INF623

2024/1

UFV

Inteligência Artificial

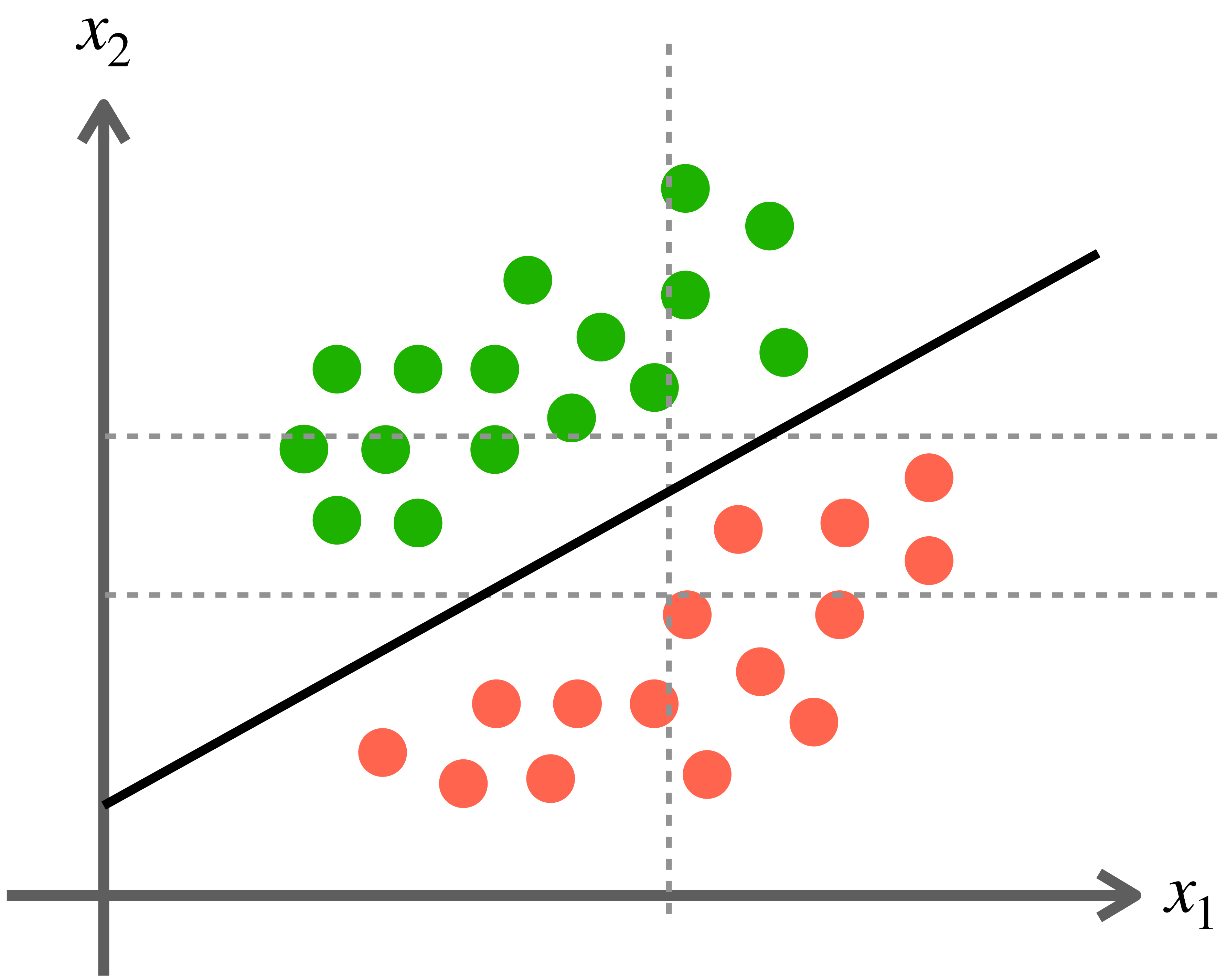
A27: Aprendizado supervisionado IV

Plano de aula

- Classificadores lineares
  - Perceptron
  - Support Vector Machines
  - Regressão logística

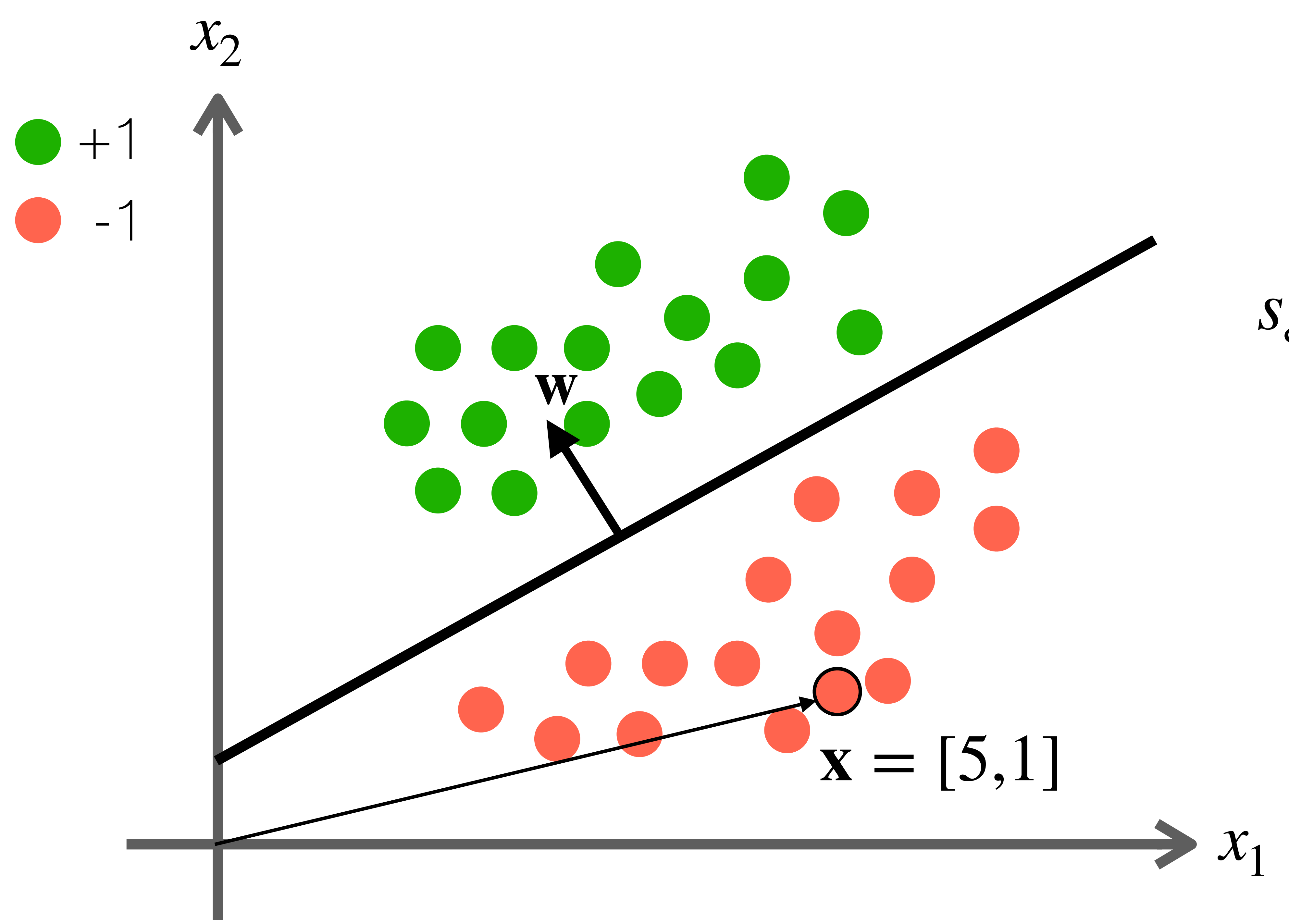
Problemas com as árvores de decisão

Árvores de decisão têm dificuldade em representar funções lineares simples, pois são baseadas em divisões ortogonais.



Perceptron

O Perceptron é um algoritmo de classificação binária onde o espaço de hipótese H é definido por uma combinação linear das características (reta em 2d, plano em 3d, hiperplano em mais dimensões).



$$h(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x} + b)$$

$$sgn(z) = \begin{cases} +1, & z > 0 \\ -1, & z < 0 \end{cases}$$

$$\mathbf{w} = [-2,1] \quad b = 0$$

$$h(\mathbf{x}) = sgn(-2 \cdot 5 + 1 \cdot 1 + 0)$$
$$h(\mathbf{x}) = sgn(-9)$$
$$h(\mathbf{x}) = -1$$

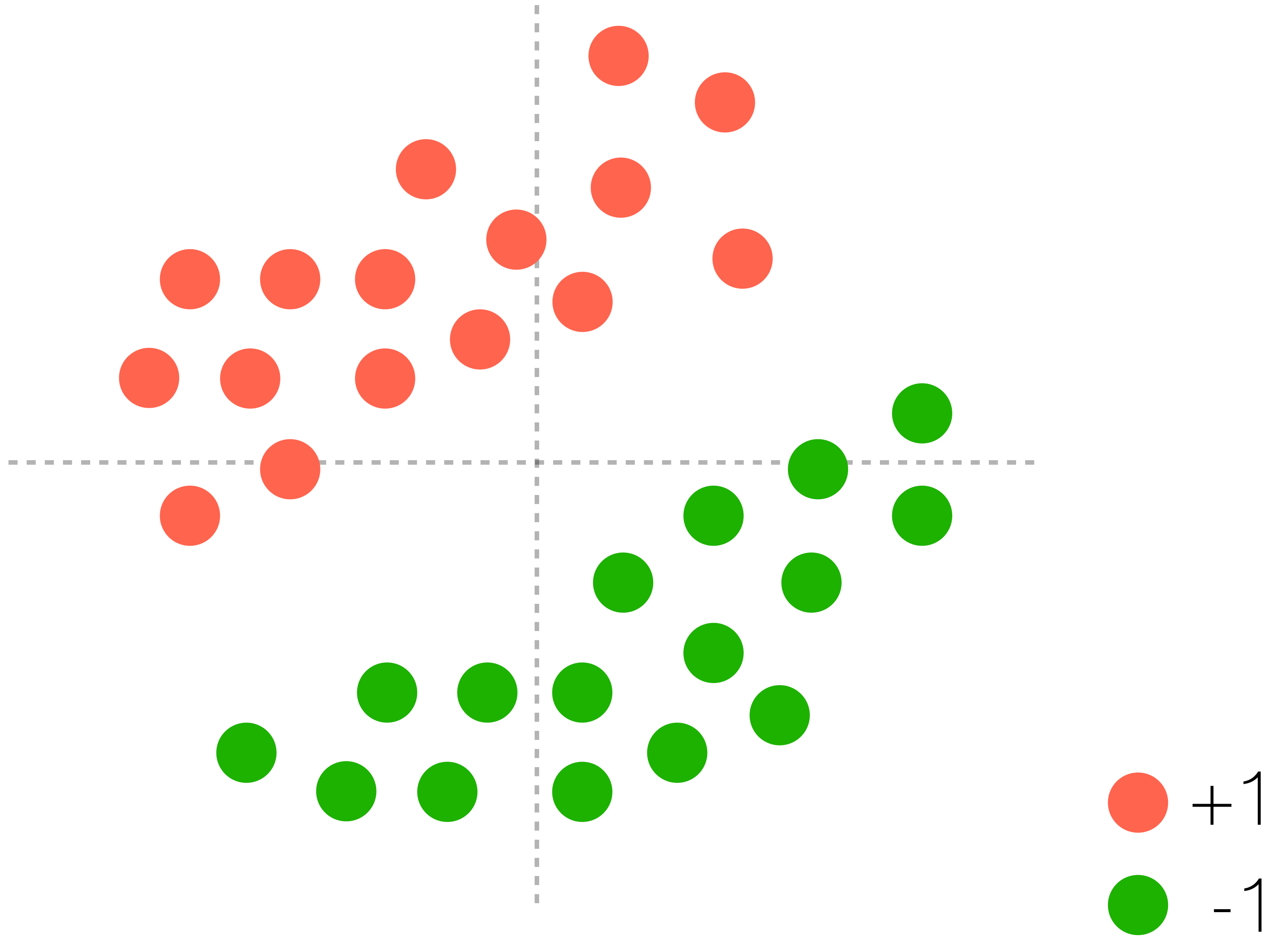
UFV

Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

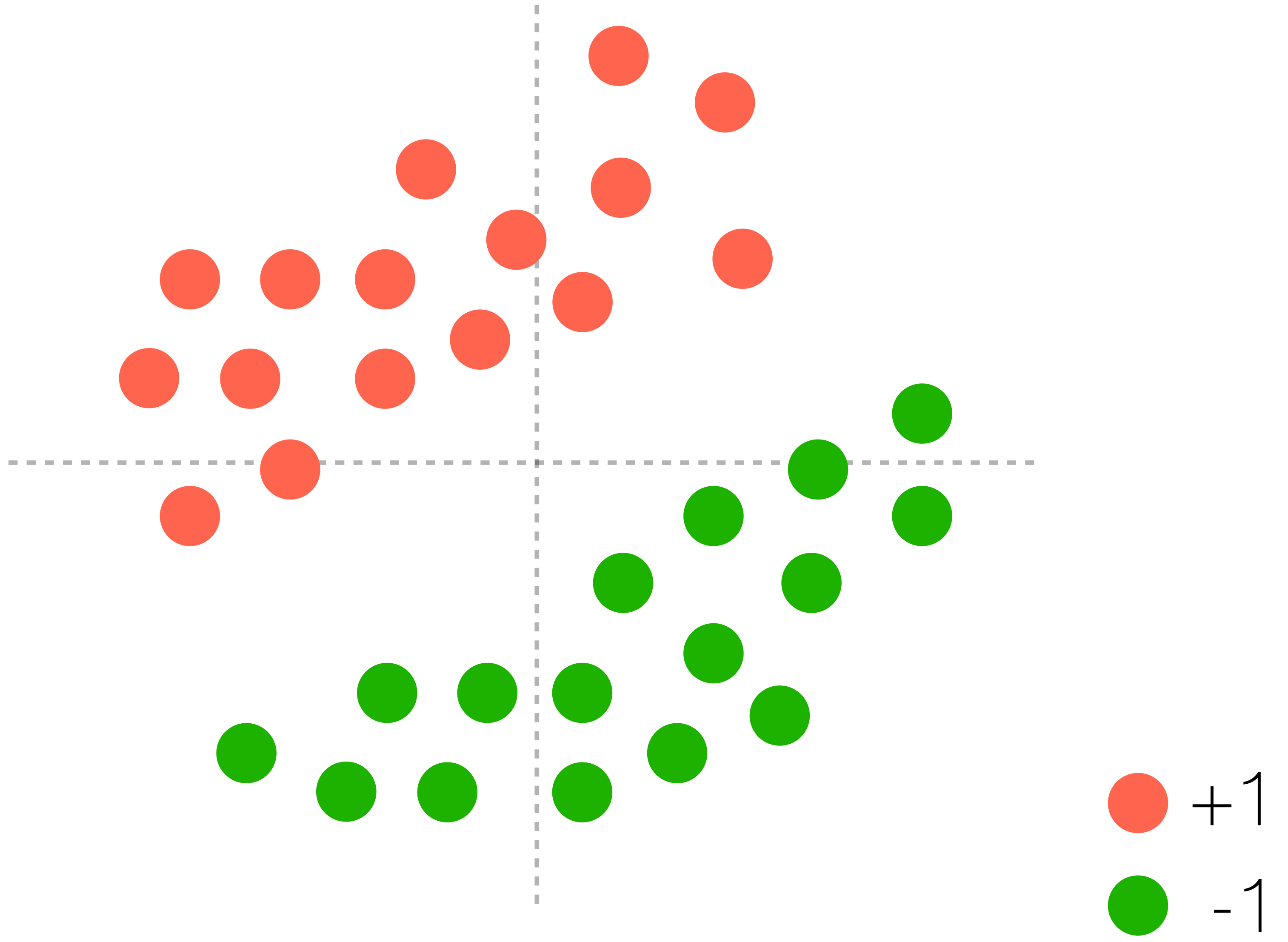


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$l = 32$

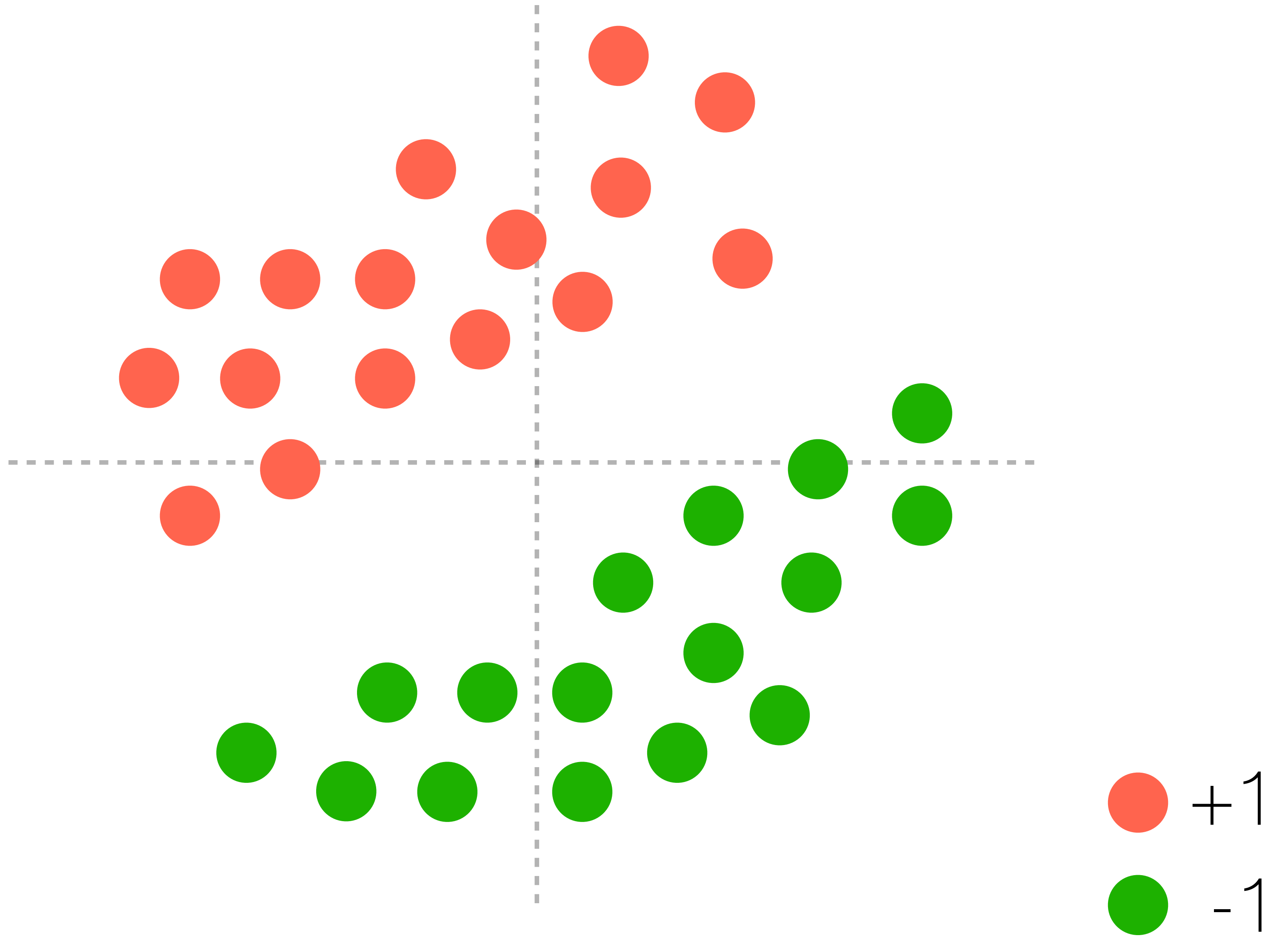


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$l = 32$
$\mathbf{w} = [0,0]$



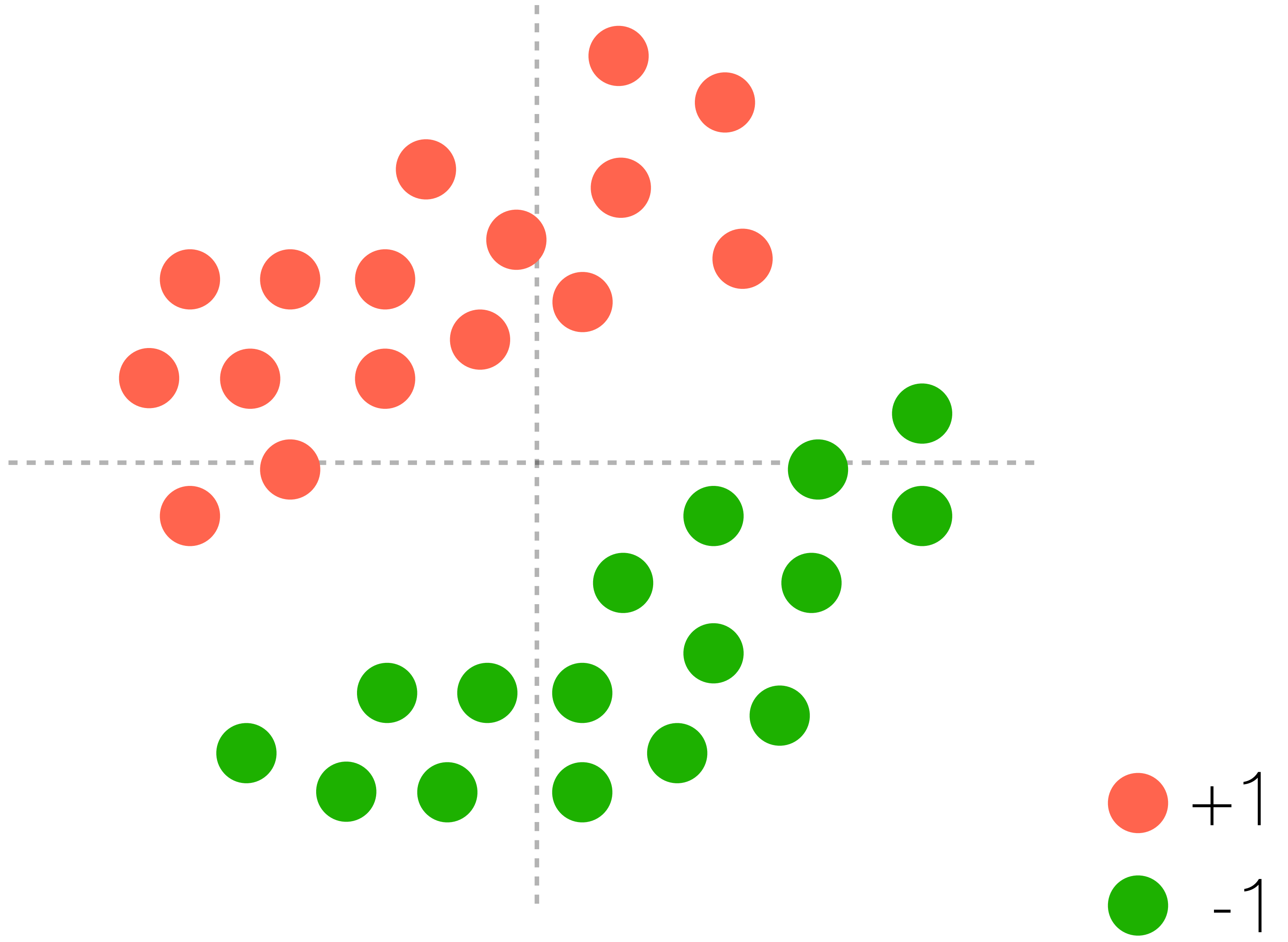
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$l = 32$

$\mathbf{w} = [0,0]$



UFV

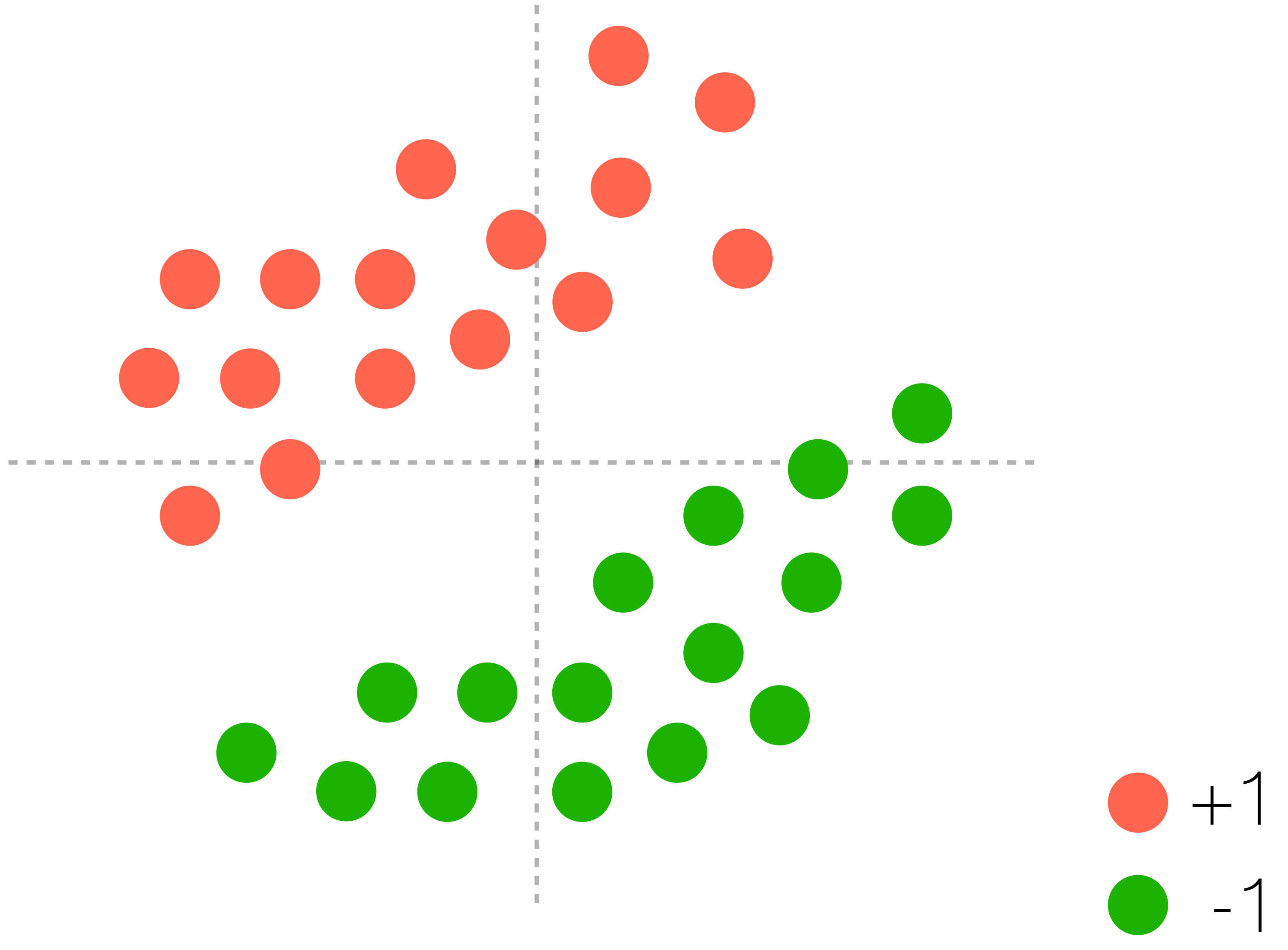
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [0{,}0]$

$l = 0$



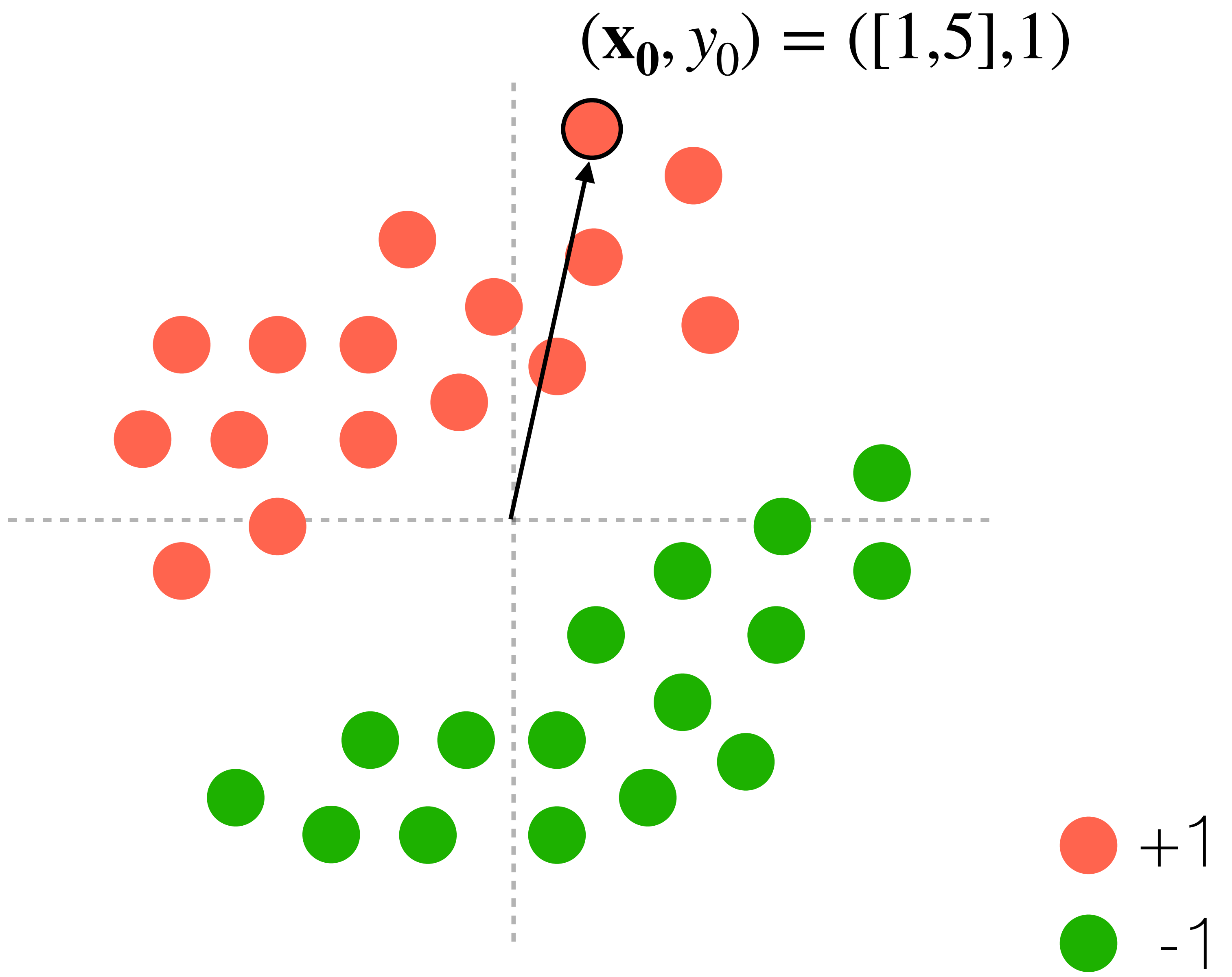
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
   l ← 0
   for (xi, yi) in D:
     if sign(w · xi) ≠ yi:
       w ← w + yi xi
       l ← l + 1
```

$\mathbf{w} = [0{,}0]$

$l = 0$



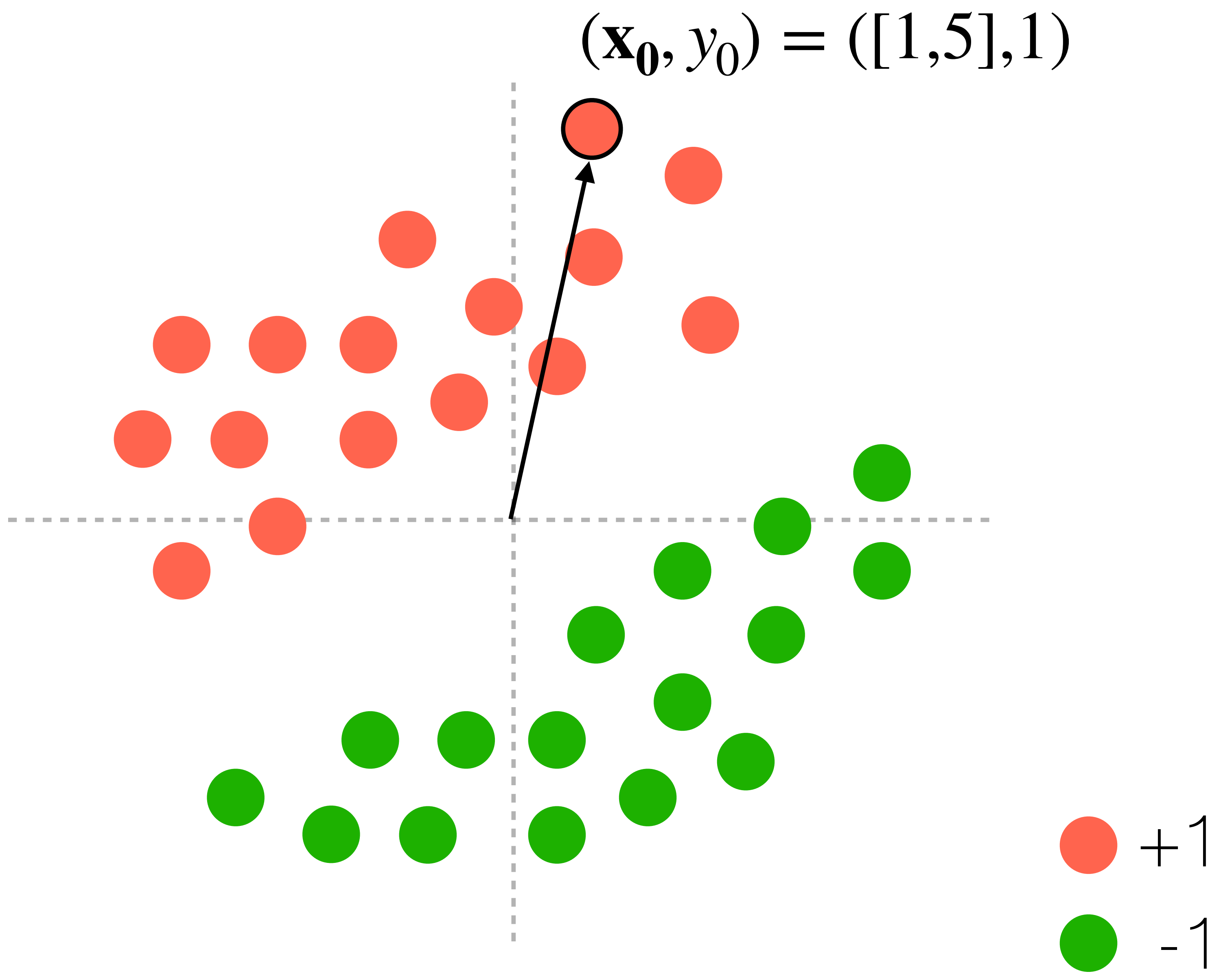
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [0{,}0]$

$l = 0$



Treinando o perceptron

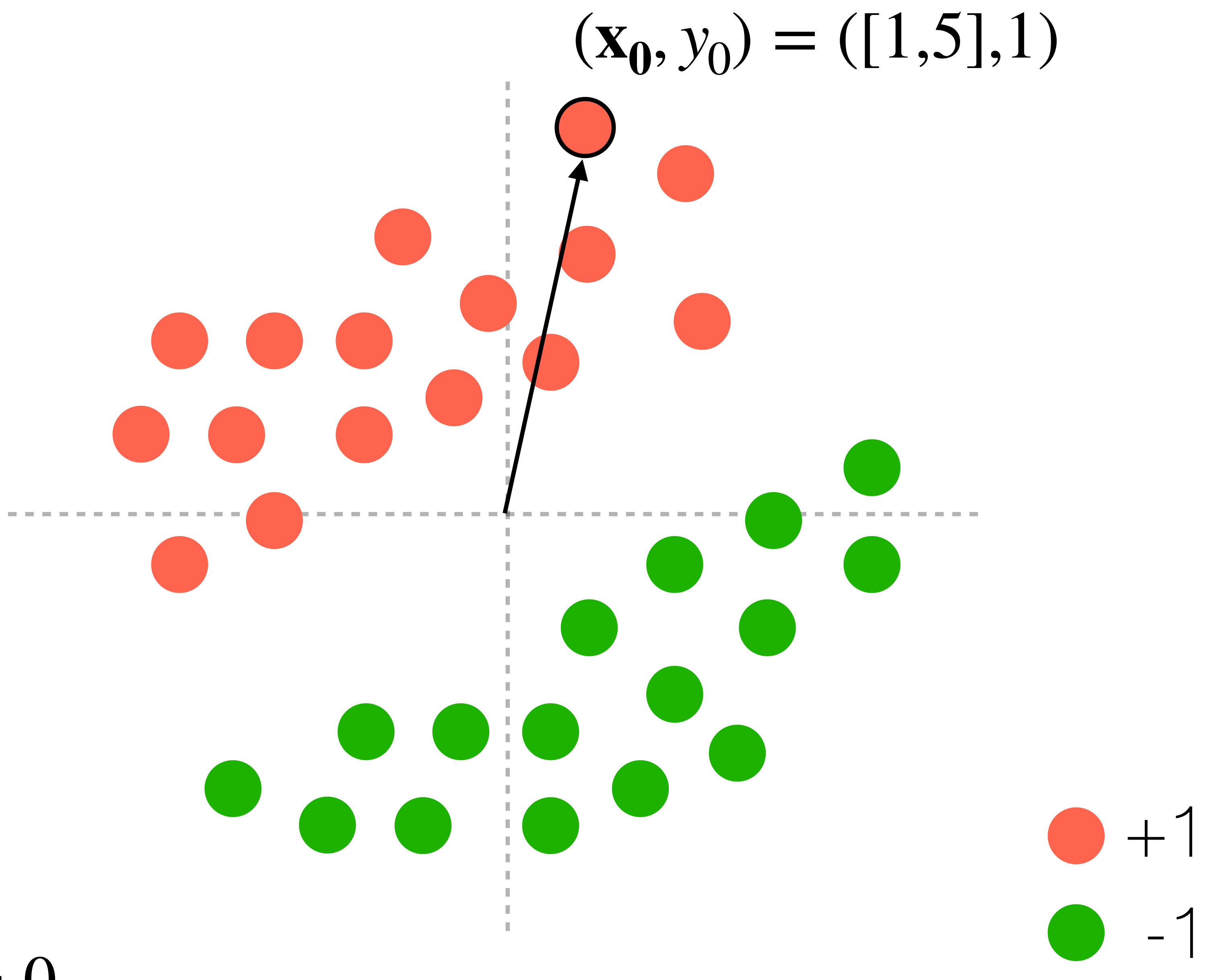
A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [0,0]$

$l = 0$

$sign(\mathbf{w} \cdot \mathbf{x_0}) = 0$



Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

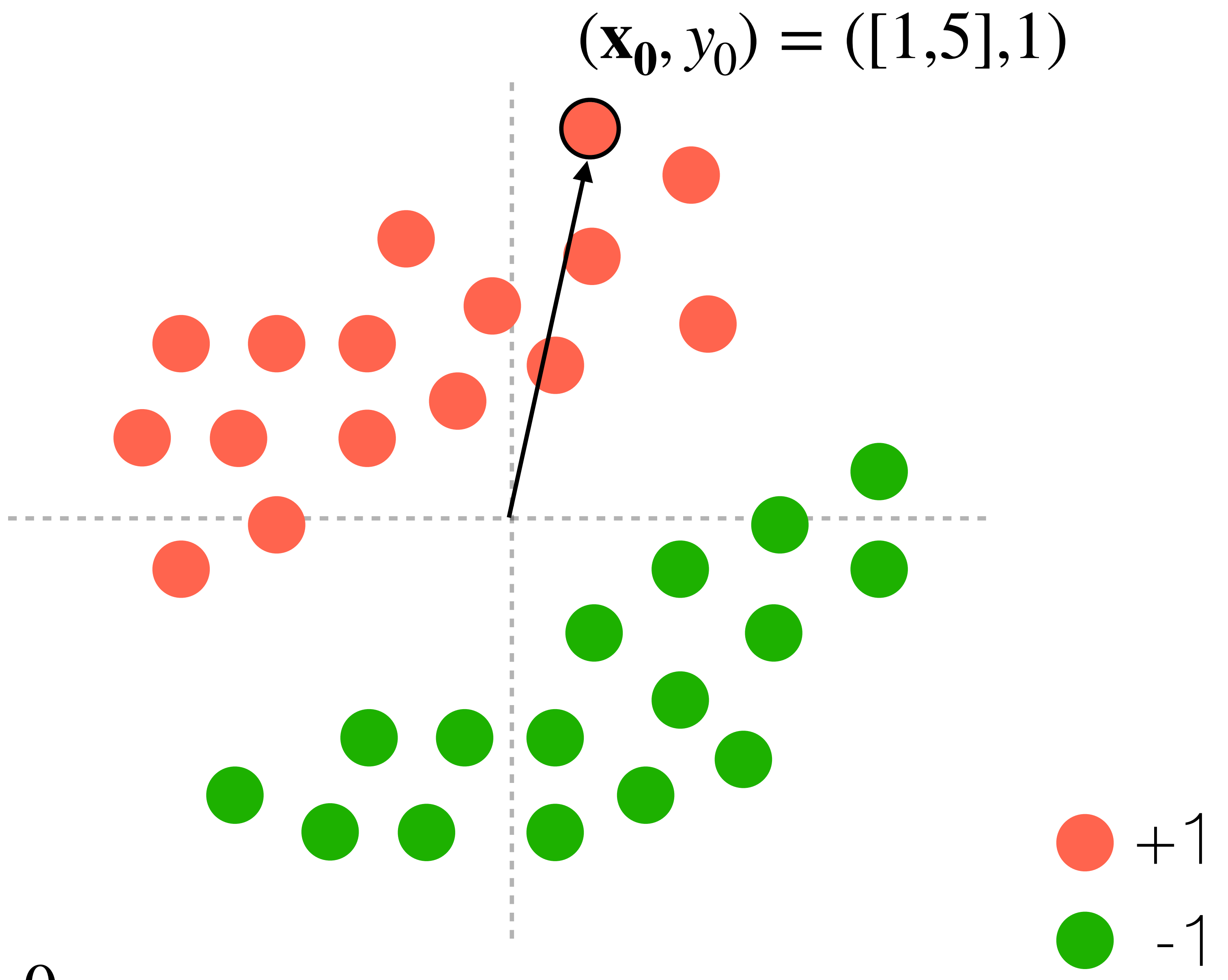
```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [0,0]$

$l = 0$

$sign(\mathbf{w} \cdot \mathbf{x_0}) = 0$

$\mathbf{w} = \mathbf{w} + y_0 \mathbf{x_0}$



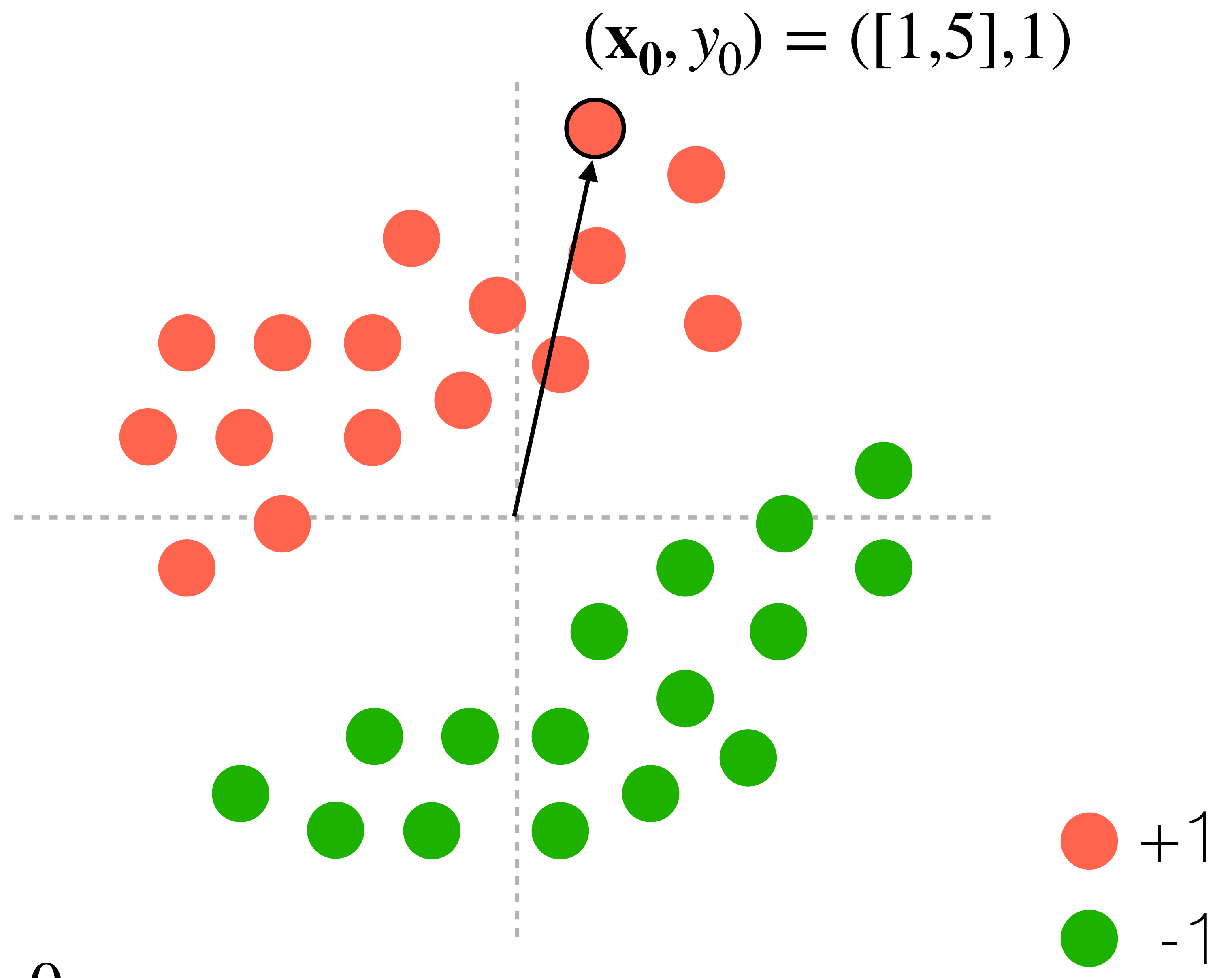
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

LearningRule(D):
  $l \leftarrow |D|$
  $\mathbf{w} \leftarrow [0,0]$
  while $l > 0$:
    $l \leftarrow 0$
    for $(\mathbf{x_i}, y_i)$ **in** D:
      if $sign(\mathbf{w} \cdot \mathbf{x_i}) \neq y_i$:
        $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x_i}$
        $l \leftarrow l + 1$

$\mathbf{w} = [0,0]$

$l = 0$



$sign(\mathbf{w} \cdot \mathbf{x_0}) = 0$

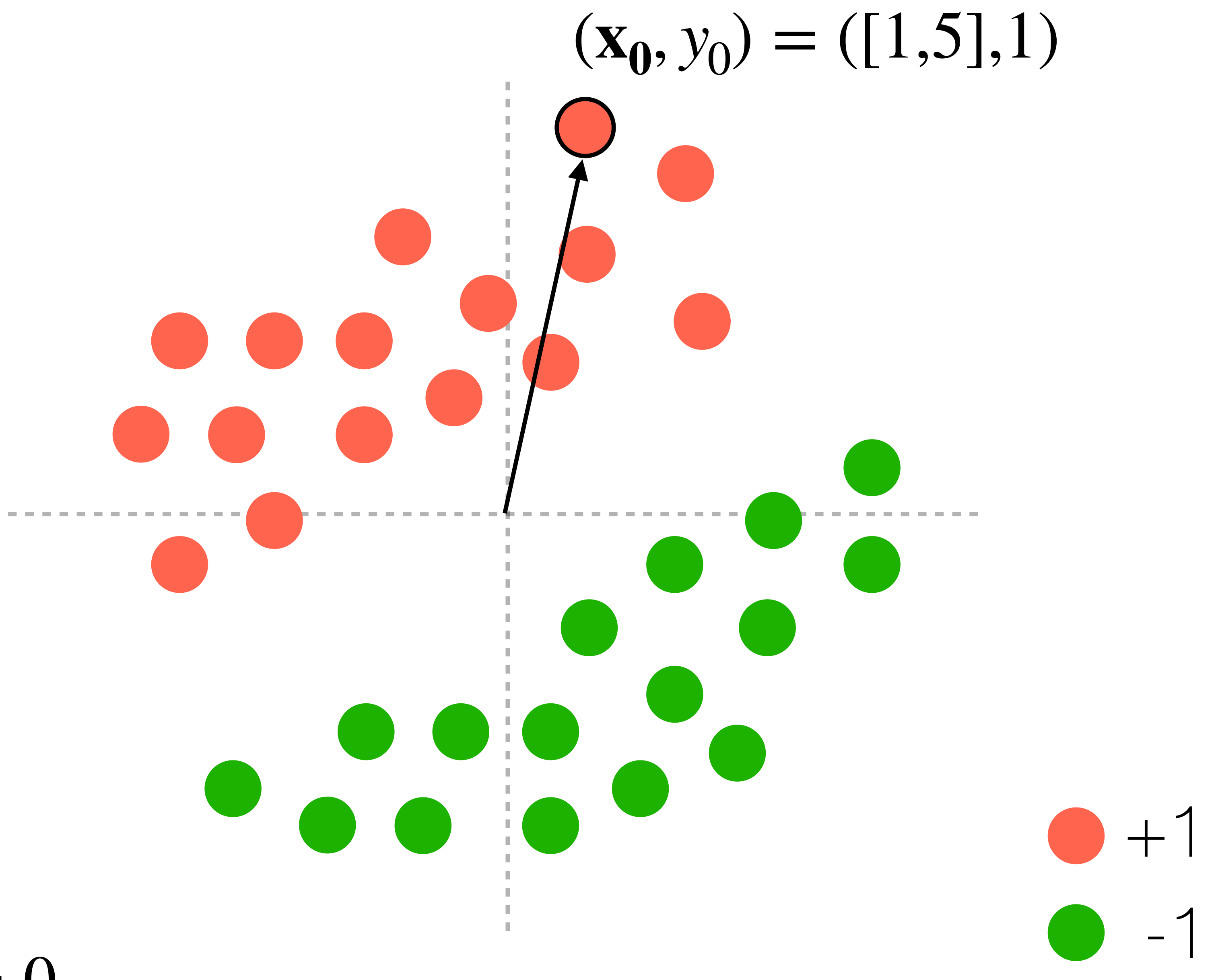
$\mathbf{w} = \mathbf{w} + y_0\mathbf{x_0} = [0,0] + 1 \times [1,5]$

UFV

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



$\mathbf{w} = [0,0]$

$l = 0$

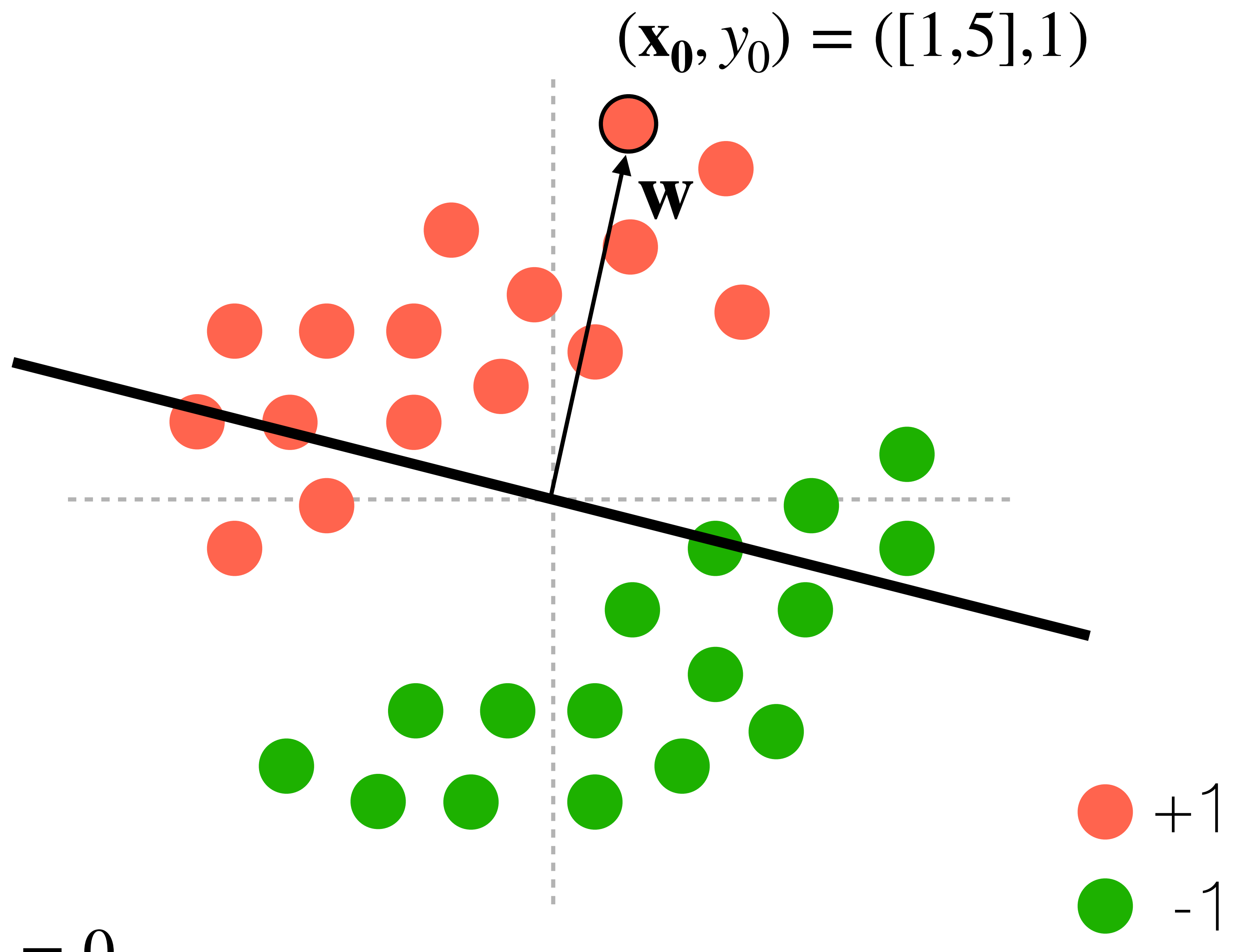
$sign(\mathbf{w} \cdot \mathbf{x_0}) = 0$

$\mathbf{w} = \mathbf{w} + y_0\mathbf{x_0} = [0,0] + 1 \times [1,5] = [1,5]$

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



$\mathbf{w} = [0,0]$

$l = 0$

$sign(\mathbf{w} \cdot \mathbf{x_0}) = 0$

$\mathbf{w} = \mathbf{w} + y_0\mathbf{x_0} = [0,0] + 1 \times [1,5] = [1,5]$

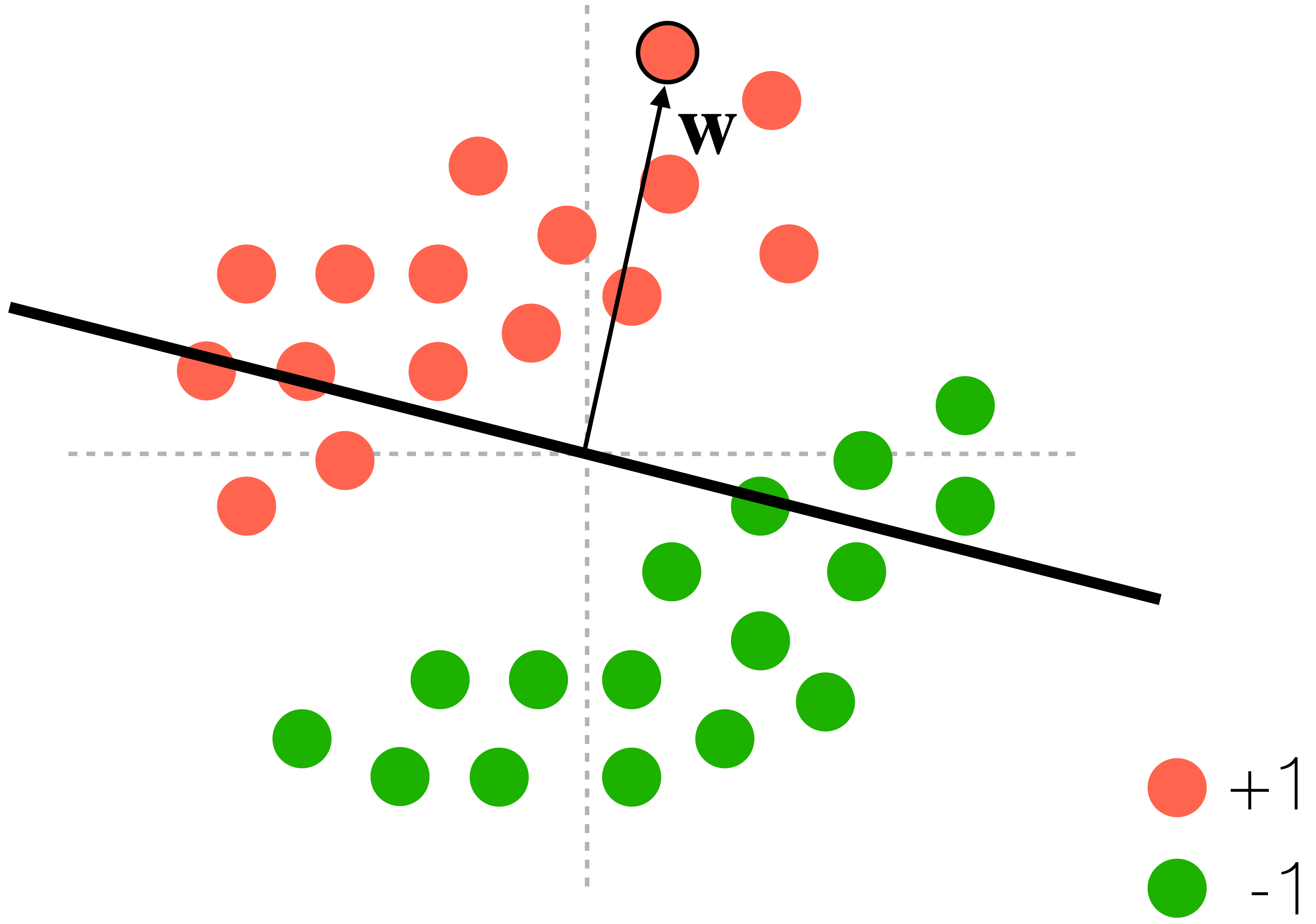
Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

LearningRule(D):
  $l \leftarrow |D|$
  $\mathbf{w} \leftarrow [0{,}0]$
  while $l > 0$:
    $l \leftarrow 0$
    for $(\mathbf{x_i}, y_i)$ **in** D:
      if $sign(\mathbf{w} \cdot \mathbf{x_i}) \neq y_i$:
        $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x_i}$
        $l \leftarrow l + 1$

$l = 0$

$\mathbf{w} = [1{,}5]$

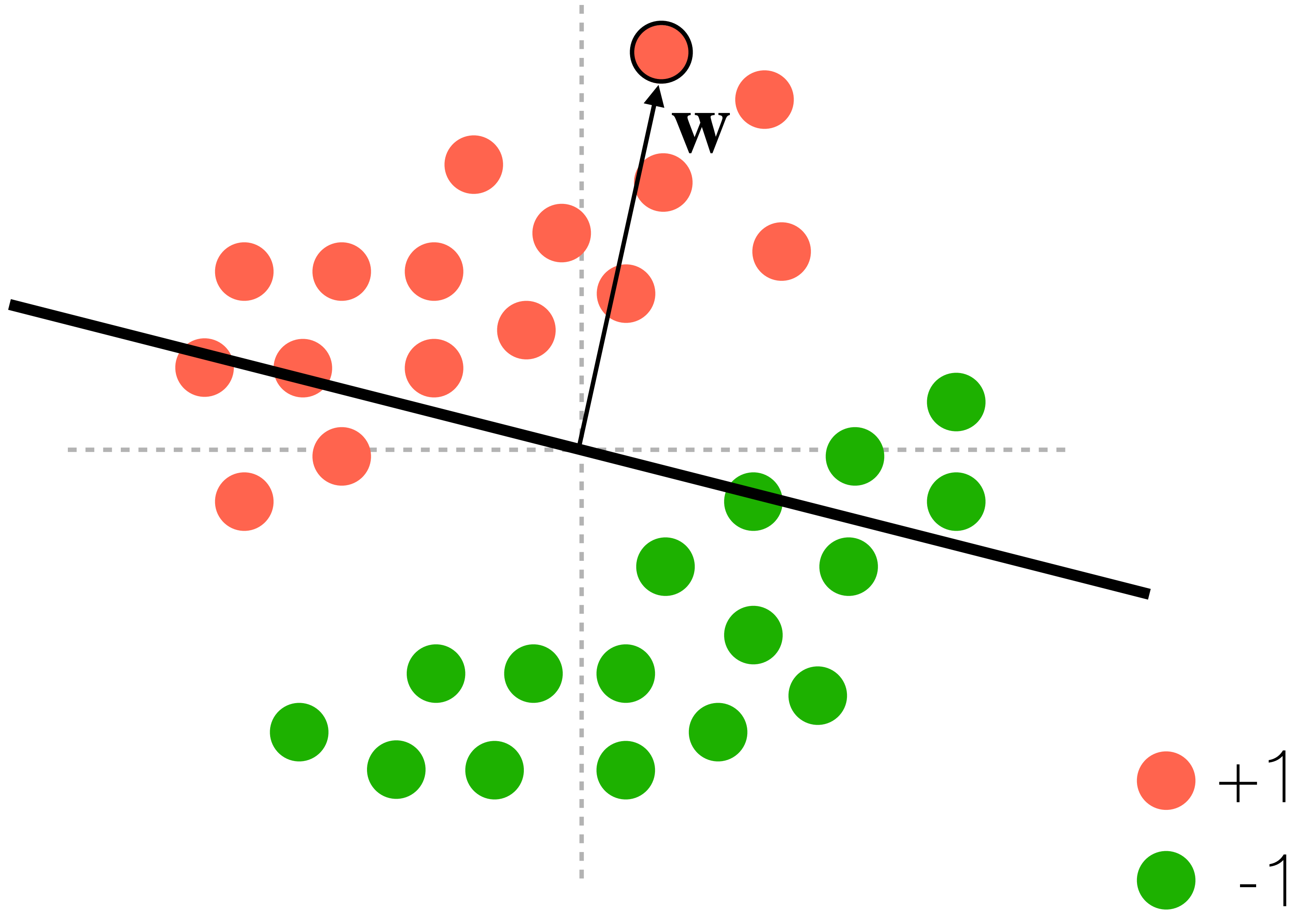


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

LearningRule(D)**:**
  $l \leftarrow |D|$
  $\mathbf{w} \leftarrow [0{,}0]$
  while $l > 0$**:**
    $l \leftarrow 0$
    for $(\mathbf{x_i}, y_i)$ **in** D**:**
      if $sign(\mathbf{w} \cdot \mathbf{x_i}) \neq y_i$**:**
        $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x_i}$
        $l \leftarrow l + 1$

$\mathbf{w} = [1{,}5]$
$l = 1$

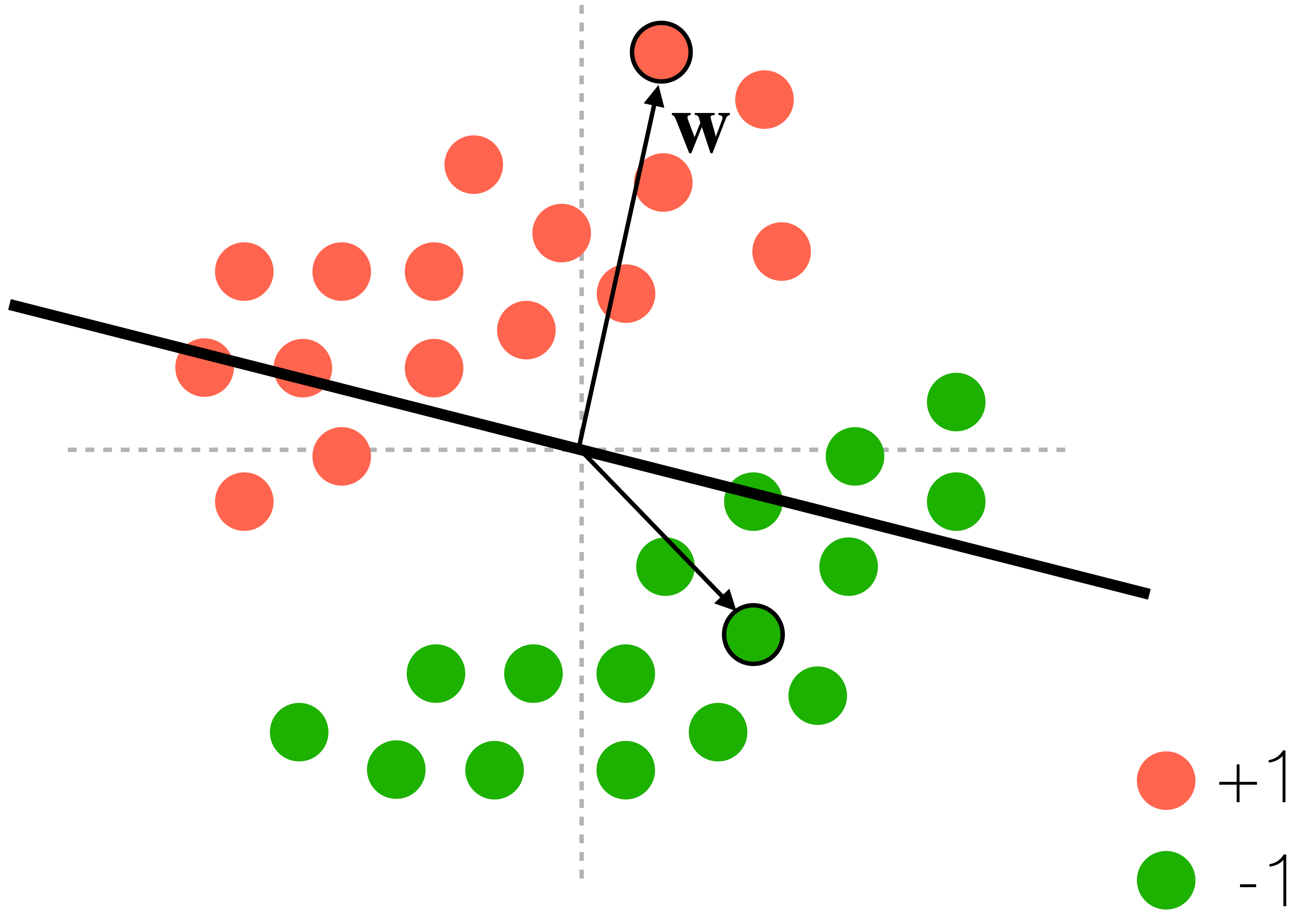


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [1,5]$
$l = 1$

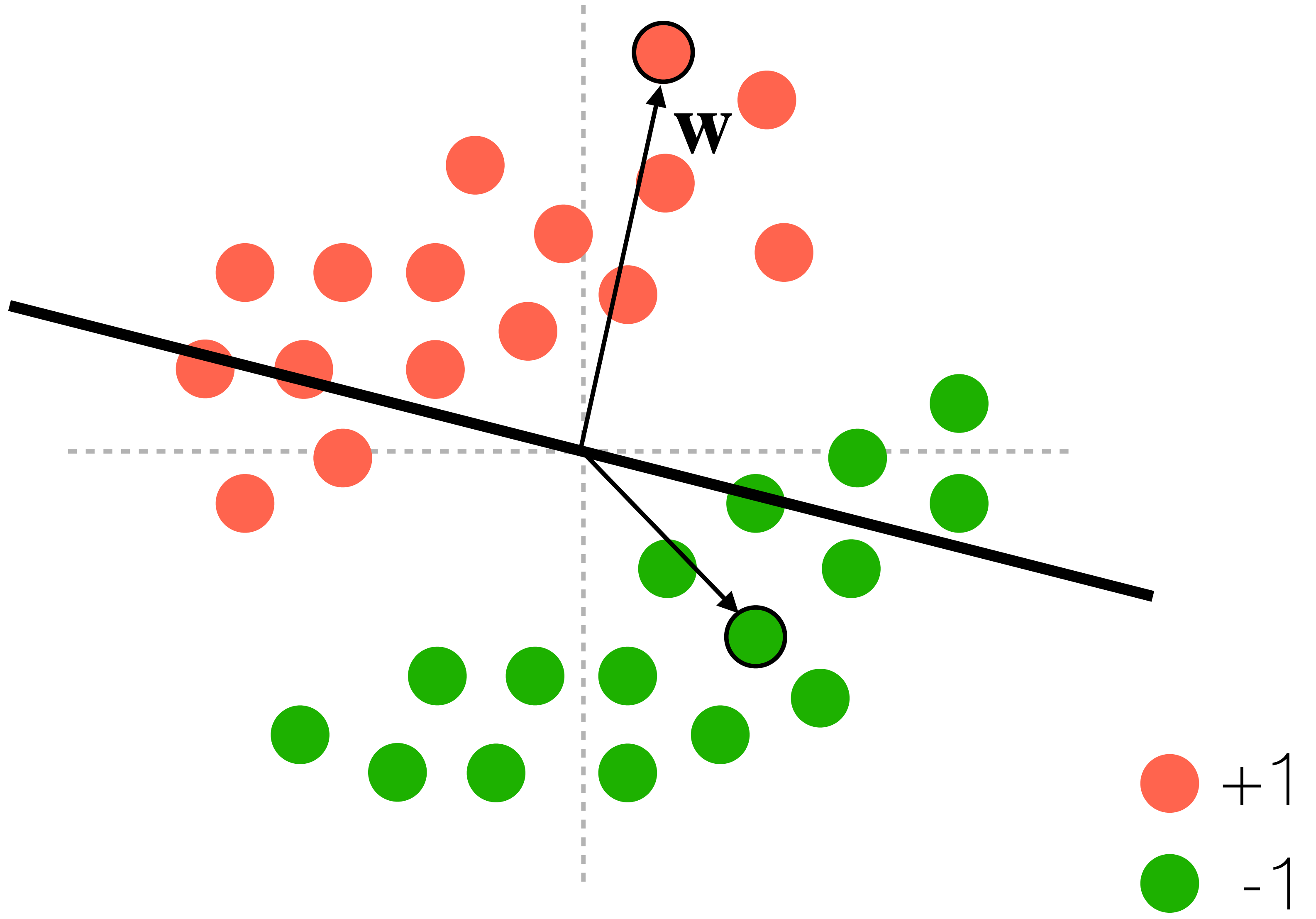


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [1,5]$
$l = 1$

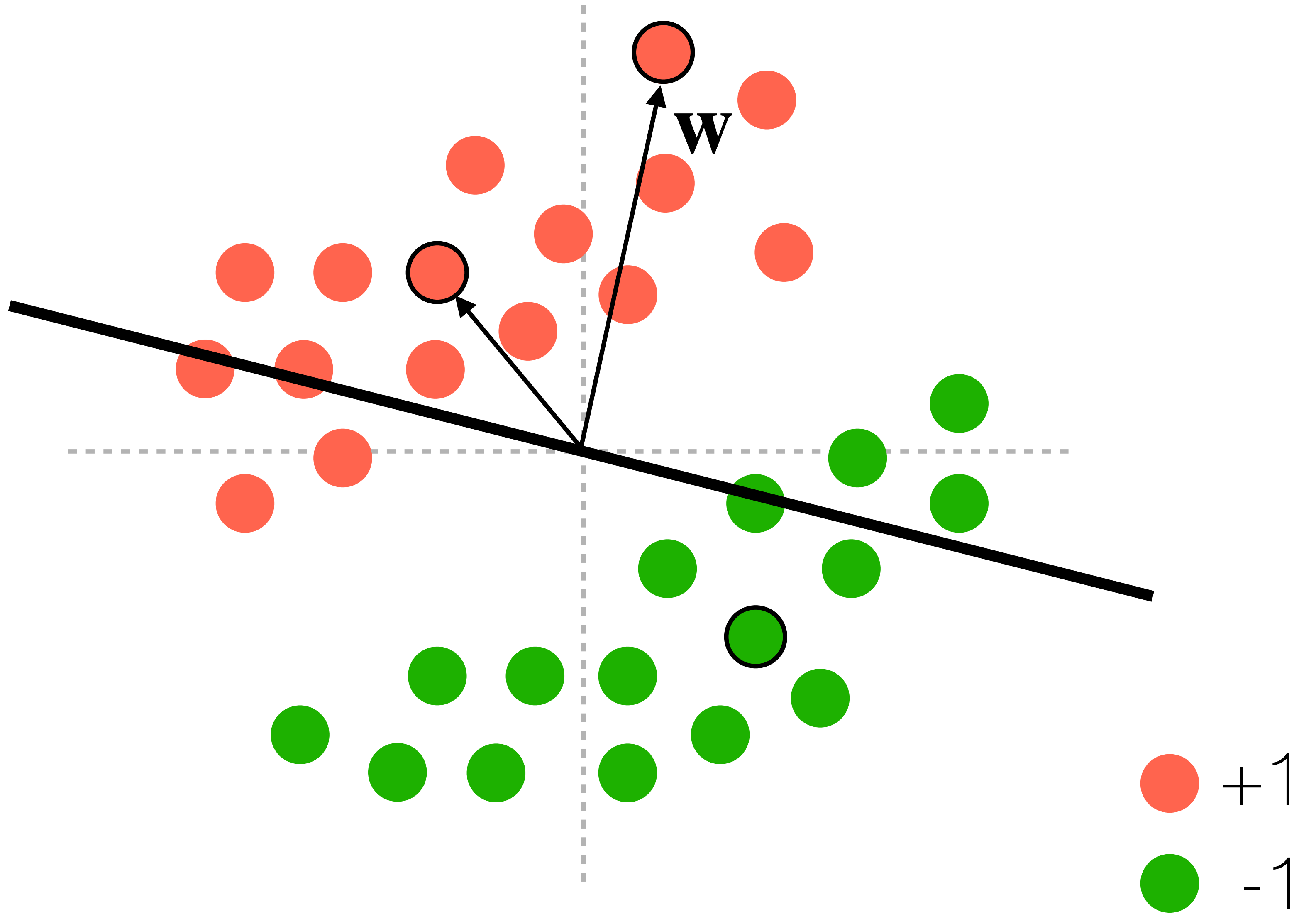


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [1{,}5]$
$l = 1$

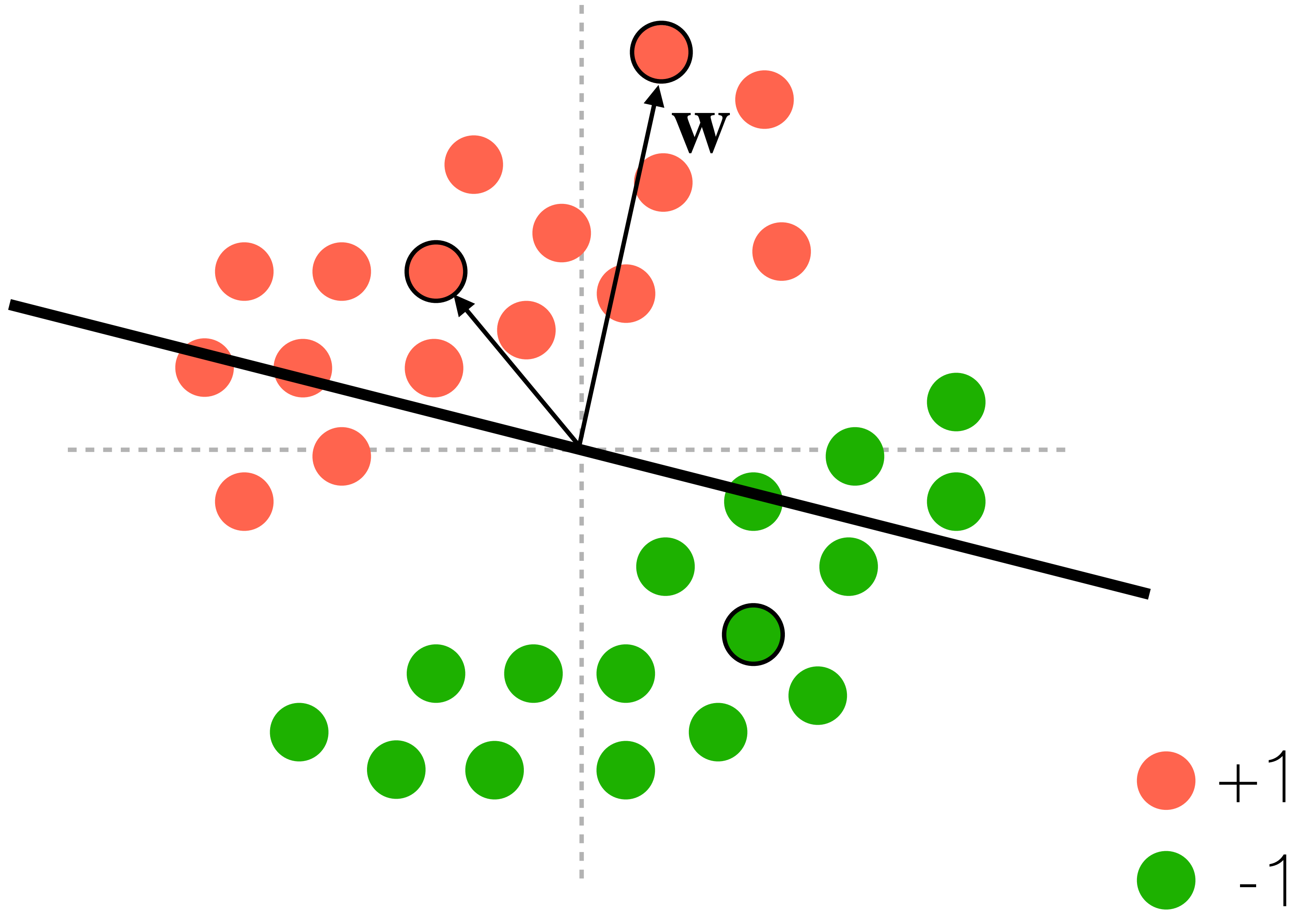


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

$\mathbf{w} = [1,5]$
$l = 1$

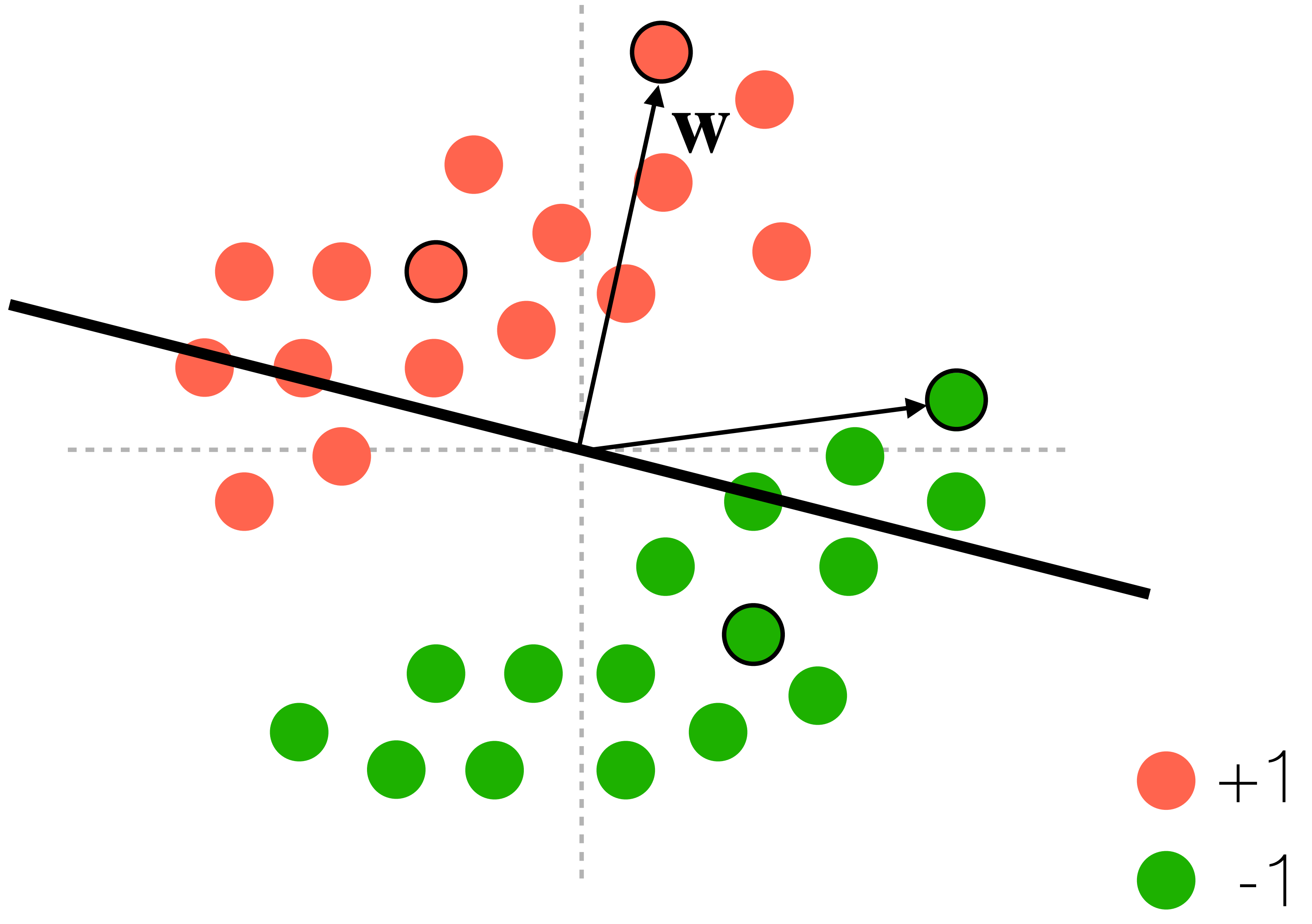


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

LearningRule(D)**:**
  $l \leftarrow |D|$
  $\mathbf{w} \leftarrow [0{,}0]$
  while $l > 0$**:**
    $l \leftarrow 0$
    for $(\mathbf{x_i}, y_i)$ **in** D**:**
      if $sign(\mathbf{w} \cdot \mathbf{x_i}) \neq y_i$**:**
        $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x_i}$
        $l \leftarrow l + 1$

$\mathbf{w} = [1{,}5]$
$l = 1$

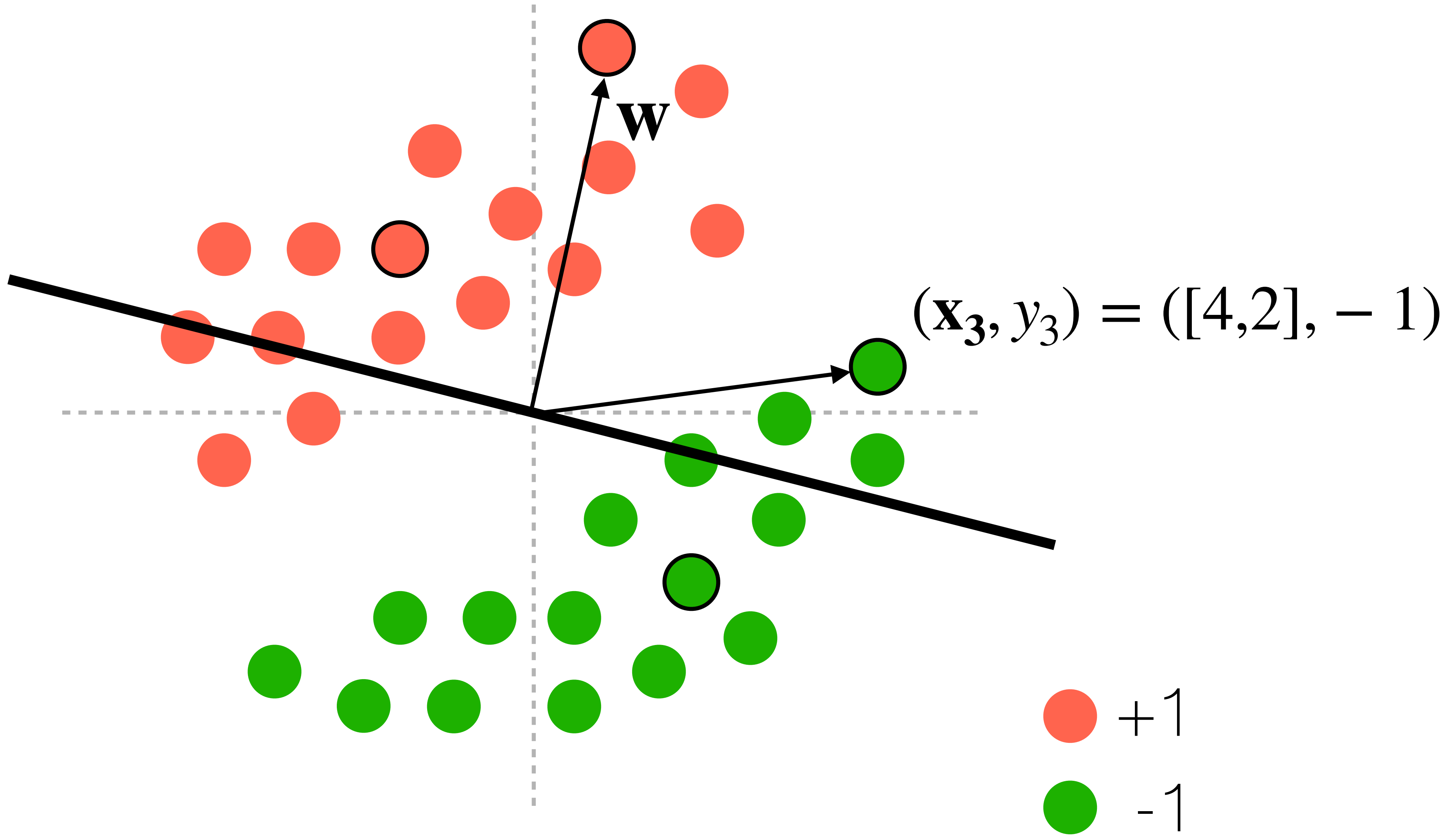


Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

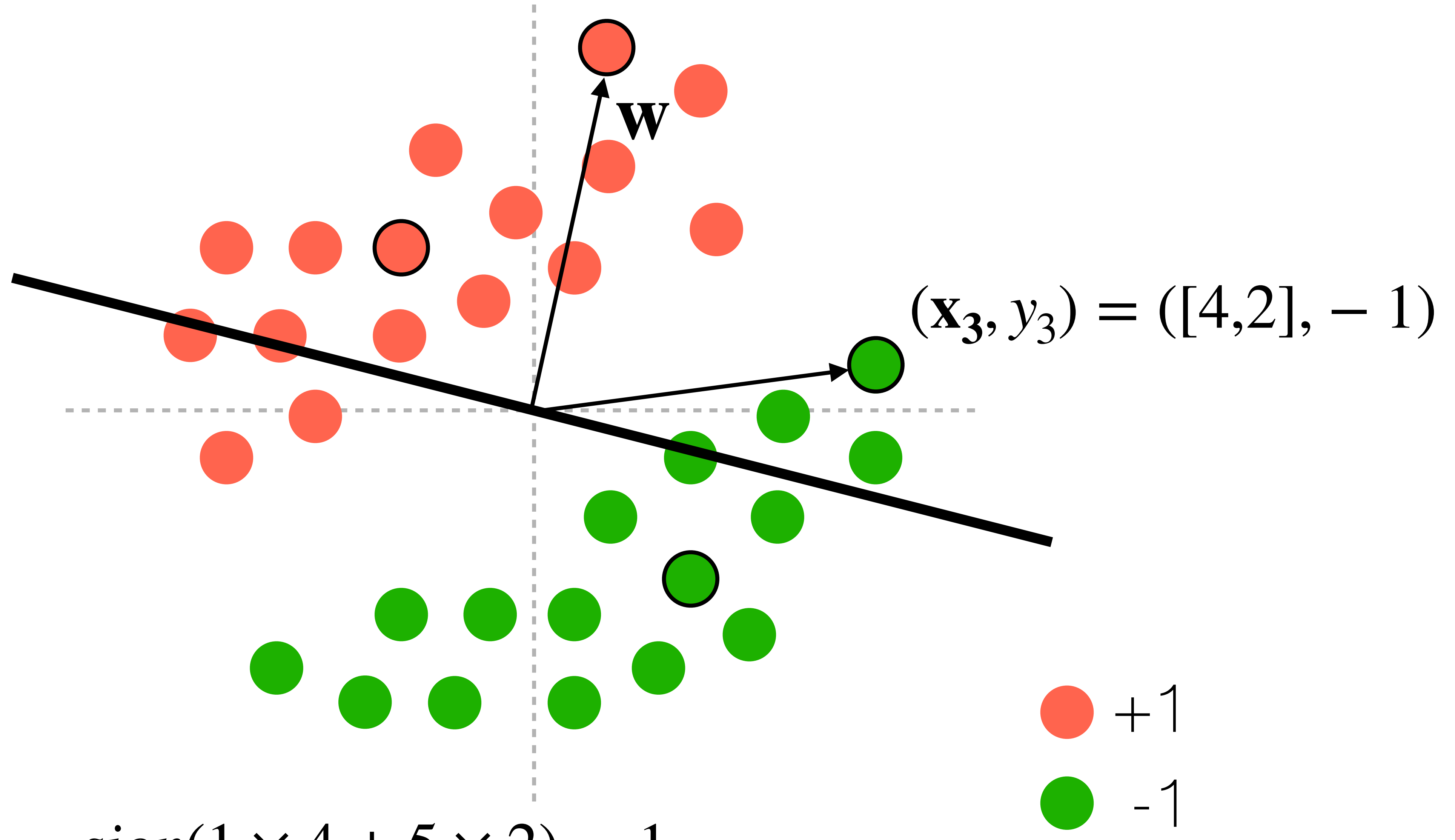
$\mathbf{w} = [1,5]$
$l = 1$



Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



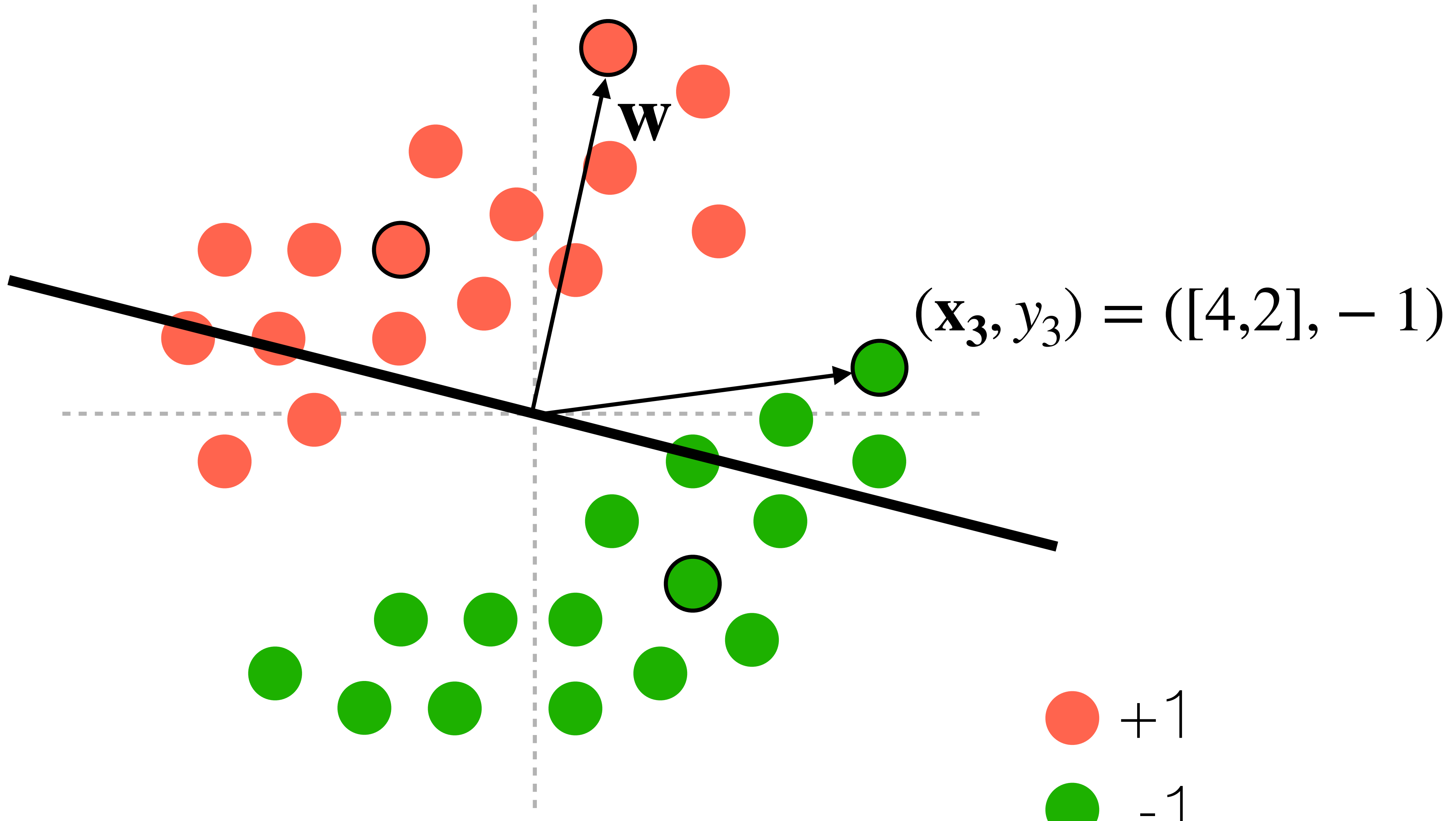
$sign(\mathbf{w} \cdot \mathbf{x_3}) = sign(1 \times 4 + 5 \times 2) = 1$

$\mathbf{w} = [1,5]$
$l = 1$

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



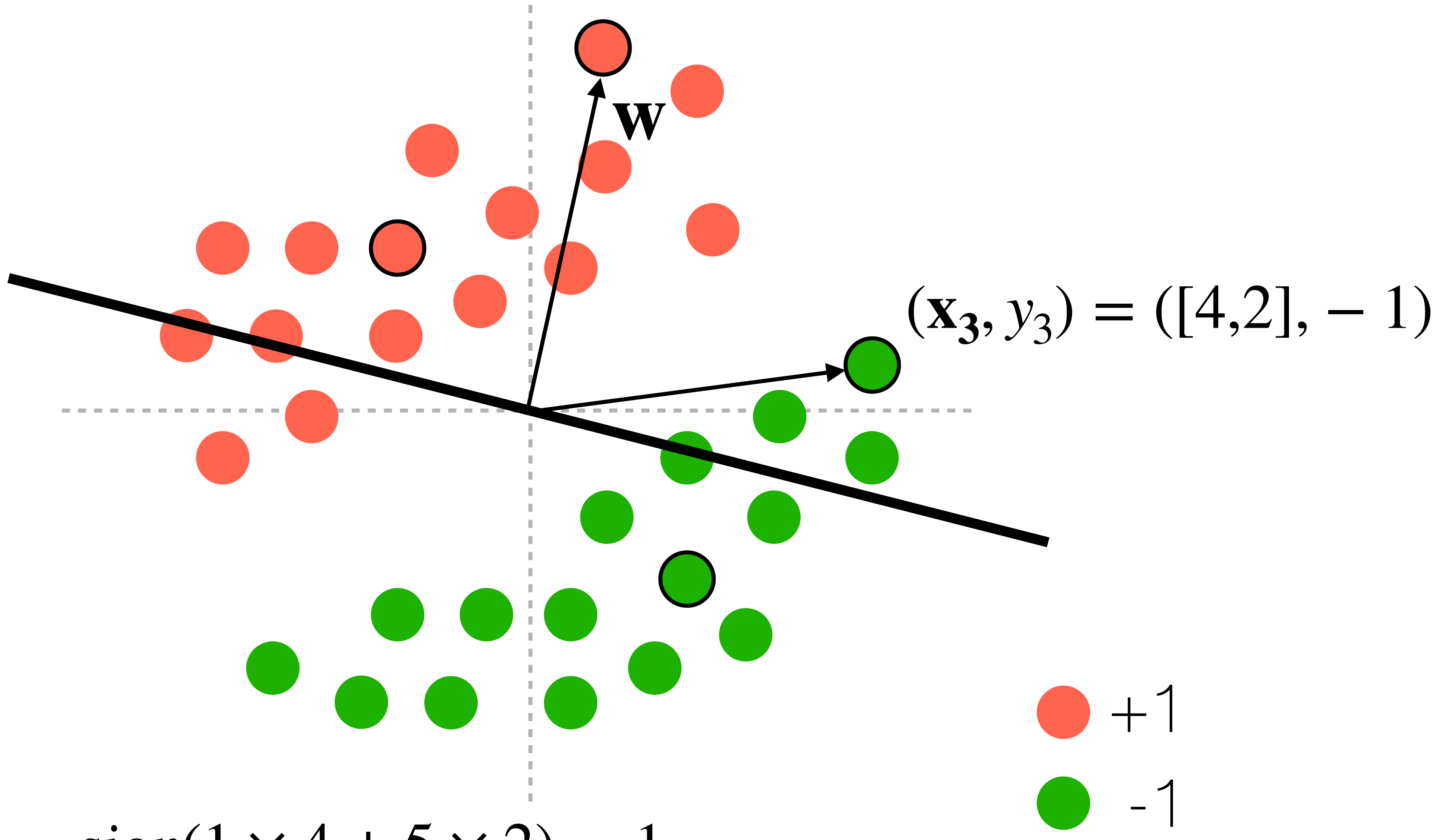
$sign(\mathbf{w} \cdot \mathbf{x_3}) = sign(1 \times 4 + 5 \times 2) = 1$

$\mathbf{w} = \mathbf{w} + y_3\mathbf{x_3}$
$l = 1$

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



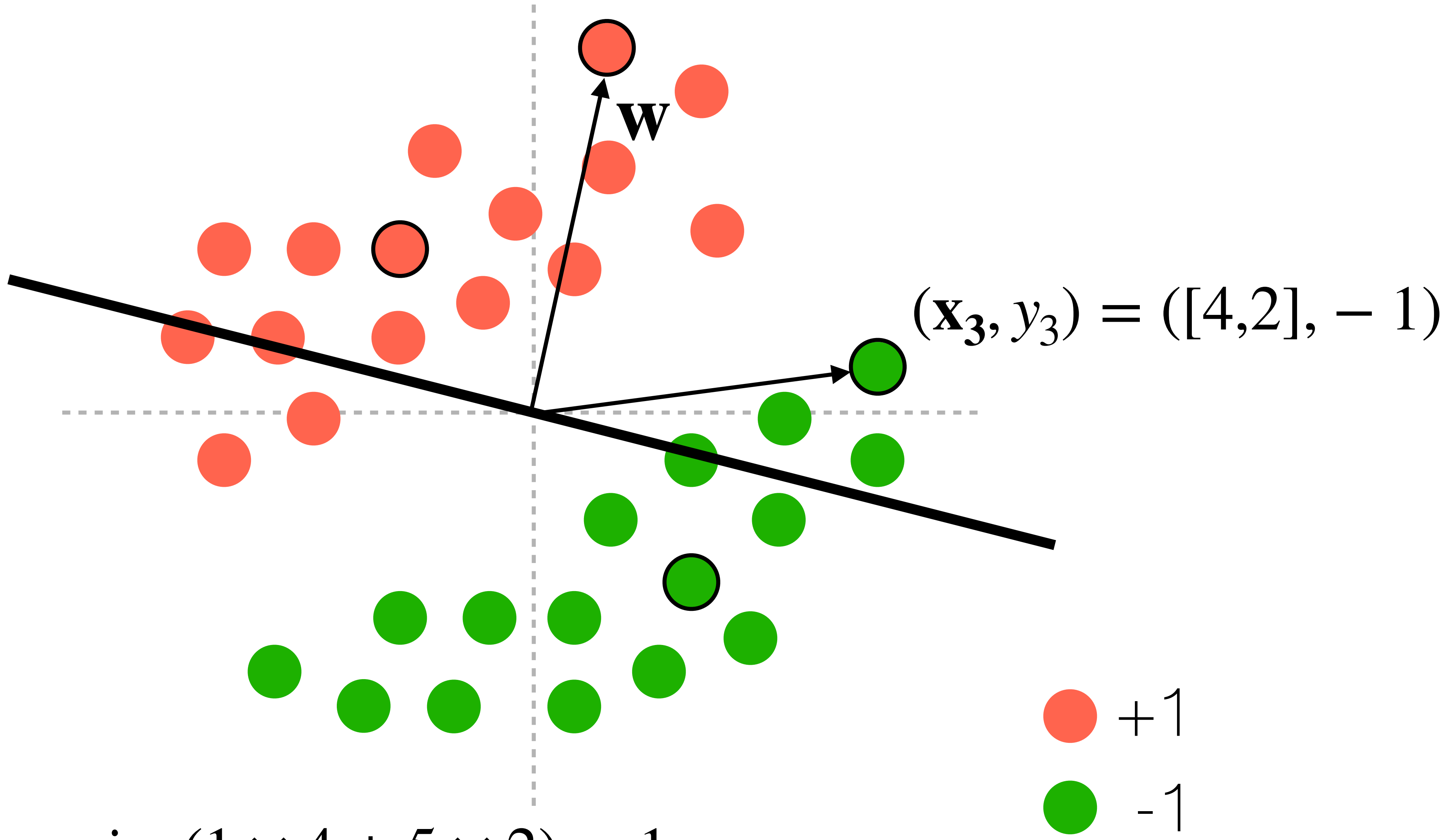
$sign(\mathbf{w} \cdot \mathbf{x_3}) = sign(1 \times 4 + 5 \times 2) = 1$

$\mathbf{w} = \mathbf{w} + y_3\mathbf{x_3} = [1,5] + (-1) \times [4,2]$
$l = 1$

Treinando o perceptron

A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```



$sign(\mathbf{w} \cdot \mathbf{x_3}) = sign(1 \times 4 + 5 \times 2) = 1$

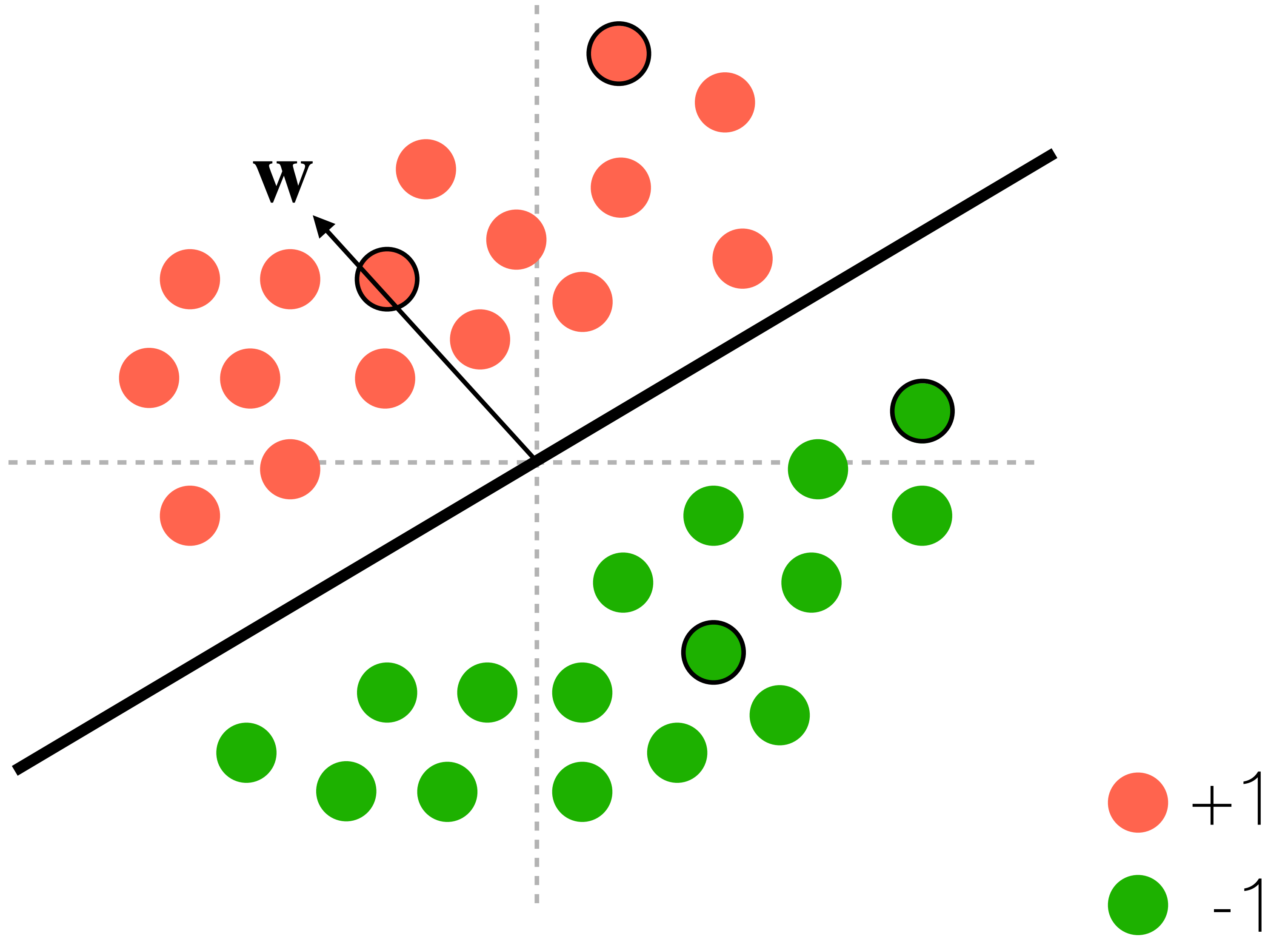
$\mathbf{w} = \mathbf{w} + y_3\mathbf{x_3} = [1,5] + (-1) \times [4,2] \;= [-3,3]$
$l = 1$

Treinando o perceptron

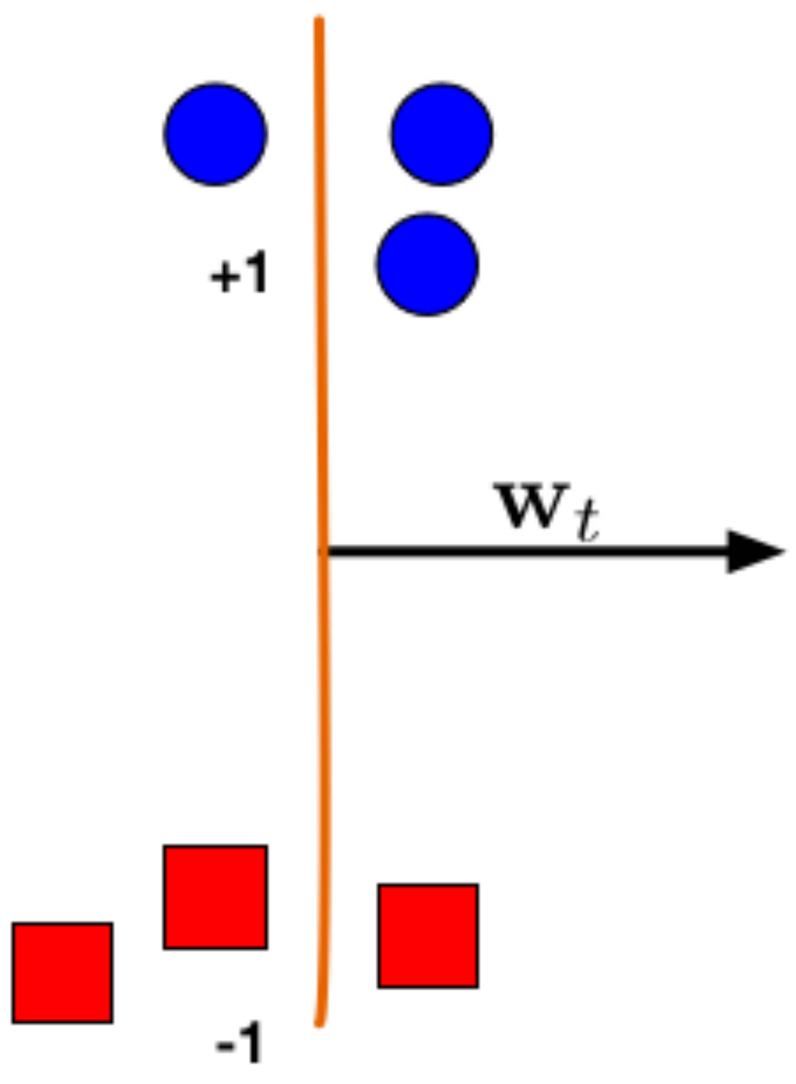
A **regra de atualização do perceptron** é o algoritmo utilizado para aprender os pesos $\mathbf{w}$ e b

```
LearningRule(D):
  l ← |D|
  w ← [0,0]
  while l > 0:
    l ← 0
    for (x_i, y_i) in D:
      if sign(w · x_i) ≠ y_i:
        w ← w + y_i x_i
        l ← l + 1
```

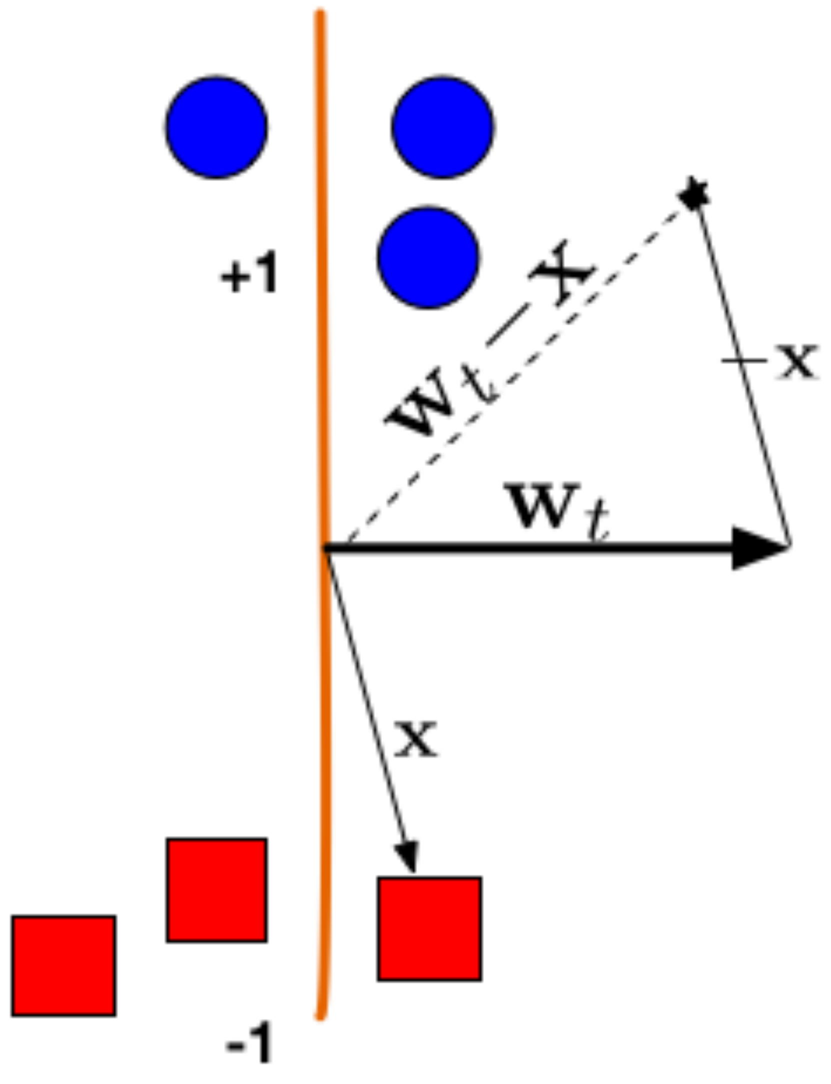
$\mathbf{w} = [-3,3]$
$l = 1$



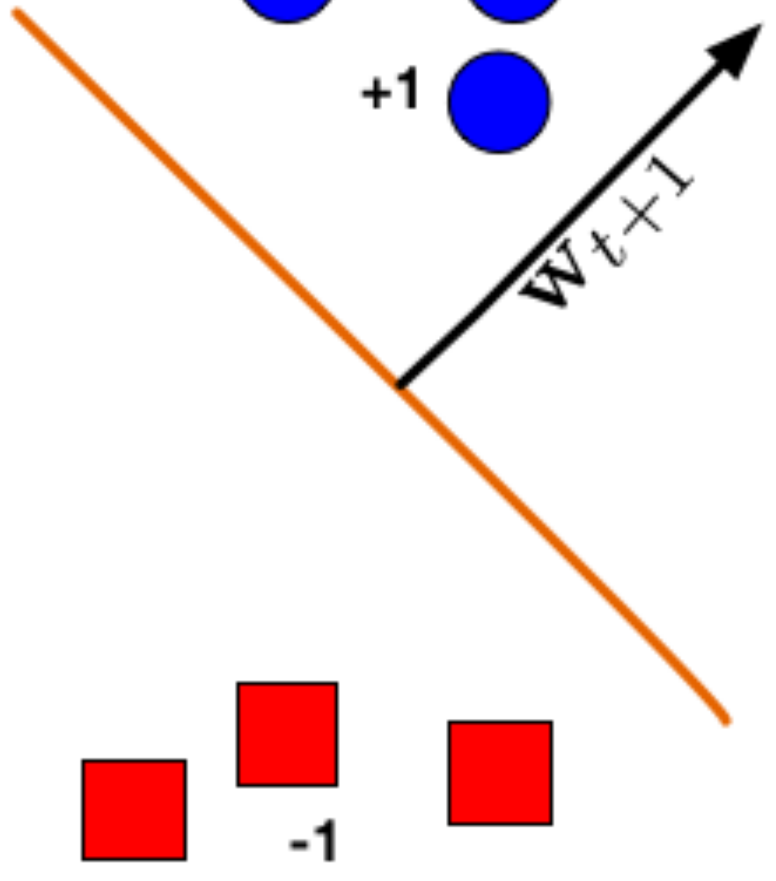
Interpretação geométrica da regra de atualização



1. A reta definida por w_t classifica erroneamente um ponto vermelho (-1) e um azul (+1)



2. O ponto vermelho x é escolhido e usado para uma atualização. Como seu rótulo é -1, precisamos subtrair x de w_t.

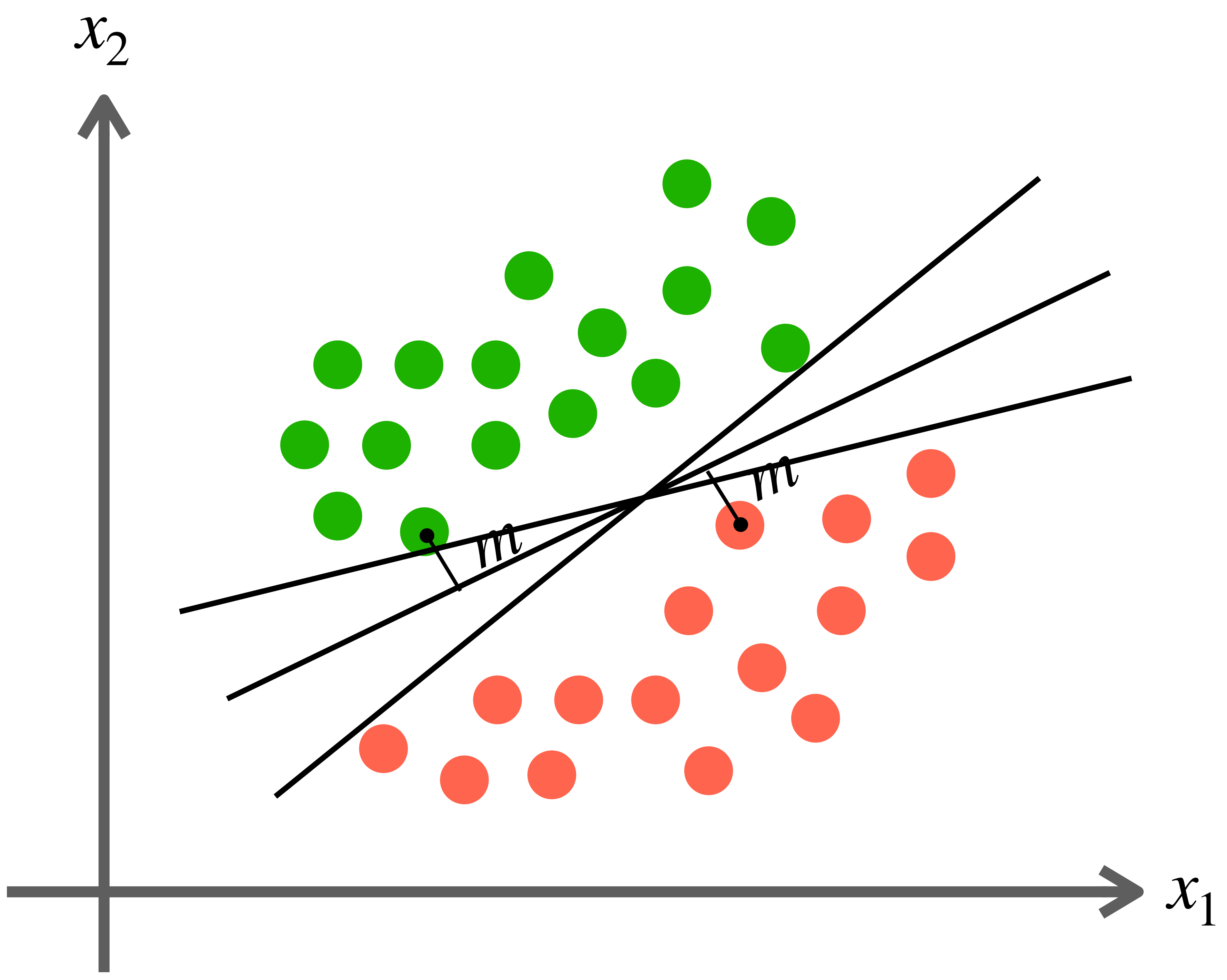


3. O hiperplano atualizado $w_{t+1} = w_t - x$ separa as duas classes e o Perceptron convergiu.

UFV

Support Vector Machine (SVM)

O SVM pode ser visto como uma extensão do Perceptron. O Perceptron garante que um hiperplano será encontrado, se ele existir. O SVM encontra o hiperplano com **margem** máxima.



Se os dados são linearmente separáveis, existem infinitos híperplanos que separam os dados

Qual o melhor?

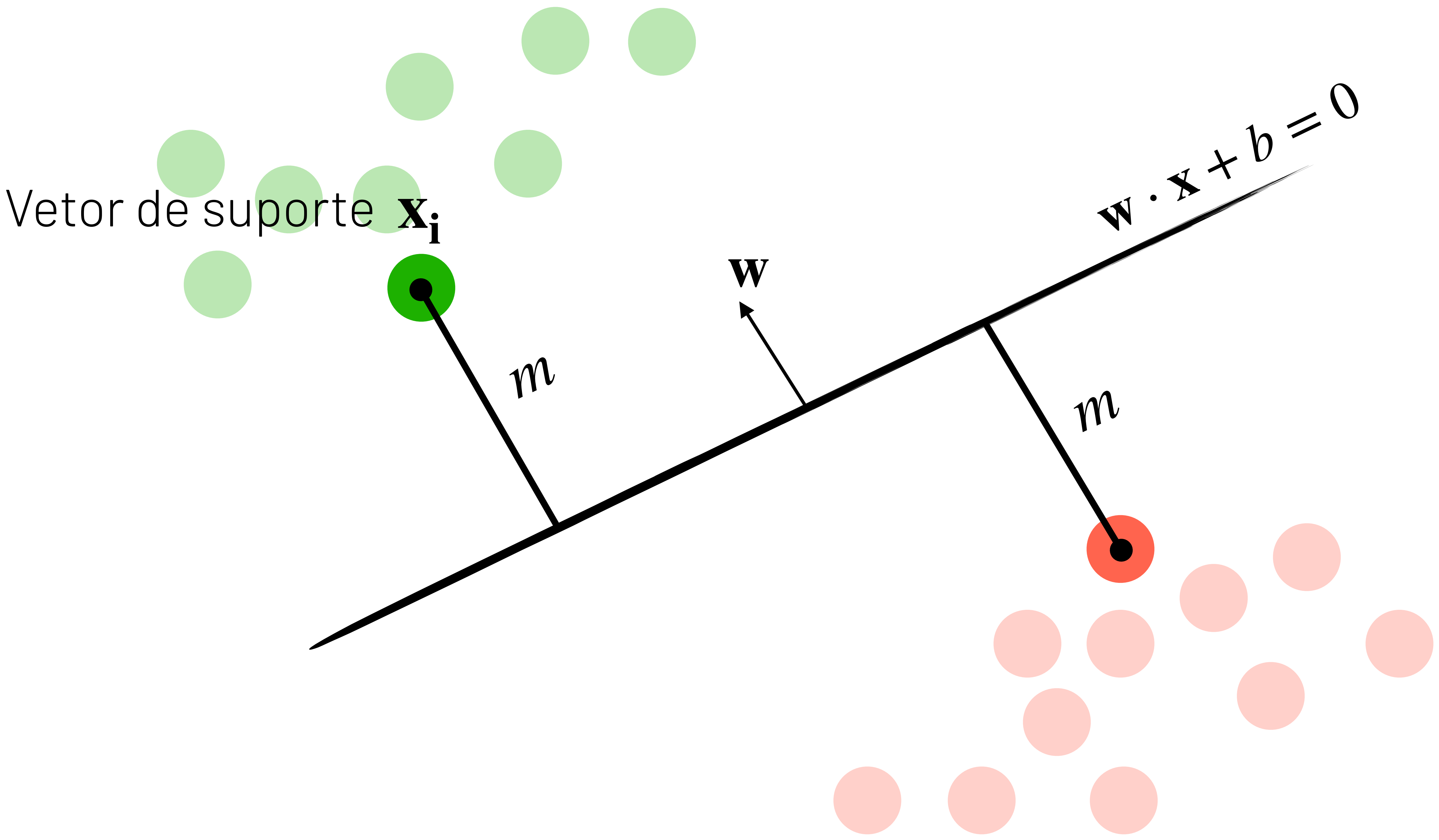
Aquele com maior *margem* m, ou seja, maior distância entre os pontos mais próximos de cada classe

Porque?

Maior generalição do modelo

Margem

A **margem** é a distância entre os pontos mais próximos de cada classe



Usando Algebra Linear

Distância d entre um ponto x_i e um hiperplano definido por $\mathbf{w}$ e b:

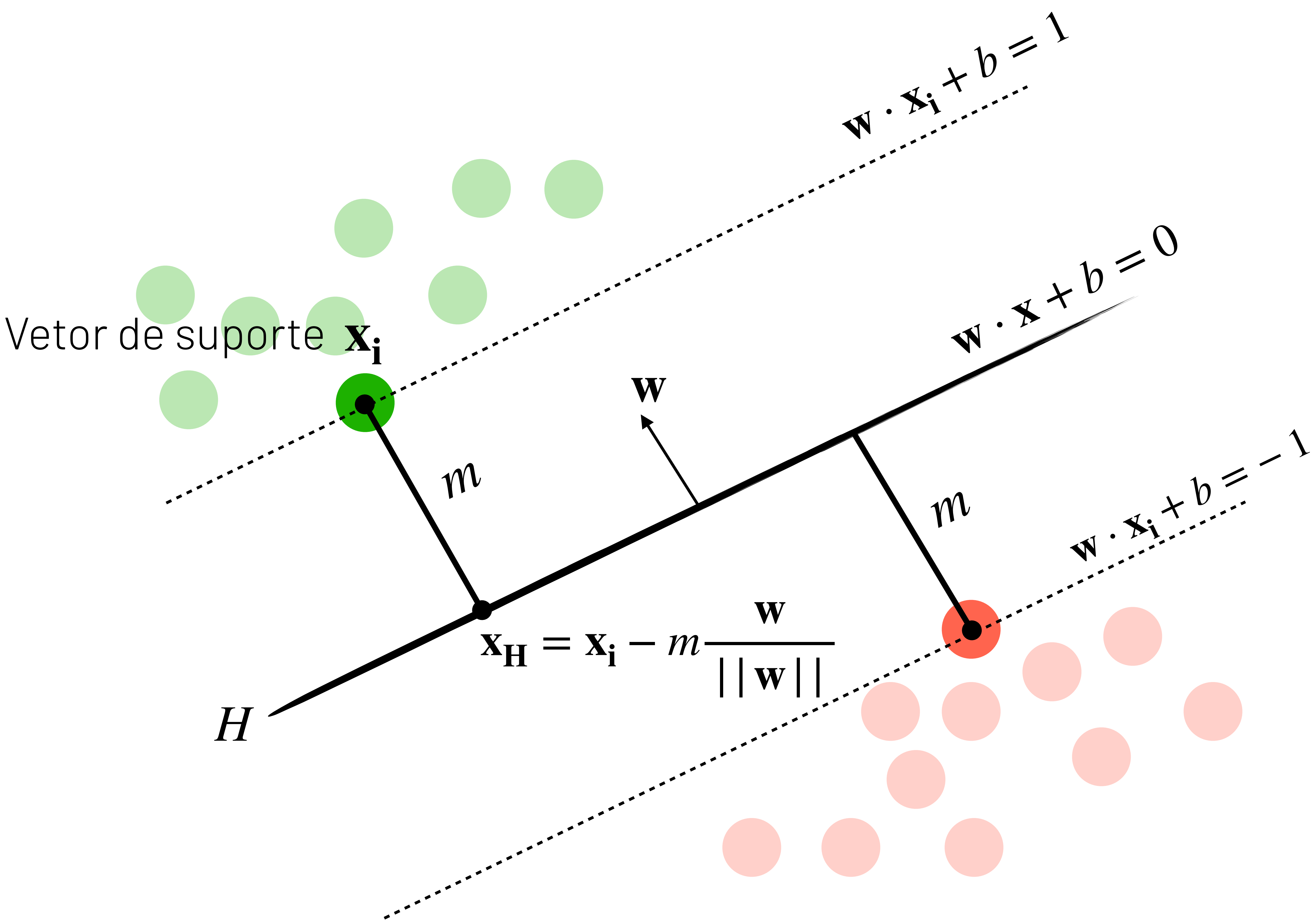
$$d = \frac{|\mathbf{w} \cdot \mathbf{x_i} + b|}{||\mathbf{w}||}$$

Margem

Menor distância m entre os pontos $\mathbf{x_i} \in D$ e o hiperplano definido por $\mathbf{w}$ e b:

$$m = min_{\mathbf{x_i} \in D} \frac{|\mathbf{w} \cdot \mathbf{x_i} + b|}{||\mathbf{w}||}$$

Maximizando a margem



Queremos encontrar parâmetros $\mathbf{w}$ e b que maximizem m. Portanto, precisamos definir m em função de $\mathbf{w}$.

Assumindo $\mathbf{x_i}$ é um vetor de suporte positivo e o projetando em H, temos $\mathbf{x_H} \in H$:

$$\mathbf{w} \cdot (\mathbf{x_i} - m\frac{\mathbf{w}}{||\mathbf{w}||}) + b = 0$$

$$(\mathbf{w} \cdot \mathbf{x_i}) - ((\mathbf{w} \cdot \mathbf{w})\frac{m}{||\mathbf{w}||}) + b = 0$$

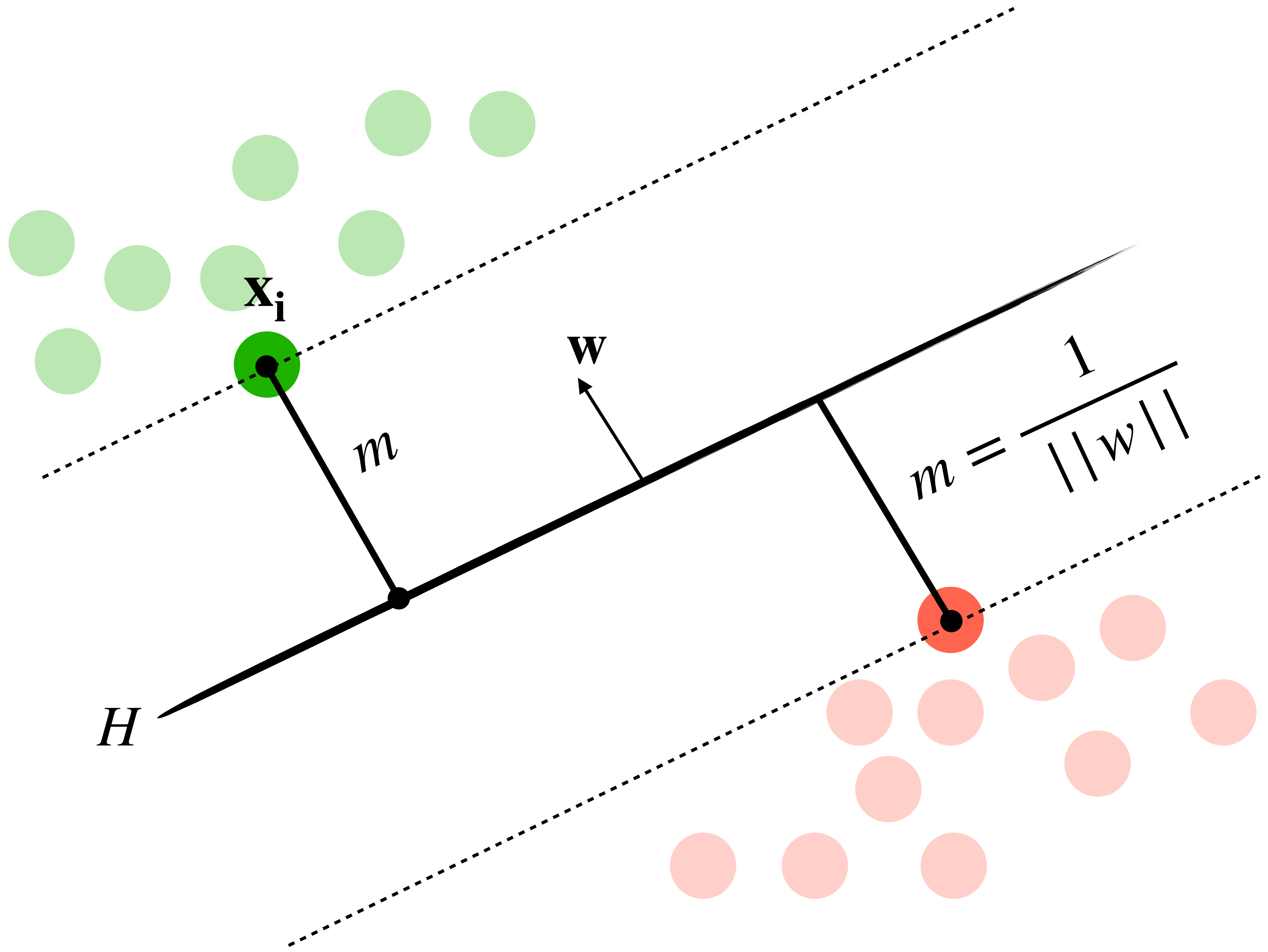
$$(\mathbf{w} \cdot \mathbf{x_i}) - (||\mathbf{w}||^2\frac{m}{||\mathbf{w}||}) + b = 0$$

$$\mathbf{w} \cdot \mathbf{x_i} + b - ||\mathbf{w}||m = 0$$

$$1 - ||\mathbf{w}||m = 0$$

$$m = \frac{1}{||\mathbf{w}||}$$

Treinando o SVM



Queremos minimizar $||\mathbf{w}||$ com a restrição de que $y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1$. Essa restrição garante que H separe os dados corretamente.

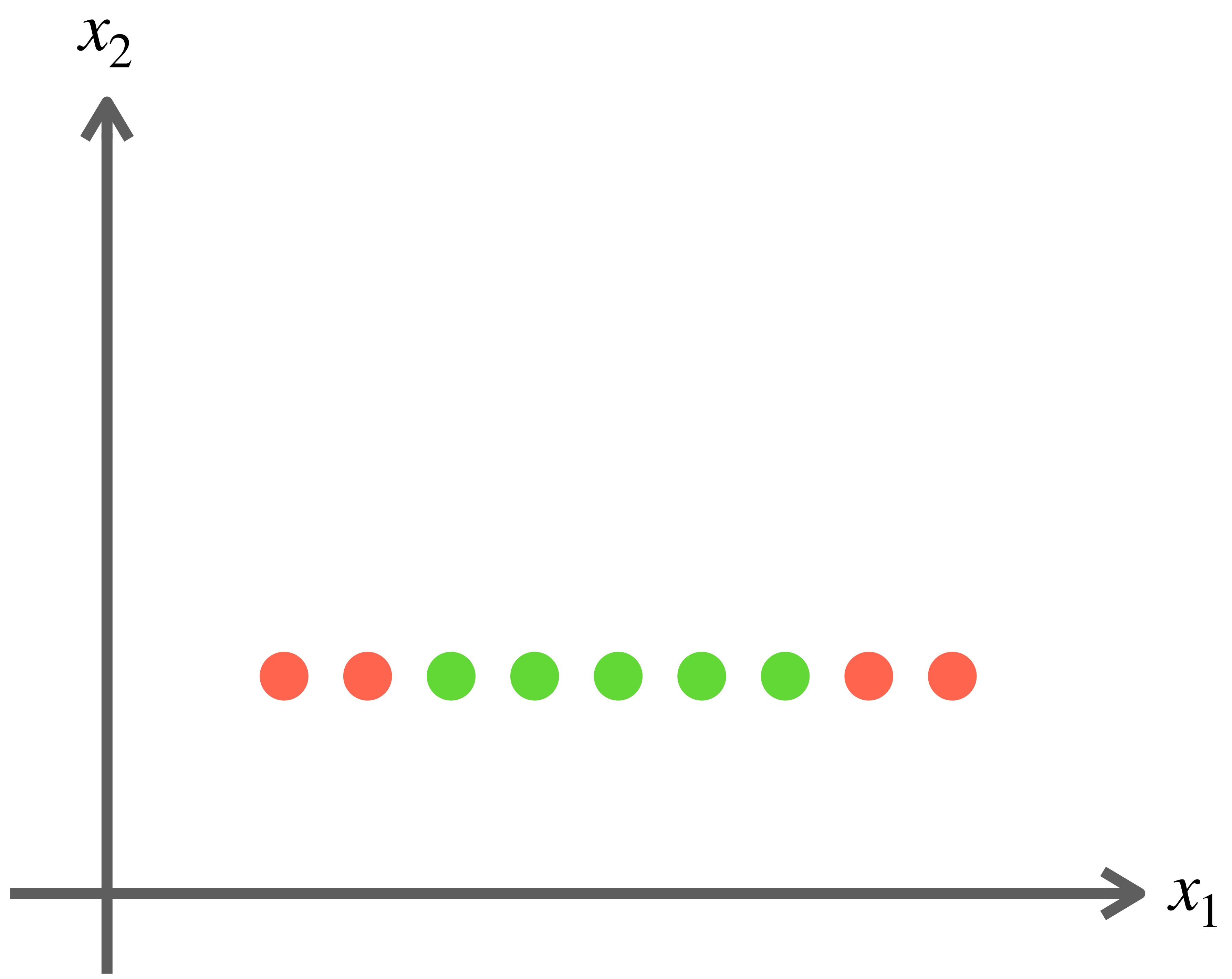
$y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1$ é uma maneira compacta de escrever:

$$\mathbf{w} \cdot \mathbf{x} + b \geq +1, \quad se\, y_i = +1$$
$$\mathbf{w} \cdot \mathbf{x} + b \leq -1, \quad se\, y_i = -1$$

Esse problema de otimização pode ser resolvido de forma fechada com **programação quadrática**

Problemas não-linearmente separáveis

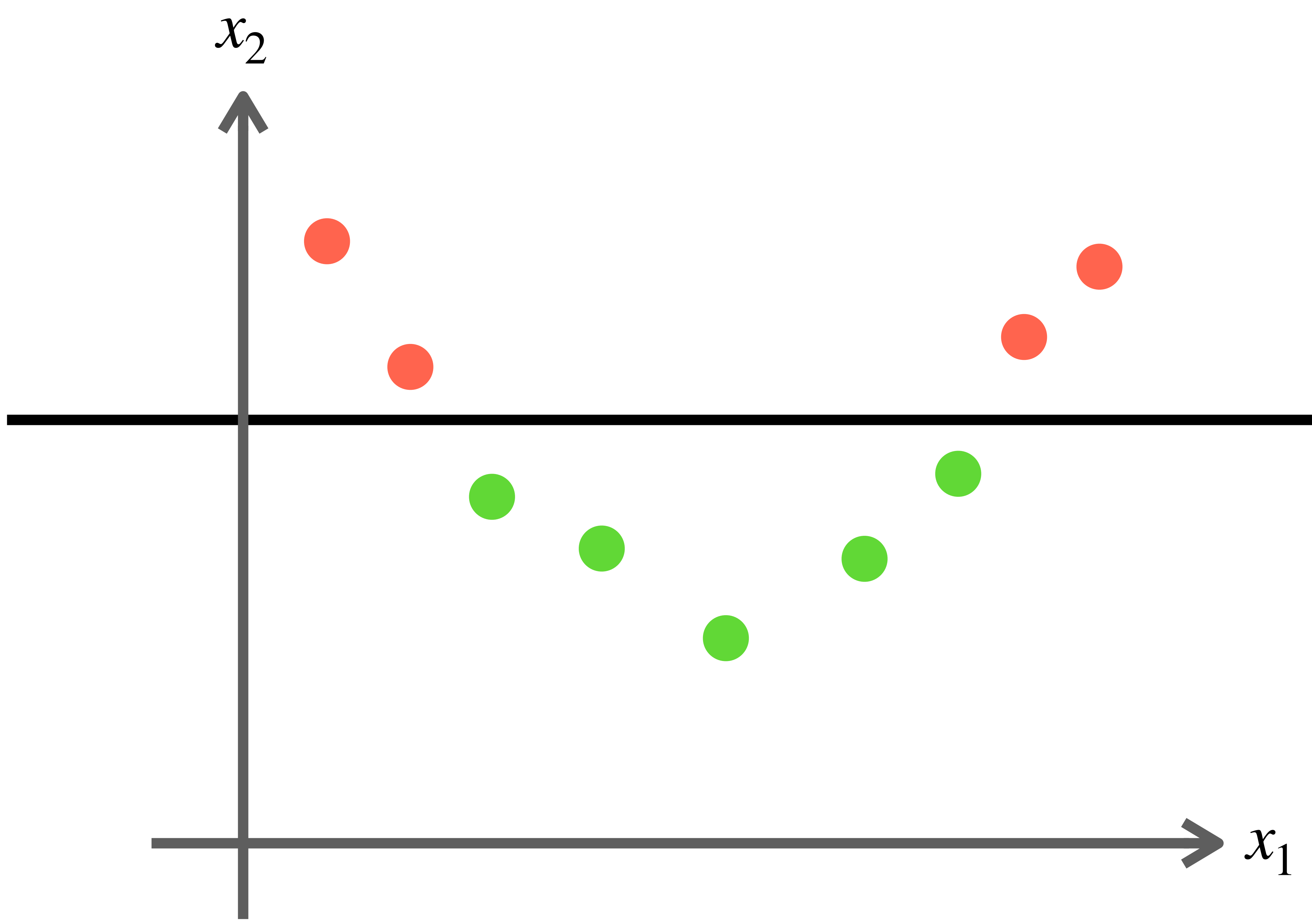
E se os dados não forem linearmente separáveis?



UFV

Problemas não-linearmente separáveis

E se os dados não forem linearmente separáveis?

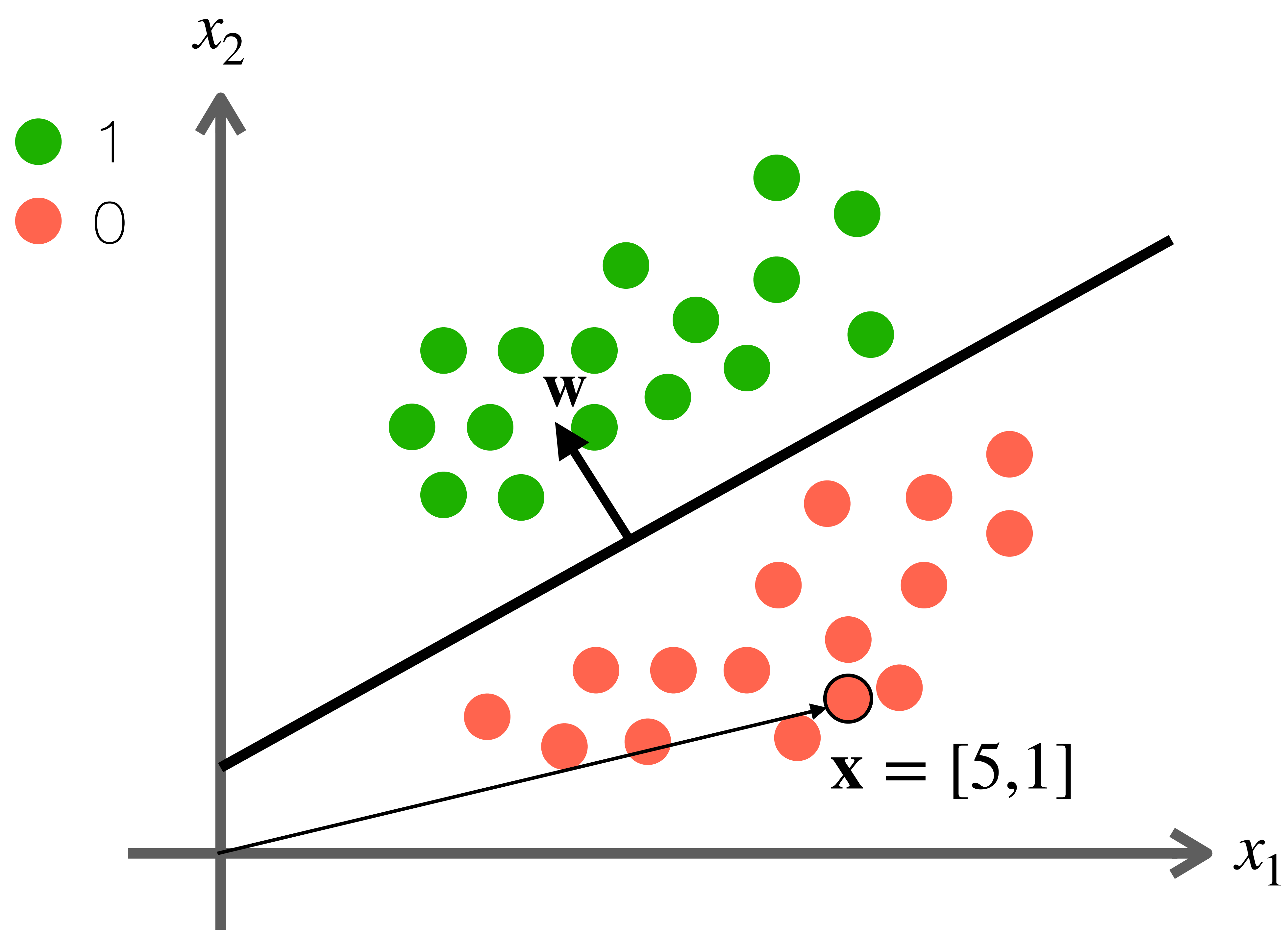


Truque do Kernel

Mapear os dados para um espaço de maior dimensão!

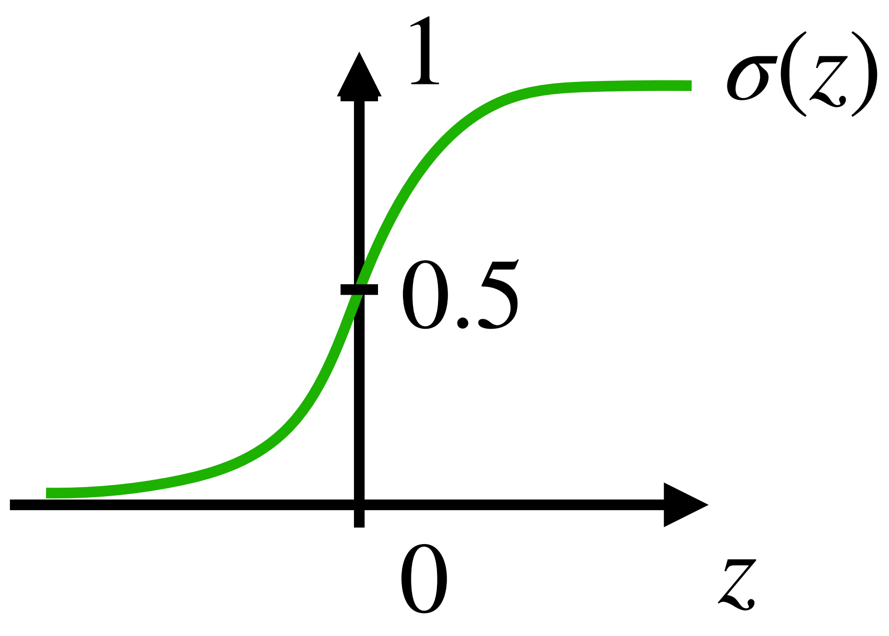
Regressão logística

A **regressão logística** pode ser vista como uma versão do Perceptron onde a função de ativação é a função logística $\boldsymbol{\sigma}$ ao invés da função sinal sgn.



$$h(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$

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



Usamos $\boldsymbol{\sigma}$ pois ela nos dá valores entre 0 e 1, e assim a saído do modelo é uma probabilidade.

$$\mathbf{w} = [-2,1] \quad b = 0$$

$$h(\mathbf{x}) = \sigma(-2 \cdot 5 + 1 \cdot 1 + 0)$$

$$h(\mathbf{x}) = \sigma(-9)$$

$$h(\mathbf{x}) \approx 0$$

Treinando a regressão logística

A regressão logística $h(\mathbf{x})$ modela a probabilidade de um exemplo $\mathbf{x}$ ser da classe $y = 1$:

$$P(y = 1 \,|\, \mathbf{x}) = h(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{wx}+b}}$$

Dado um conjunto de dados $D = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)\}$, queremos $h(\mathbf{x_i}') \approx y_i'$

1. A probabilidade de um único exemplo $\mathbf{x_i}$:

$$P(y_i = 1 \,|\, \mathbf{x_i}) = h(\mathbf{x_i})$$

$$P(y_i = 0 \,|\, \mathbf{x_i}) = 1 - h(\mathbf{x_i})$$

2. Podemos compactar essas duas probabilidades em uma:

$$P(y_i \,|\, \mathbf{x_i}) = h(\mathbf{x}_i)^{y_i} \cdot (1 - h(\mathbf{x}_i))^{(1-y_i)}$$

3. Queremos maximizar $P(y_i \,|\, \mathbf{x_i})$ para todo $(\mathbf{x_i}, \mathbf{y_i}) \in D$:

$$L(h) = \prod_{i}^{m} h(\mathbf{x}_i)^{y_i} \cdot (1 - h(\mathbf{x}_i))^{(1-y_i)}$$

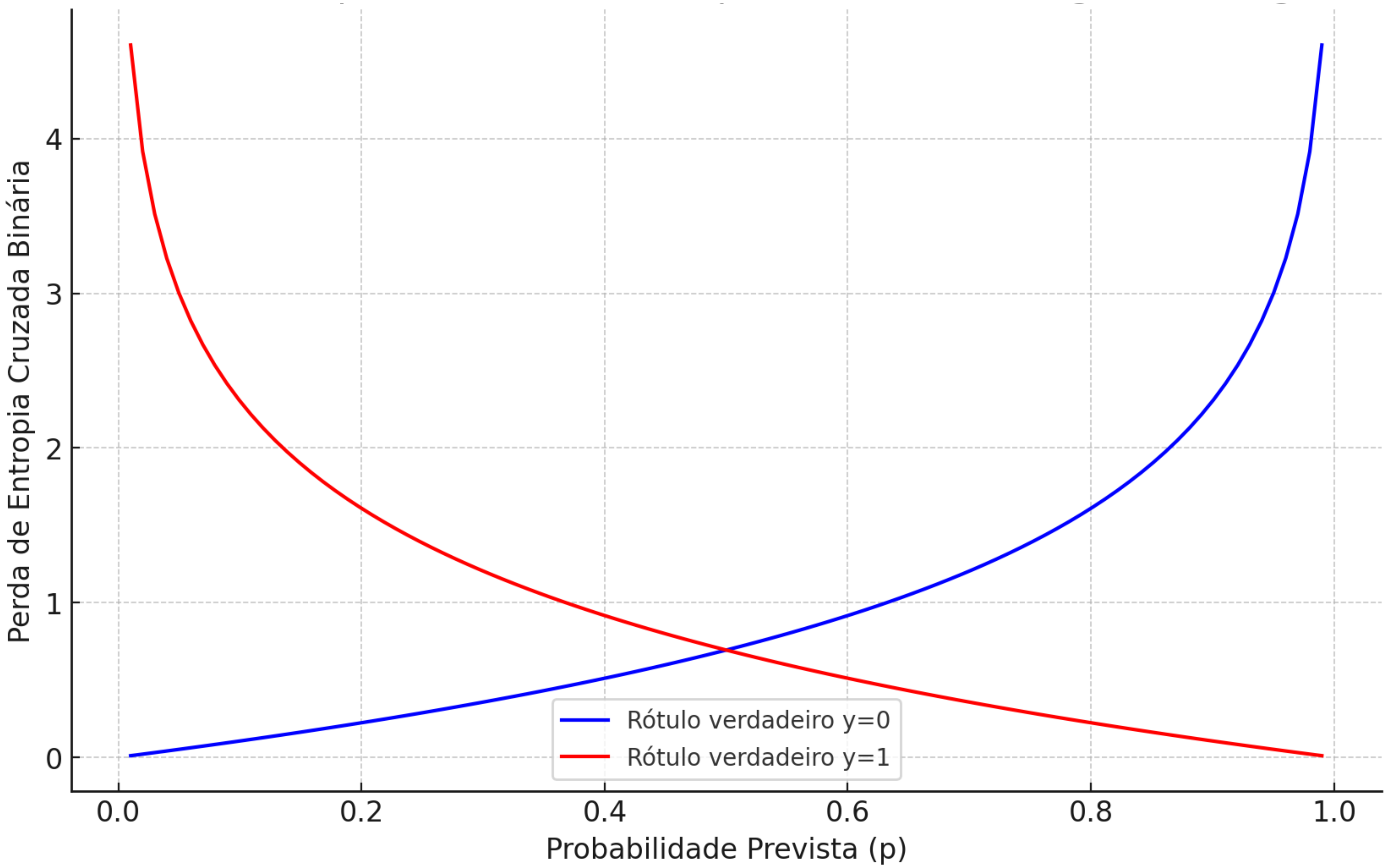
4. Aplicando o log e negando para transformar em erro:

$$L(h) = \underbrace{-\frac{1}{m}\sum_{i}^{m} y_i \, log(h(\mathbf{x_i})) + (1 - y_i)\, log(1 - h(\mathbf{x_i}))}_{\textbf{Entropia binária cruzada}}$$

UFV

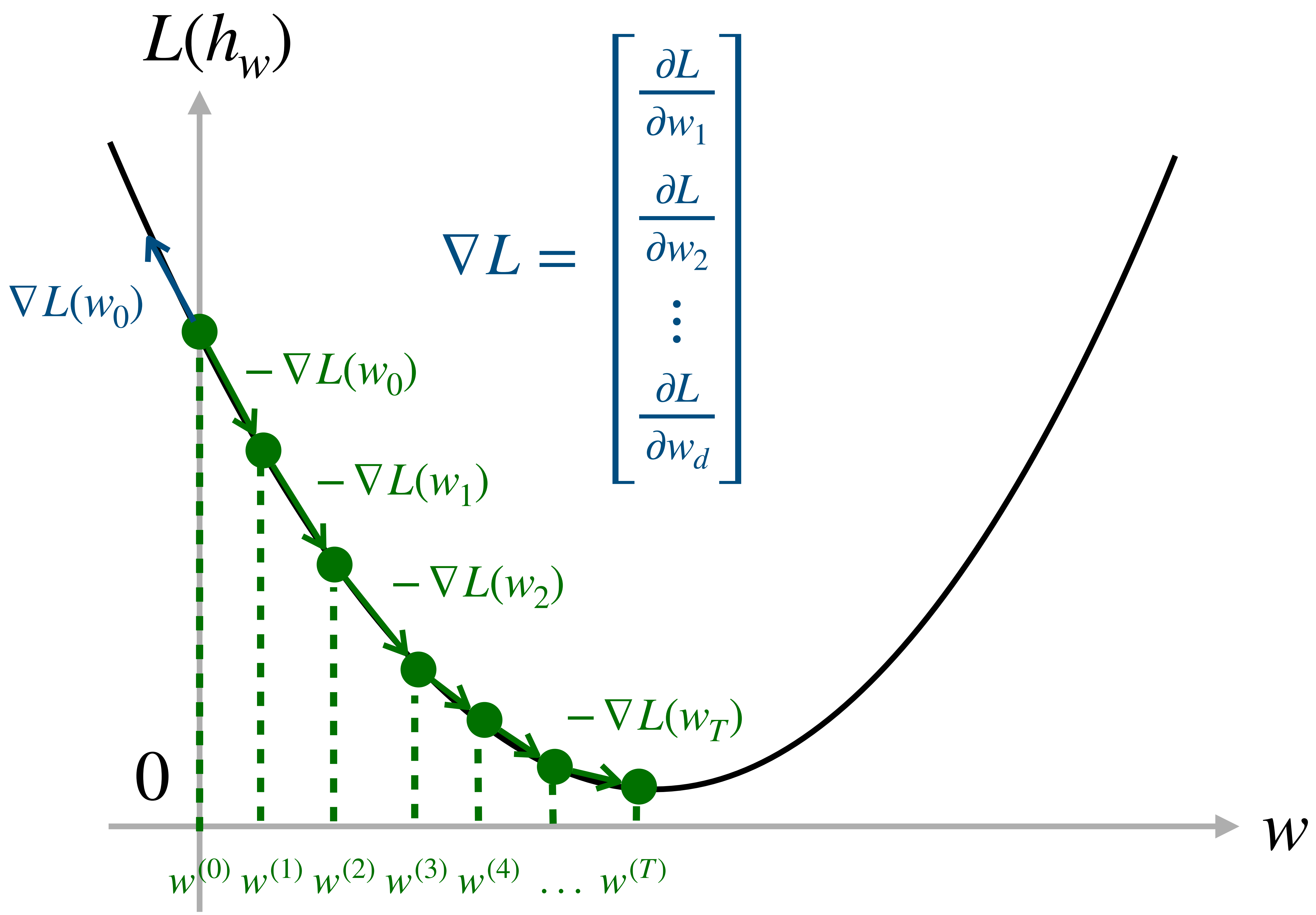
Entropia binária cruzada

A função de perda entropia binária cruzada é uma função convexa se $h(x) = \sigma(\mathbf{xw} + \mathbf{b})$



$$L(h) = -\sum_{i}^{m} y_i \, log(h(\mathbf{x_i})) + (1 - y_i) \, log(1 - h(\mathbf{x_i}))$$

Gradiente descendente



$$w_t \leftarrow w_{t-1} - \alpha \nabla L(w_{t-1})$$

Dado um valor inicial w, atualizamos iterativamente o valor de w na direção de descida mais íngrime de L a partir do ponto $(w, L(w))$

α é um híper-parâmetro chamado de **taxa de aprendizado** (*learning rate*), responsável por controlar o comprimento do vetor gradiente.

UFV

Implementação do gradiente descendente

```
def gradient_descent(X, Y, epochs, alpha):
    # Init weights to zero
    w, b = 0, 0
    # Optimize weihts iteratively
    for t in range(epochs):
        # Predict X labels with w and b
        Y_hat = logistic_regression(X, w, b)
        # Compute gradients
        dw, db = gradients(X, Y, Y_hat)
        # Update weights
        w = w - alpha * dw
        b = b - alpha * db
    return w, b
```

Gradiente descendente para regressão logística

```
def gradient_descent(X, Y, epochs, alpha):
# Init weights to zero
w, b = 0, 0
# Optimize weihts iteratively
for t in range(epochs):
    # Predict X labels with w and b
    Y_hat = logistic_regression(X, w, b)
    # Compute gradients
    dw, db = gradients(X, Y, Y_hat)
    # Update weights
    w = w - alpha * dw
    b = b - alpha * db
return w, b
```

Regressão Logística

$$z = \mathbf{wx} + b$$

$$\hat{y} = h(\mathbf{x}) = \frac{1}{1 + e^{-z}}$$

Função de Perda

$$L(h) = -\frac{1}{m}\sum_{i=1}^{m}(y_i \ log \ \hat{y}_i + (1 - y_i) \ log \ (1 - \hat{y}_i))$$

Gradientes

$$\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_i$$

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

Gradiente descendente para regressão logística

Grafos computacionais nos ajudam a calcular o gradiente de $L(h_{w,b}(\mathbf{x}))$ com relação aos pesos $\mathbf{w}$ e b da regressão logística

Regressão Logística

$$z = \mathbf{wx} + b\ ,\ \ \hat{y} = h(\mathbf{x}) = \frac{1}{1+e^{-z}}$$

Função de perda para um exemplo

$$L(\hat{y}, y) = y_i \ log\ \hat{y}_i + (1-y_i)\ log\ (1-\hat{y}_i)$$

x_1, w_1, x_2, w_2, b

$$z = w_1x_1 + w_2x_2 + b \longrightarrow \hat{y} = \sigma(z) \longrightarrow L(\hat{y}, y)$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}$$

$$= (-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}) \cdot \hat{y}(1-\hat{y})$$

$$= \hat{y} - y$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$= (\hat{y} - y)\ \cdot\ x_1$$

Regra da Cadeia!

UFV

Fronteira de decisão

Para encontrar o rótulo $\hat{y}$ com a hipótese, usamos um limiar:

$$\hat{y} = \begin{cases} 1, \text{ se } h(\mathbf{x}) \geq 0.5 \\ 0, \text{ se } h(\mathbf{x}) < 0.5 \end{cases}$$

Repare que o liminar acima é equivalente ao seguinte limiar:

$$\hat{y} = \begin{cases} 1, \text{ se } \mathbf{wx} + b \geq 0 \\ 0, \text{ se } \mathbf{wx} + b < 0 \end{cases}$$

Pois:

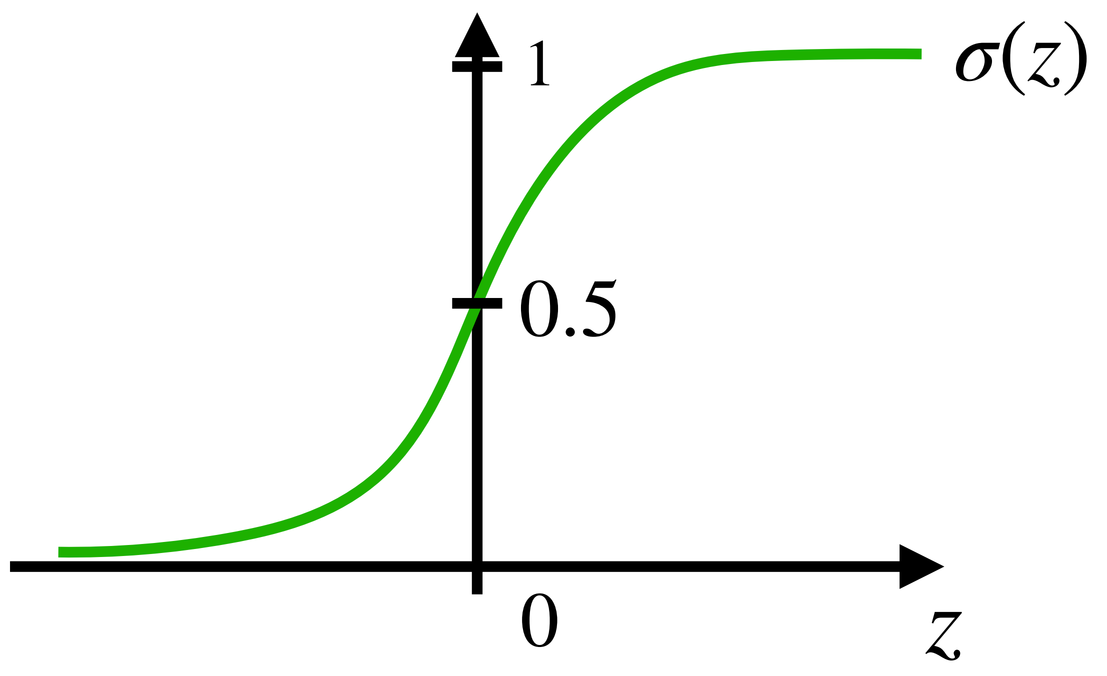
$h(\mathbf{x}) \geq 0.5$ quando $z \geq 0$

$h(\mathbf{x}) < 0.5$ quando $z < 0$

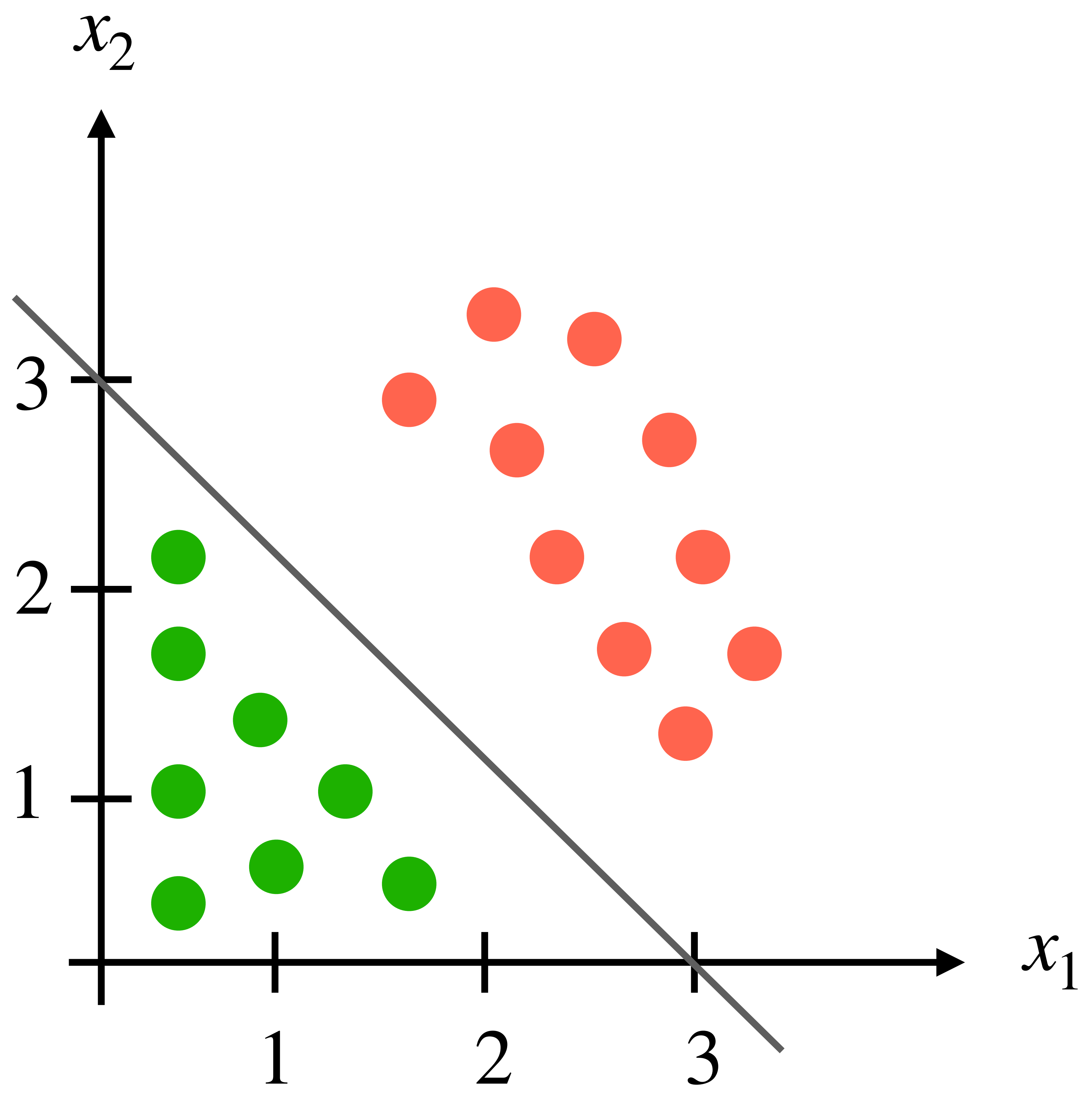
Regressão Logística

$$z = \mathbf{wx} + b$$

$$\hat{y} = h(\mathbf{x}) = \frac{1}{1 + e^{-z}}$$



Fronteira de decisão



Regressão Logística

$z = \mathbf{wx} + b$

$\hat{y} = h(\mathbf{x}) = \dfrac{1}{1 + e^{-z}}$

Considere a seguinde hipótese após o treinamento:

$w_1 = 1,\ w_2 = 1,\ b = -3$

$$\hat{y} = \begin{cases} 1, \text{ se } x_1 + x_2 - 3 \geq 0 \\ 0, \text{ se } x_1 + x_2 - 3 < 0 \end{cases}$$

A reta $x_1 + x_2 = 3$ é chamada de **fronteira de decisão**.

UFV

Próxima aula

A28: Aprendizado supervisionado V

Redes neurais artificiais