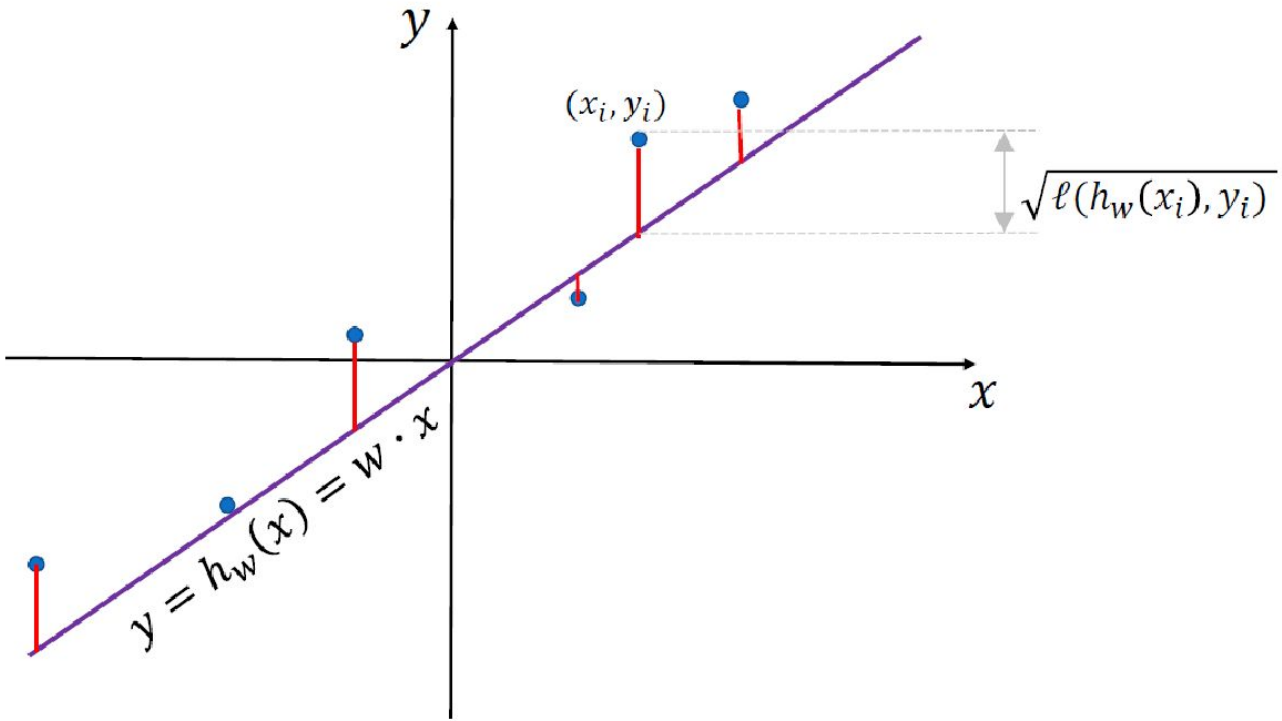
Domain: $X \in \mathbb{R}^d,\ Y \in \mathbb{R}$

Hypothesis class: $H_{\text{reg}} = L_d = \{\mathbf{x} \to \langle \mathbf{w}, \mathbf{x} \rangle + b, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$

Loss function: commonly, L2 loss $\mathbb{E}_D \left[(\langle \mathbf{w}, \mathbf{x} + b - y)^2 \right]$

Empirical risk: Mean Square Error (MSE) $L_S(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i + b - y_i)^2$

The linear regression problem



ERM: least squares approach

The ERM approach to regression is solved by the least squares algorithm

$$L_S(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i + b - y_i)^2$$

The square is a quadratic function, so we can treat it as a convex problem: it has a single minimum!

Least squares derivation (first step)

The minimum coincides with the zero-gradient point

$$\frac{\partial L_S}{\partial \mathbf{w}} = \frac{2}{m} \sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i) \mathbf{x}_i$$

We then need to solve the following equation for w:

$$\frac{2}{m} \sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i) = 0$$

This is equivalent to:

$$\sum_{i=1}^{m} \langle \mathbf{w}, \mathbf{x}_i \rangle \mathbf{x}_i = \sum_{i=1}^{m} y_i \mathbf{x}_i$$

Least squares derivation (second step)

The minimum coincides with the zero-gradient point

$$\frac{\partial L_S}{\partial \mathbf{w}} = \frac{2}{m}\sum_{i=1}^{m}(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)\mathbf{x}_i$$

We then need to solve the following equation for w:

$$\frac{2}{m}\sum_{i=1}^{m}(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i)\mathbf{x}_i = 0$$

This is equivalent to:

$$\sum_{i=1}^{m}\langle \mathbf{w}, \mathbf{x}_i \rangle \mathbf{x}_i = \sum_{i=1}^{m} y_i \mathbf{x}_i$$

Least squares derivation (final step)

We can define two utility matrices:

Samples over columns

$$\mathbf{A} = \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix}^T \qquad \mathbf{b} = \sum_{i=1}^{m} y_i \mathbf{x}_i = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

We can then take our equation:

$$\sum_{i=1}^{m} \langle \mathbf{w}, \mathbf{x}_i \rangle \mathbf{x}_i = \sum_{i=1}^{m} y_i \mathbf{x}_i$$

By applying linear operations, it becomes a linear system:

$$\mathbf{A}\mathbf{w} = \mathbf{b} \qquad \mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$$

The least squares algorithm

$$\mathbf{A} = \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix}$$

$$\mathbf{b} = \sum_{i=1}^{m} y_i \mathbf{x}_i = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

There are several solvers for linear systems (e.g., numpy.linalg.solve in Python)

$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$

If A is not invertible, we need to handle things a bit differently (see Appendix C in the book)

Ridge regression

Ridge regression is linear regression with Tikhonov regularization

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} \lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i \right)^2$$

The parameter λ tunes the trade-off between stability and empirical risk

Ridge regression

$$\mathbf{A} = \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix}$$

$$\mathbf{b} = \sum_{i=1}^{m} y_i \mathbf{x}_i = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{x}_1 & \dots & \mathbf{x}_m \\ \vdots & & \vdots \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

Setting the gradient to 0, we get

$$2\lambda \mathbf{w} + \frac{1}{m} \sum_{i=1}^{m} (\langle \mathbf{w}, \mathbf{x}_i \rangle - y_i) \mathbf{x}_i = 0$$

This is equivalent to

$$2\lambda m \mathbf{w} + \sum_{i=1}^{m} \langle \mathbf{w}, \mathbf{x}_i \rangle \mathbf{x}_i = \sum_{i=1}^{m} y_i \mathbf{x}_i$$

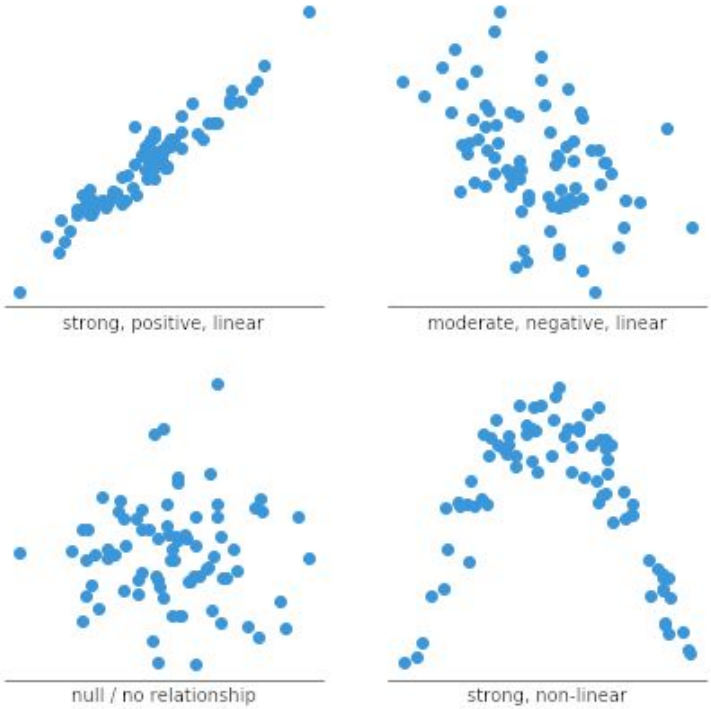
Using A and b, we have (the matrix is always invertible)

$$(2\lambda m \mathbf{I} + \mathbf{A})\, \mathbf{w} = \mathbf{b}$$

Are all problems linear?

Linear regression has a low true risk on the upper left problem. On the upper right, there is more noise (the relation is weaker), and on the lower left, the inputs and outputs are uncorrelated.

The problem on the lower right is non-linear: the linear hypothesis class is not powerful enough!



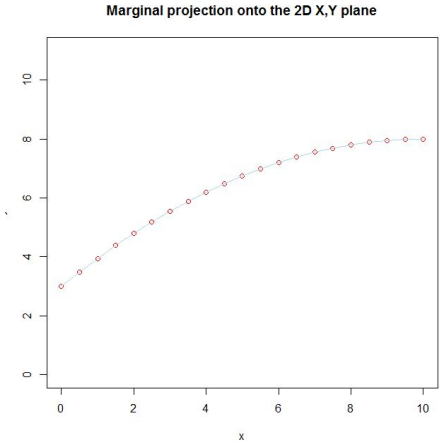
Generalized linear regression

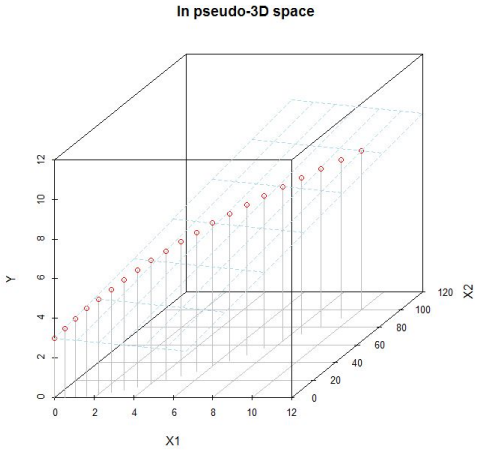
Solving a polynomial problem directly is complicated even in one dimension

We can add dimensions to make a non-linear problem linear in the wider space through the transformation

$$\psi(x) = (1, x, x^2, \ldots, x^n)$$

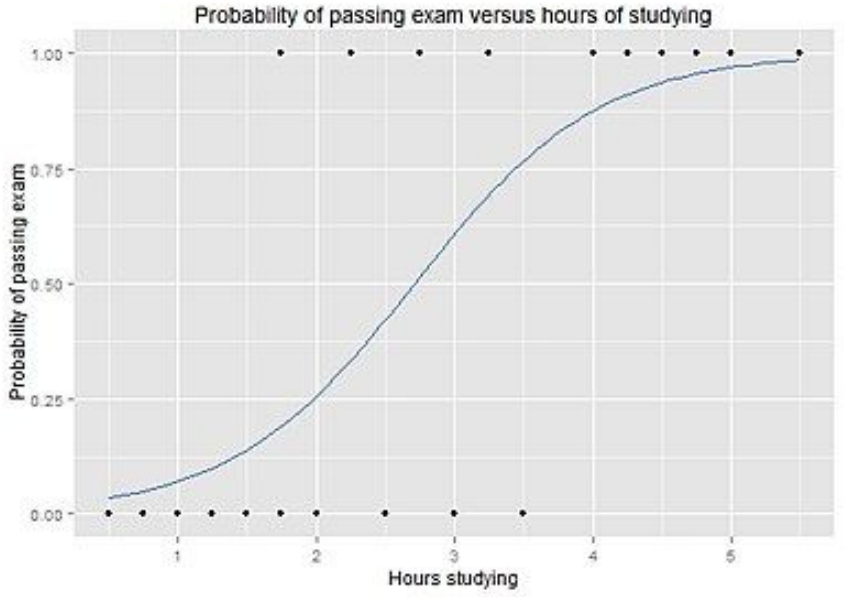
Note that this is not limited to polynomials, and can be applied to any function of the input, but the input variables are not statistically independent





Logistic regression

Classification poses a hard threshold, but in some cases, the probability of being in one class does not go sharply from 0 to 1

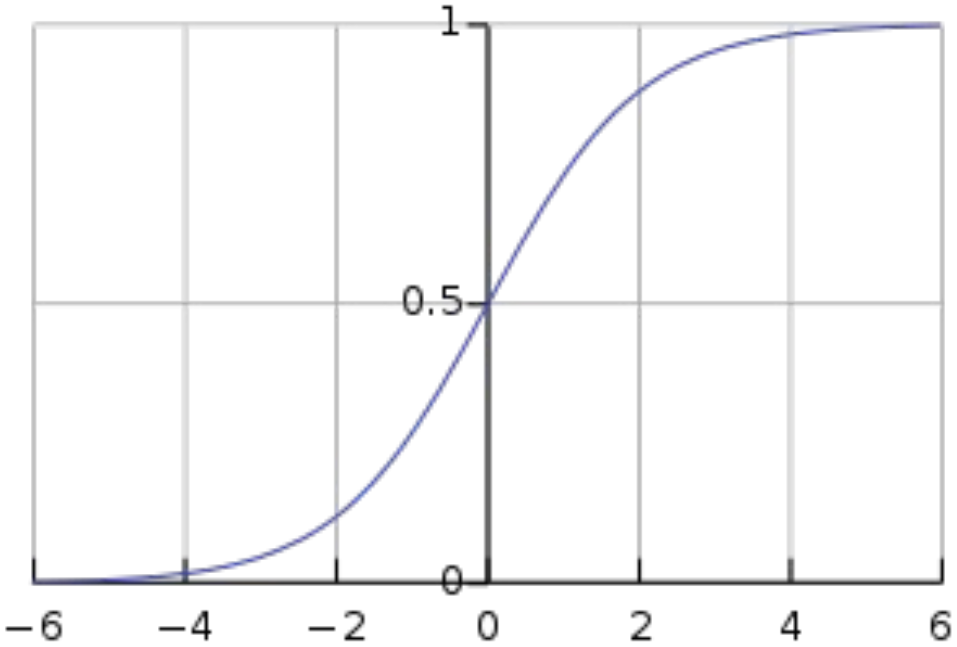


In these cases, we can pose a classification problem as a regression problem, using the probability as a label

Logistic regression

We assume that the probability follows a sigmoid function

$$\phi_{\text{sig}}(x) = \frac{1}{1 + e^{-x}}$$



Probability is higher than 0.5 for positive values - this is like a smooth sign function with a soft boundary

Logistic regression

The hypothesis class for logistic regression combines the sigmoid with a linear model

$$H_{\text{sig}} = \phi_{\text{sig}} \circ L_d = \{\mathbf{x} \to \phi_{\text{sig}}(\langle \mathbf{w}, \mathbf{x} \rangle),\ \mathbf{w} \in \mathbb{R}^d\} \qquad h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}$$

We consider a logarithmic loss function

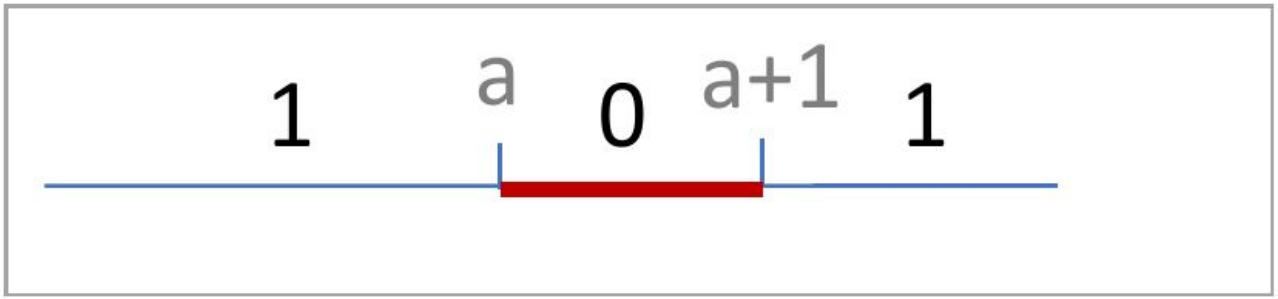
$$\ell(h_{\mathbf{w}}, (\mathbf{x}, y)) = \log\left(1 + e^{-y\langle \mathbf{w}, \mathbf{x} \rangle}\right)$$

If $y\langle \mathbf{w}, \mathbf{x} \rangle > 0$, the exponential is between 0 and 1, and the loss is low. On the other hand, the loss grows approximately linearly as the exponent increases

Part 2: Exercises

Università degli Studi di Padova

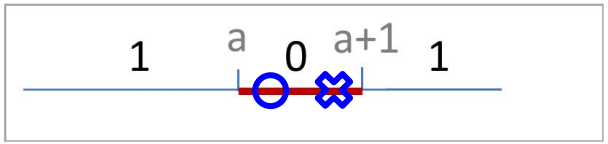
Exercise 1.1



$$h_a(x) = \begin{cases} 1 & if\ x \leq a\ OR\ x \geq a+1 \\ 0 & otherwise \end{cases}$$

Compute the VC dimension of this

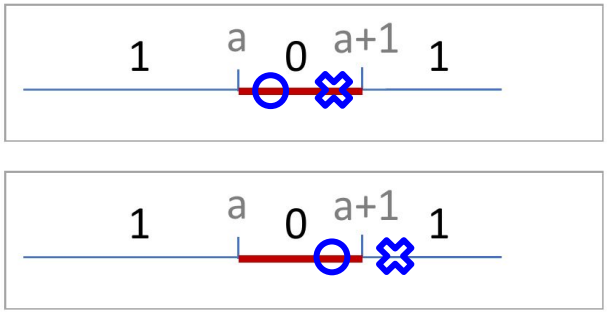
VC≥2



We can find two points (any points that are closer to each other than 1) that are shattered by our class. Let's pick 0 and 0.5

➔ a=-0.25: output 00

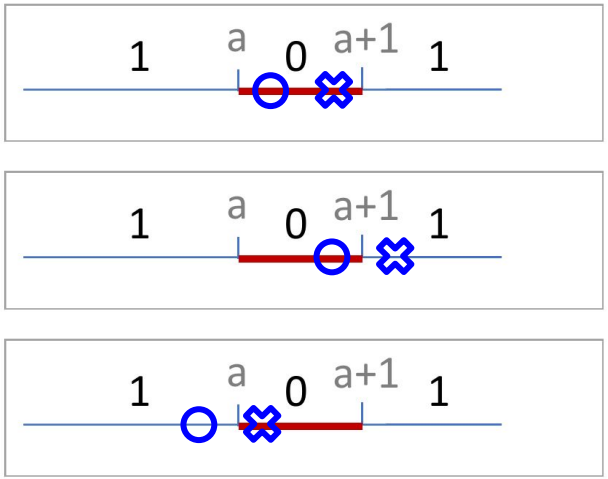
VC≥2



We can find two points (any points that are closer to each other than 1) that are shattered by our class. Let's pick 0 and 0.5

➔ a=-0.25: output 00
➔ a=-0.75: output 01

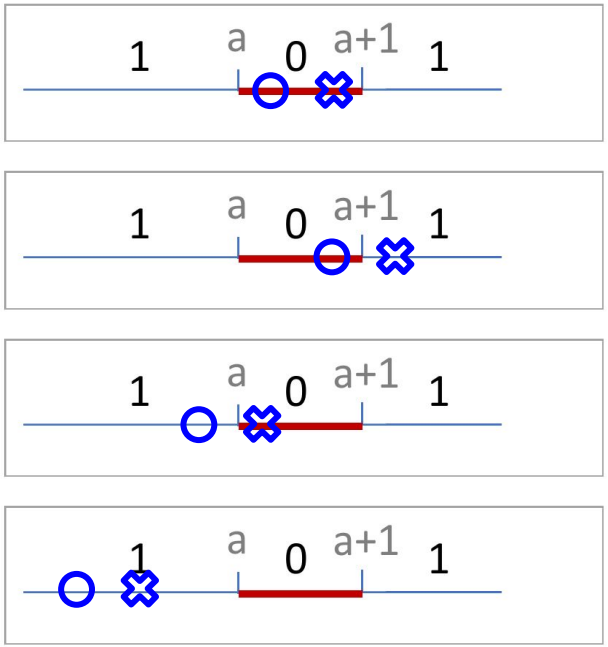
VC≥2



We can find two points (any points that are closer to each other than 1) that are shattered by our class. Let's pick 0 and 0.5

- ➔ a=-0.25: output 00
- ➔ a=-0.75: output 01
- ➔ a=0.25: output 10

VC≥2



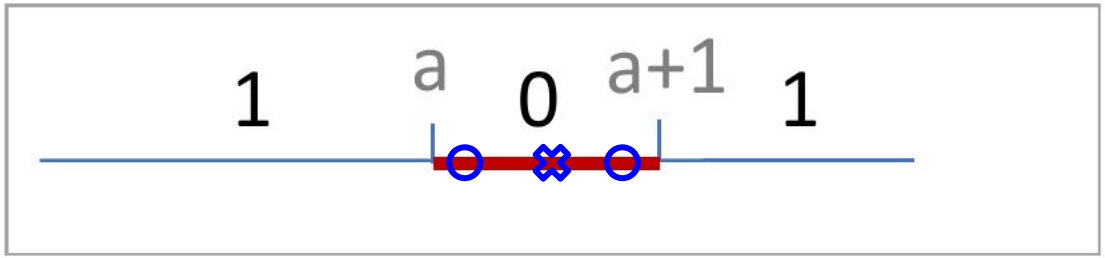
We can find two points (any points that are closer to each other than 1) that are shattered by our class. Let's pick 0 and 0.5

- ➔ a=-0.25: output 00
- ➔ a=-0.75: output 01
- ➔ a=0.25: output 10
- ➔ a=0.75: output 11

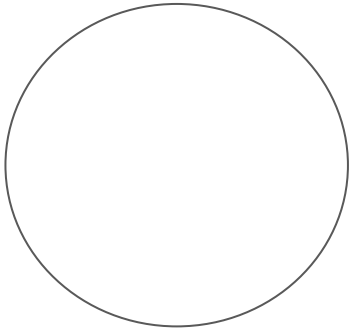
As we only need to find one set that is shattered, we know that the VC dimension is at least 2

VC=2

Second step: we need to prove that the VC dimension is exactly 2. This is easy, as, when we order any 3 points from the smallest to the largest value, output 010 is impossible

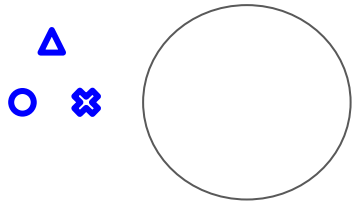


Exercise 1.2



The function we selected is a circle with radius 1. Points inside the circle are in class 1, points outside it are in class -1

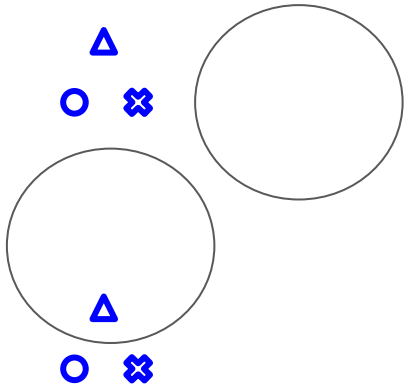
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

➔ O=(2,0): output 000

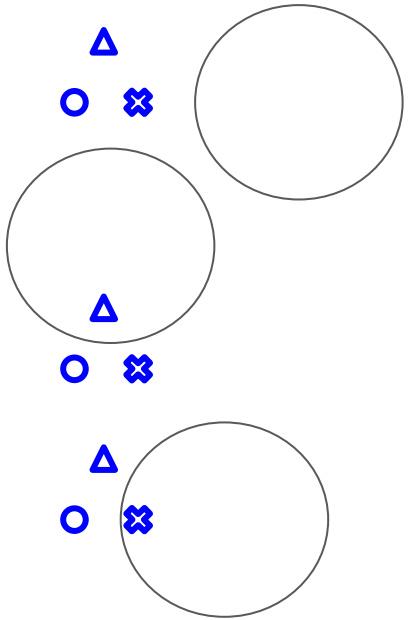
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

➔ O=(2,0): output 000
➔ O=(0,1+√2/8): output 001

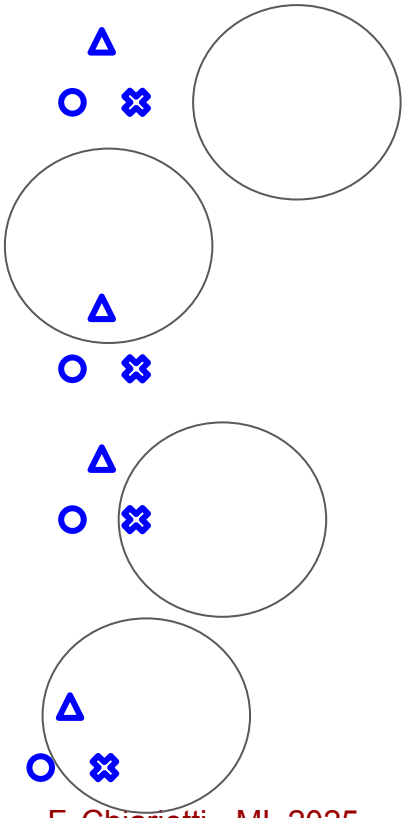
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010

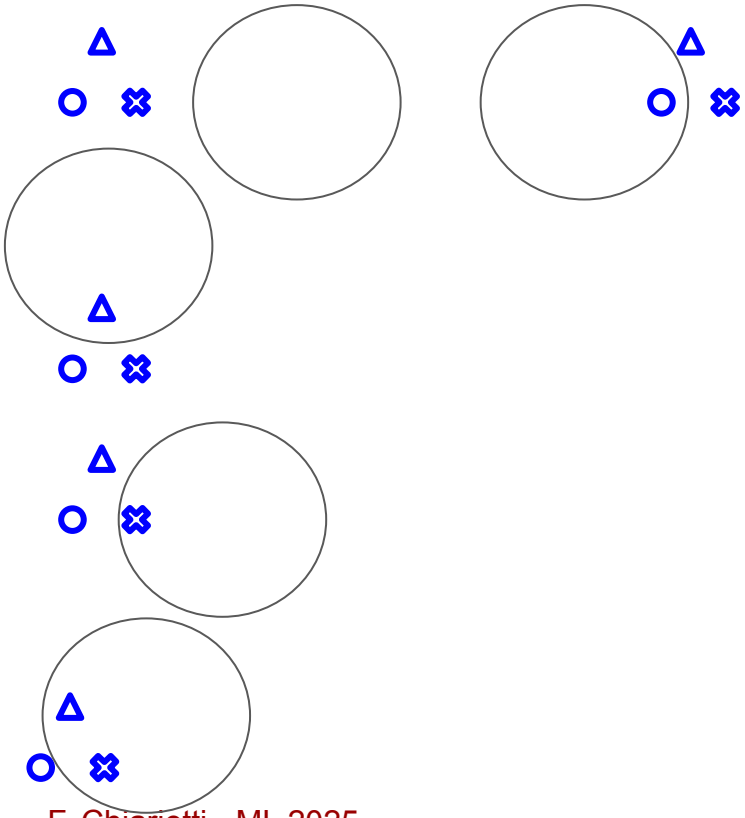
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011

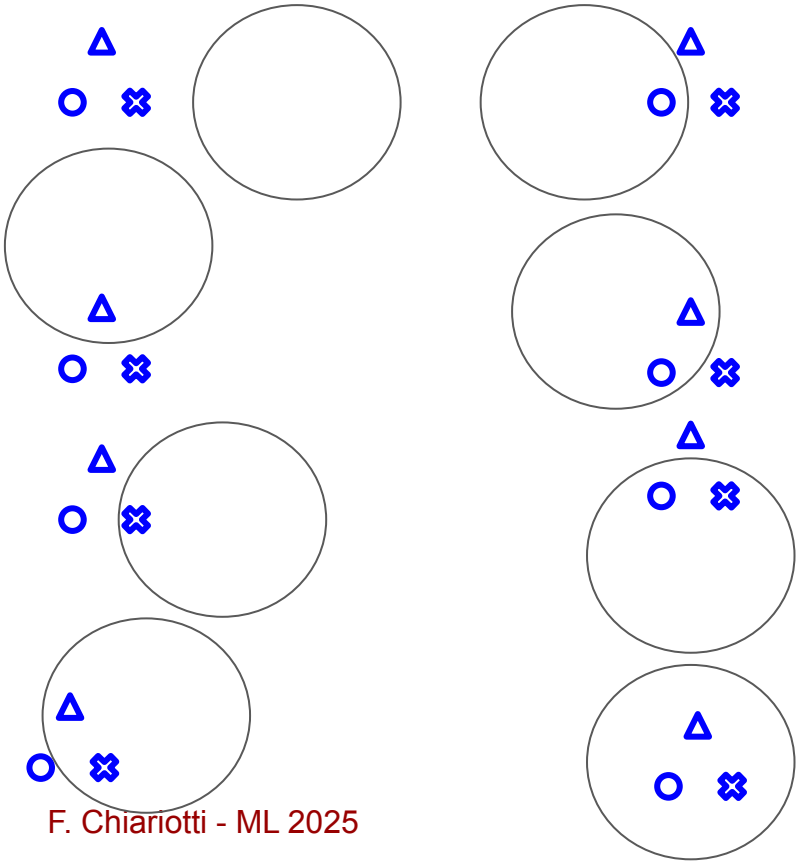
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011
- ➔ O=(-1.1,0): output 100

VC≥3

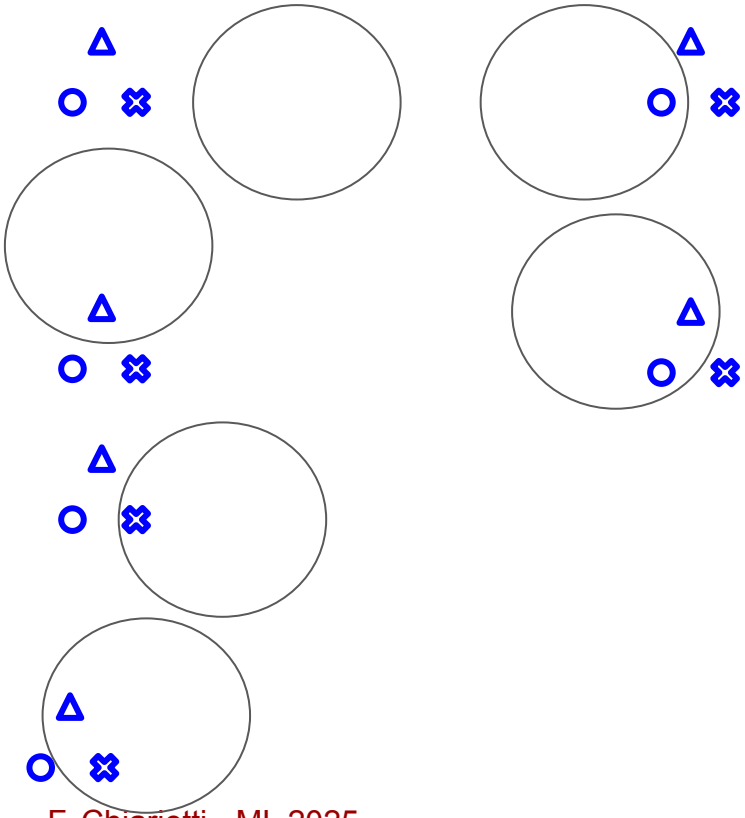


Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011
- ➔ O=(-1.1,0): output 100
- ➔ O=(-0.9,√2/4): output 101
- ➔ O=(0,-2/3): output 110
- ➔ O=(0,√2/8): output 111

As we found a set that works, the VC dimension is at least 3

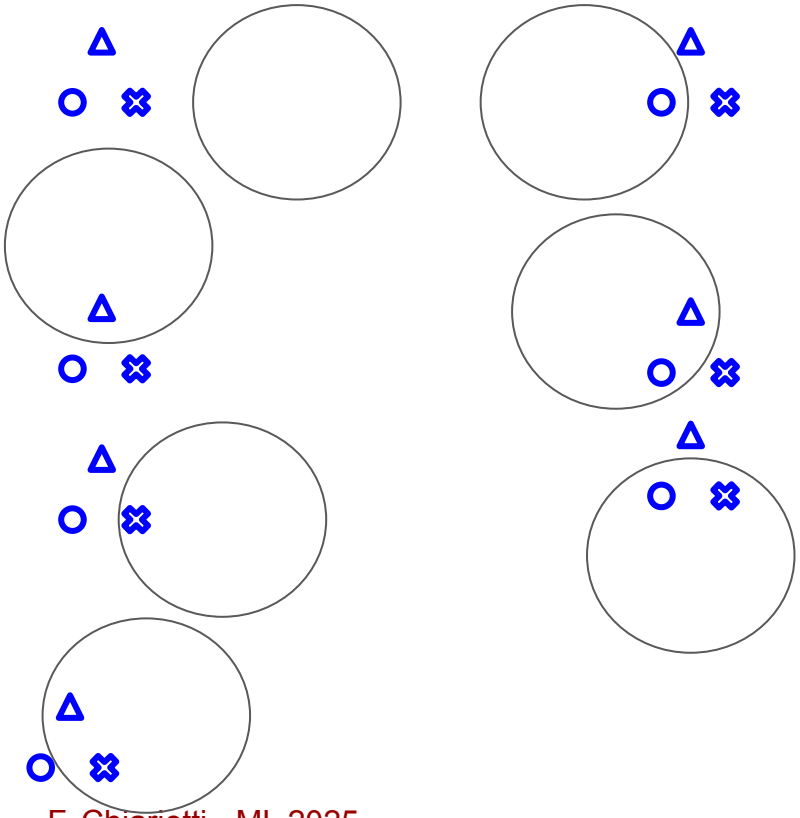
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011
- ➔ O=(-1.1,0): output 100
- ➔ O=(-0.9,√2/4): output 101

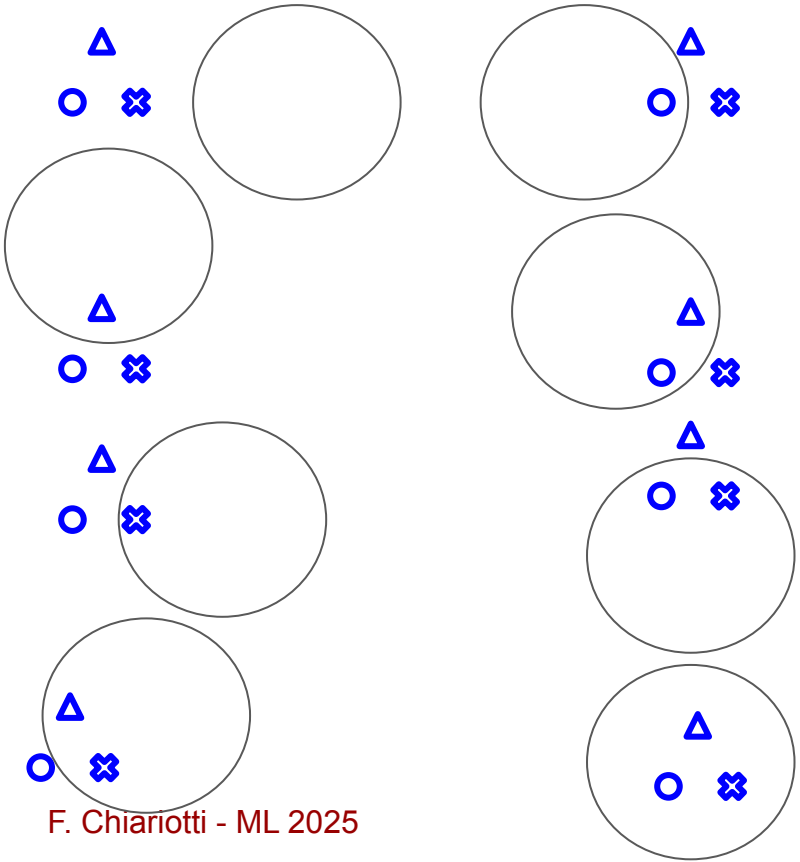
VC≥3



Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011
- ➔ O=(-1.1,0): output 100
- ➔ O=(-0.9,√2/4): output 101
- ➔ O=(0,-2/3): output 110

VC≥3

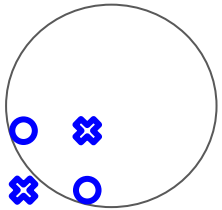


Let us define an equilateral triangle, with points A=(-0.25,0), B=(0.25,0), C=(0,√2/4)

- ➔ O=(2,0): output 000
- ➔ O=(0,1+√2/8): output 001
- ➔ O=(1.1,0): output 010
- ➔ O=(0.9,√2/4): output 011
- ➔ O=(-1.1,0): output 100
- ➔ O=(-0.9,√2/4): output 101
- ➔ O=(0,-2/3): output 110
- ➔ O=(0,√2/8): output 111

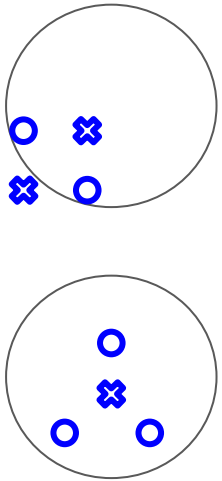
As we found a set that works, the VC dimension is at least 3

VC=3



No point is in the triangle formed by the other 3: 1010 is impossible (if one diagonal is inside, at least one point on the other diagonal is also inside)

VC=3

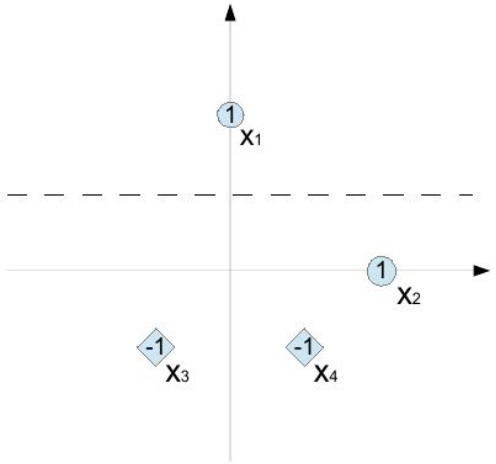


No point is in the triangle formed by the other 3: 1010 is impossible (if one diagonal is inside, at least one point on the other diagonal is also inside)

One point is in the triangle formed by the other 3: 1110 is impossible (if all three external points are inside, the fourth must also be inside the circle)

Since it is impossible to shatter a set of size 4, the VC dimension is exactly equal to 3.

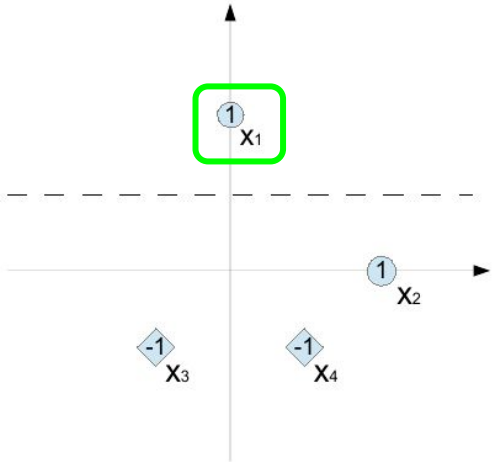
Exercise 2.1



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

The initial weight vector is (-1,0,1) and the learning rate is 1

Checking points

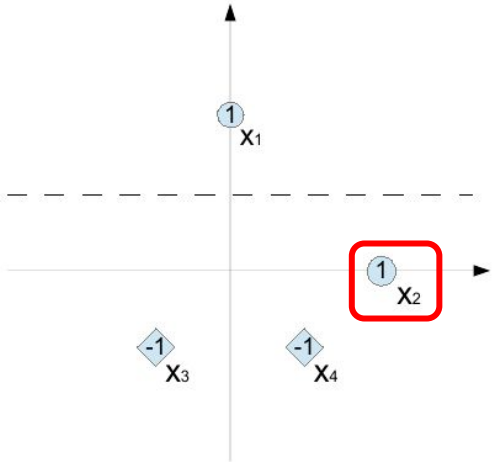


x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

-1x1: bias
0x0: first coordinate
1x2: second coordinate
total:-1+0+2=1

Checking points

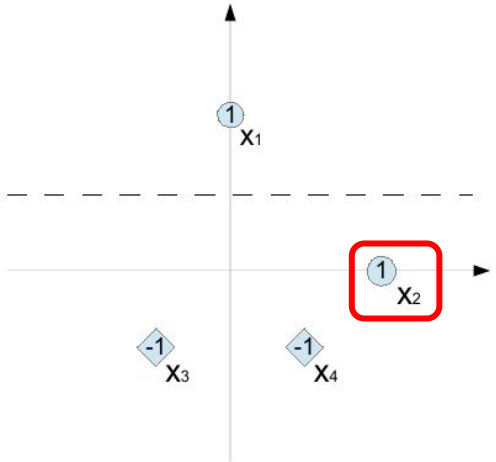


x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

-1x1: bias
0x2: first coordinate
1x0: second coordinate
total:-1+0+0=-1

Perceptron update



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(1)}=w^{(0)}+y_i x_i$

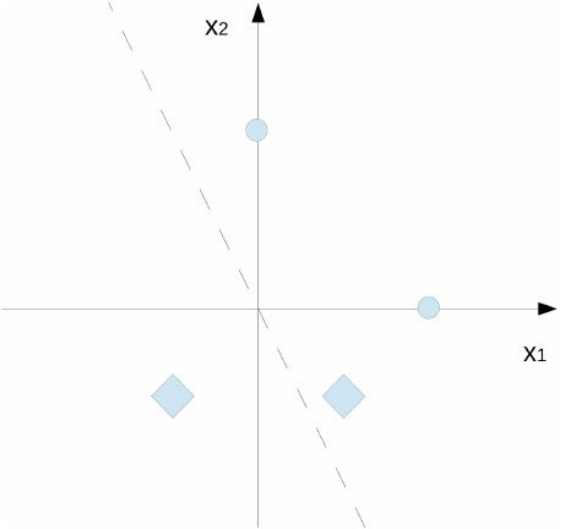
-1+1: bias

0+2: first coord.

1+0: second coord.

$w^{(1)}=(0,2,1)$

Drawing the new boundary



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

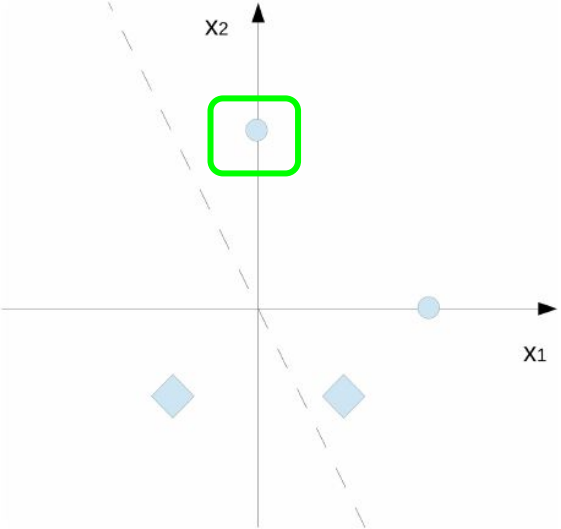
$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w_0+w_1x_1+w_2x_2=0$

$x_2=-(w_1/w_2)x_1-(w_0/w_2)$ -> $x_2=-2x_1$

Checking points



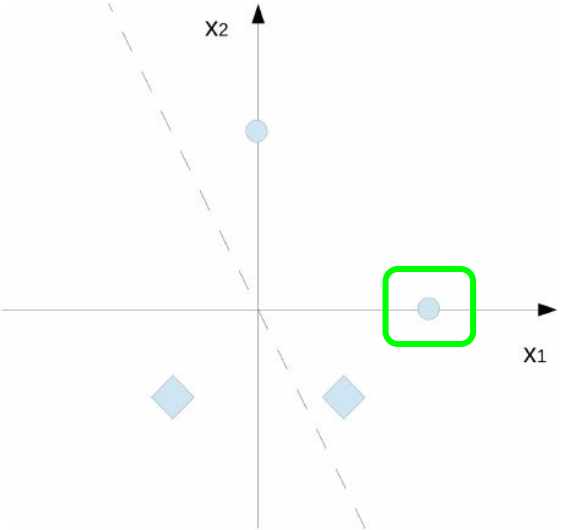
x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

0x1: bias
2x0: first coordinate
1x2: second coordinate
total:0+0+2=2

Checking points



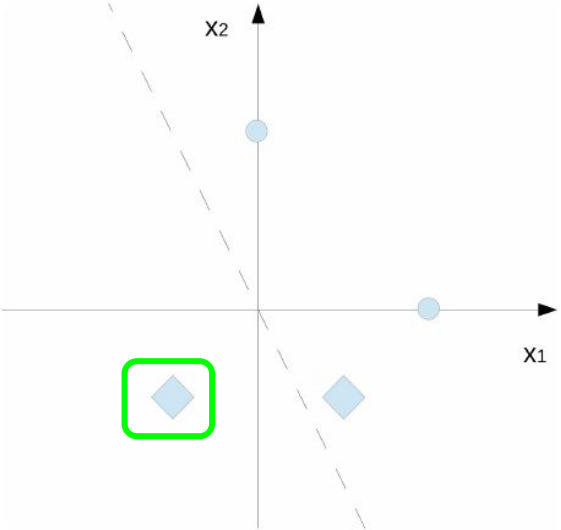
x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

0x1: bias
2x2: first coordinate
1x0: second coordinate
total:0+4+0=4

Checking points



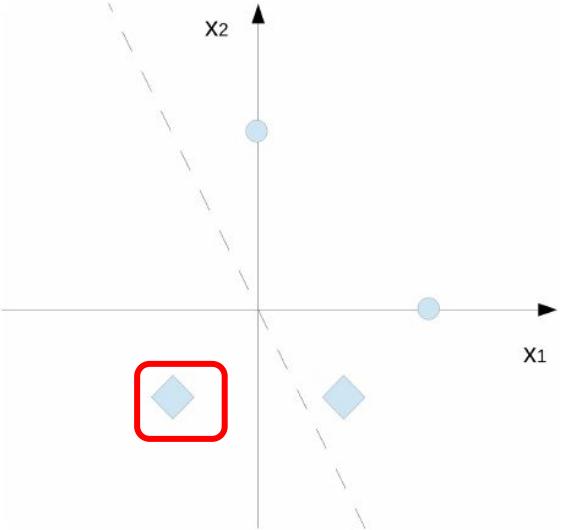
x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

0x1: bias
2x(-1): first coordinate
1x(-1): second coordinate
total:0-2-1=-3

Checking points



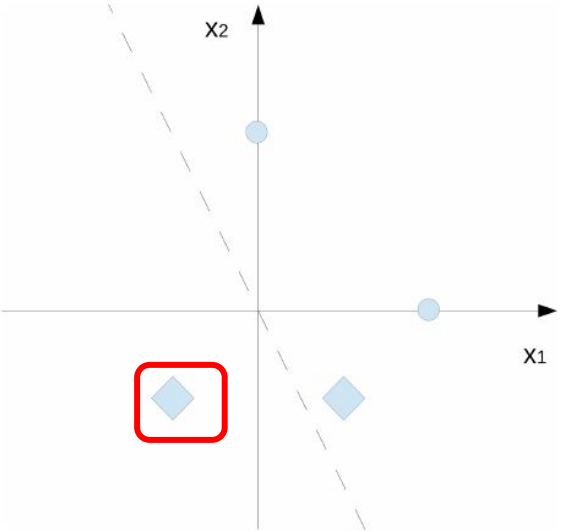
x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

0x1: bias
2x1: first coordinate
1x(-1): second coordinate
total:0+2-1=1

Perceptron update



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(2)}=w^{(1)}+y_i x_i$

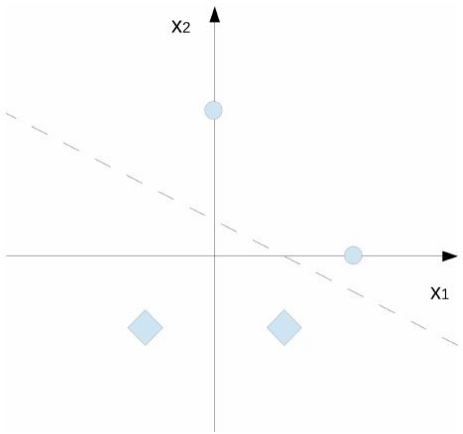
0-1: bias

2-1: first coord.

1-(-1): second coord.

$w^{(2)}=(-1,1,2)$

Drawing the new boundary



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

$w^{(0)}=(-1,0,1)$

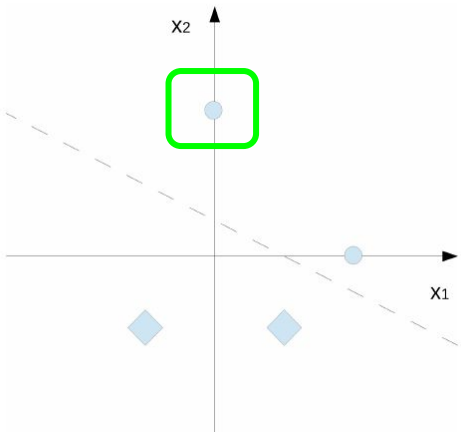
$w^{(1)}=(0,2,1)$

$w^{(2)}=(-1,1,2)$

$w_0+w_1x_1+w_2x_2=0$

$x_2=-(w_1/w_2)x_1-(w_0/w_2)$ -> $x_2=-1/2x_1+1/2$

Checking points



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

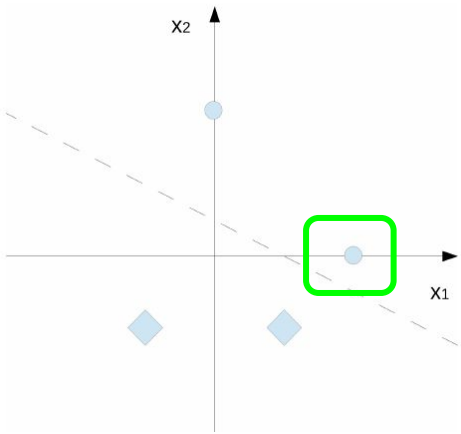
$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(2)}=(-1,1,2)$

0x(-1): bias
1x0: first coordinate
2x2: second coordinate
total:-1+0+4=3

Checking points



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

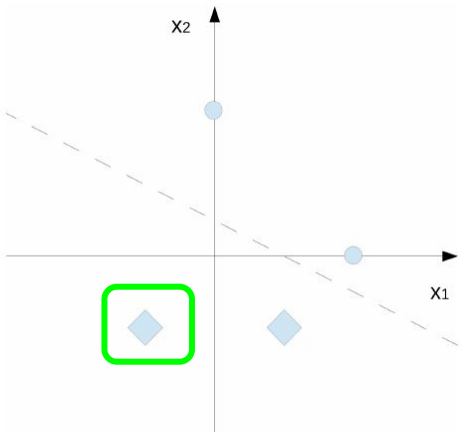
$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(2)}=(-1,1,2)$

0x(-1): bias
1x2: first coordinate
2x0: second coordinate
total:-1+2+0=1

Checking points



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

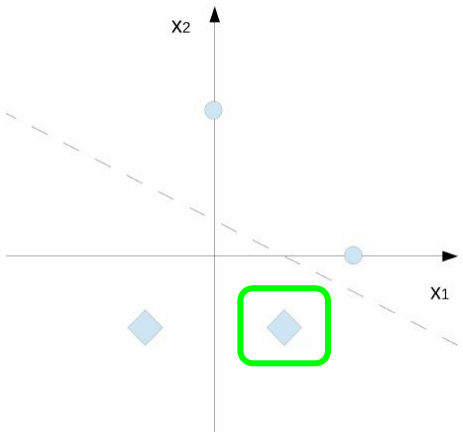
$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(2)}=(-1,1,2)$

0x(-1): bias
1x(-1): first coordinate
2x(-1): second coordinate
total:-1-1-2=-4

Checking points



x_1	x_2	y
0	2	1
2	0	1
-1	-1	-1
1	-1	-1

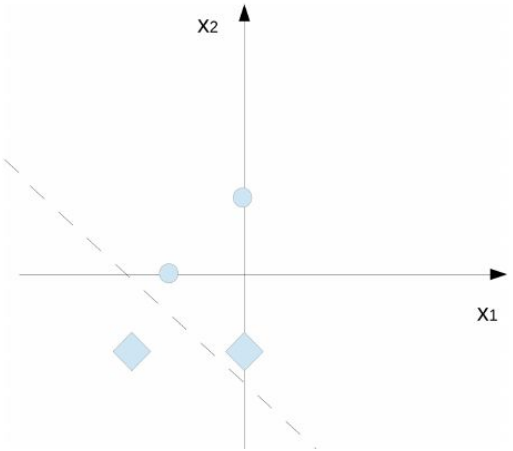
$w^{(0)}=(-1,0,1)$

$w^{(1)}=(0,2,1)$

$w^{(2)}=(-1,1,2)$

0x(-1): bias
1x1: first coordinate
2x(-1): second coordinate
total:-1+1-2=-2

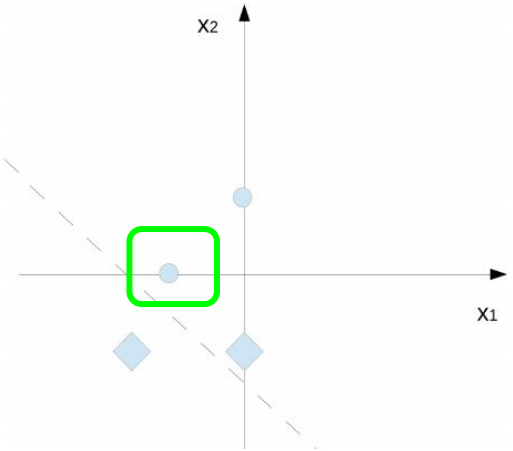
Exercise 2.2



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

The initial weight vector is (2,1.5,1.5) and the learning rate is 1

Checking the points

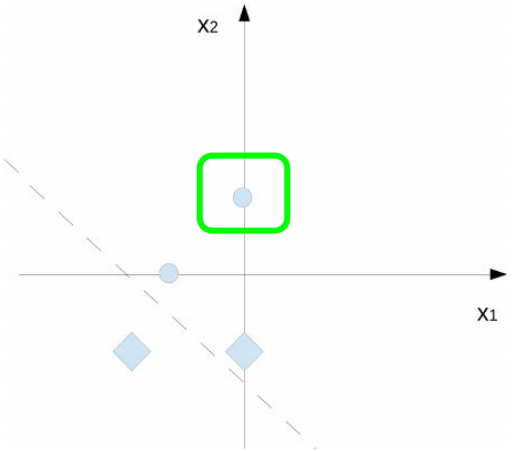


x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

2x1: bias
1.5x(-1): first coordinate
1.5x0: second coordinate
total:2-1.5=0.5

Checking the points

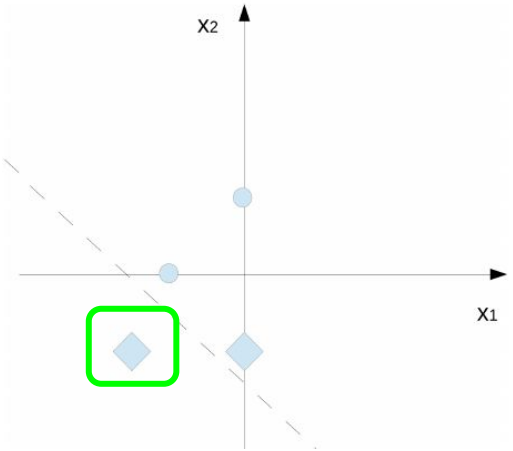


x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

2x1: bias
1.5x0: first coordinate
1.5x1: second coordinate
total:2+1.5=3.5

Checking the points

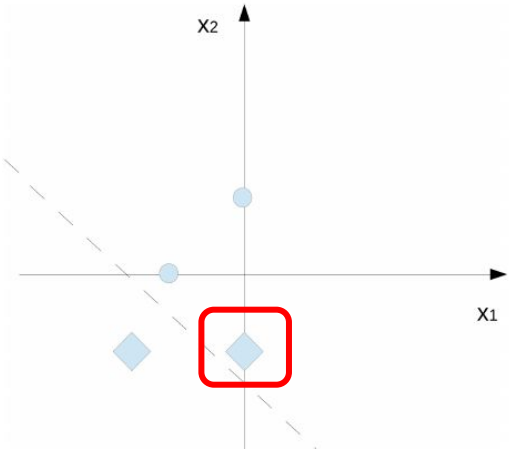


x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

2x1: bias
1.5x(-1.5): first coordinate
1.5x(-1): second coordinate
total:2-2.25-1.5=-1.75

Checking the points

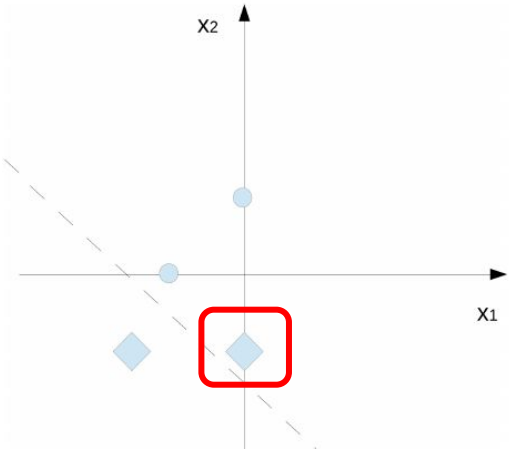


x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

2x1: bias
1.5x0: first coordinate
1.5x(-1): second coordinate
total:2-1.5=0.5

Perceptron update



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(1)}=w^{(0)}+y_i x_i$

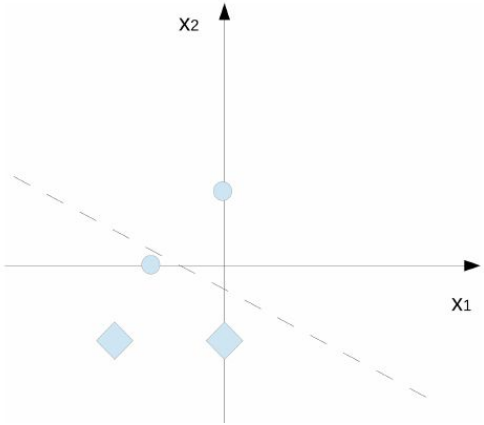
2-1: bias

1.5-0: first coord.

1.5-(-1): second coord.

$w^{(1)}=(1,1.5,2.5)$

Drawing the new boundary



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

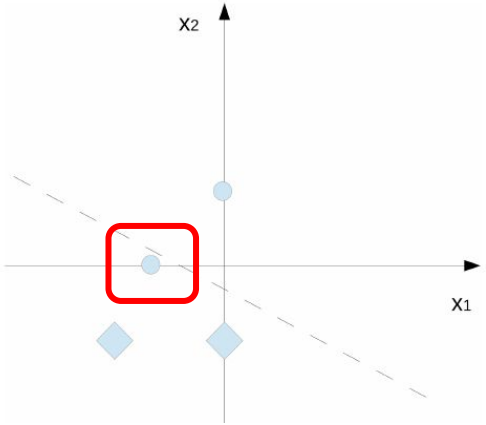
$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w_0+w_1x_1+w_2x_2=0$

$x_2=-(w_1/w_2)x_1-(w_0/w_2)$ -> $x_2=-3/5x_1-2/5$

Checking the points



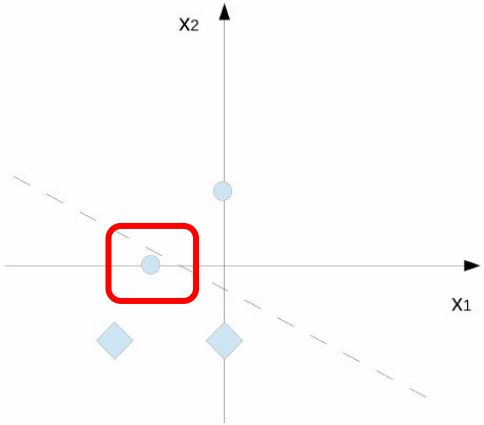
x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

1x1: bias
1.5x(-1): first coordinate
2.5x0: second coordinate
total:1-1.5=-0.5

Perceptron update



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

$w^{(2)}=w^{(1)}+y_i x_i$

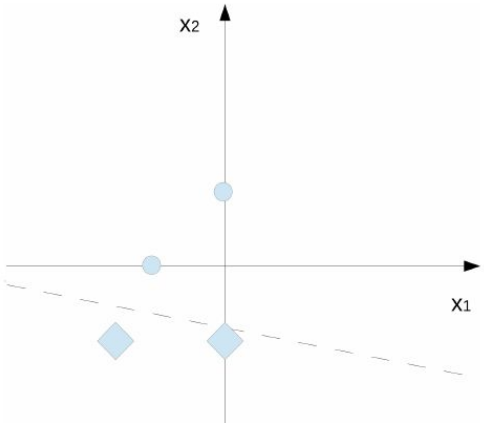
1+1: bias

1.5+(-1): first coord.

2.5+0: second coord.

$w^{(2)}=(2,0.5,2.5)$

Drawing the new boundary



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

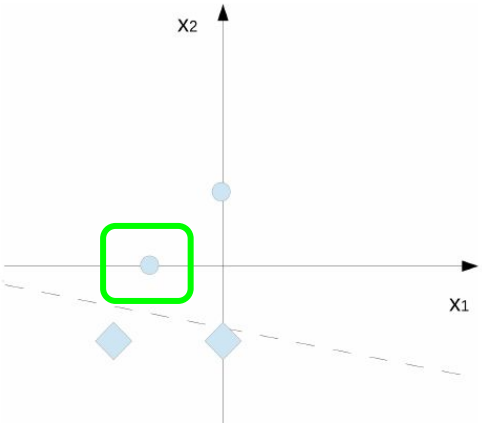
$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

$w_0+w_1x_1+w_2x_2=0$

$x_2=-(w_1/w_2)x_1-(w_0/w_2)$ -> $x_2=-1/5x_1-4/5$

Checking the points



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

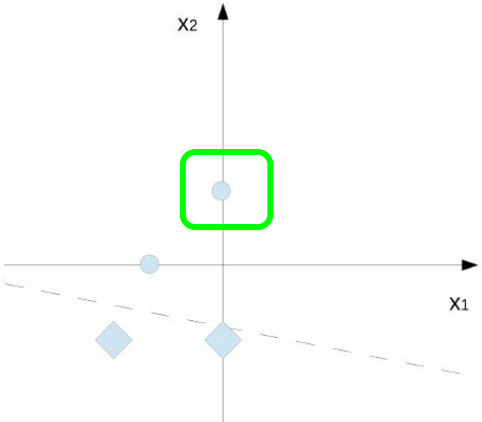
$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

2x1: bias
0.5x(-1): first coordinate
2.5x0: second coordinate
total:2-0.5+0=1.5

Checking the points



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

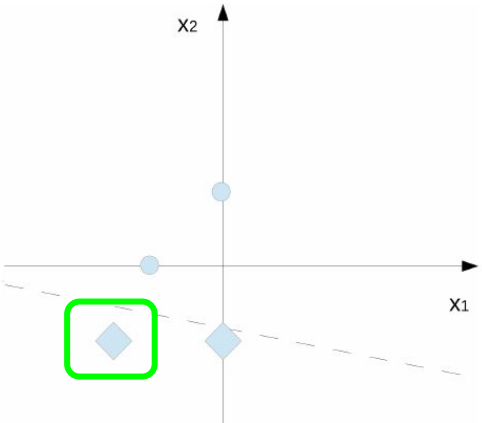
$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

2x1: bias
0.5x0: first coordinate
2.5x1: second coordinate
total:2+0+2.5=4.5

Checking the points



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

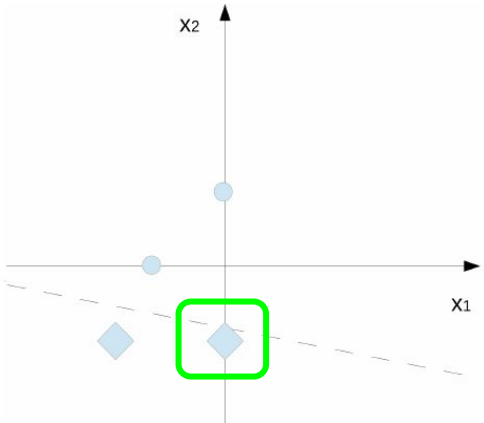
$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

2x1: bias
0.5x(-1.5): first coordinate
2.5x(-1): second coordinate
total:2-0.75-2.5=-1.25

Checking the points



x_1	x_2	y
-1	0	1
0	1	1
-1.5	-1	-1
0	-1	-1

$w^{(0)}=(2,1.5,1.5)$

$w^{(1)}=(1,1.5,2.5)$

$w^{(2)}=(2,0.5,2.5)$

2x1: bias
0.5x0: first coordinate
2.5x(-1): second coordinate
total:2+0-2.5=-0.5