CS473 - Pattern Recognition Tutorial 4

TAs:
- Emmanouil Sylligardos, sylligardos@csd.uoc.gr
- Despina - Ekaterini Argiropoulos, despargy@csd.uoc.gr

Instructor:
- Prof. Panos Trahanias, trahania@csd.uoc.gr

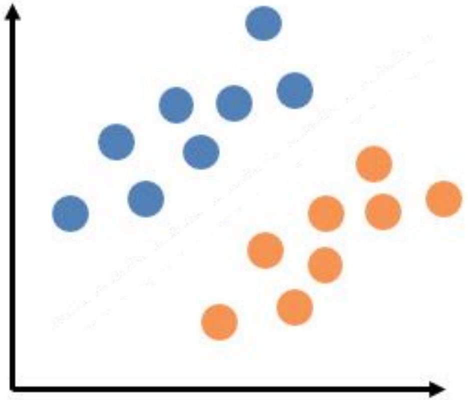
Spring Semester 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
UNIVERSITY OF CRETE

Table of contents

The dataset & the linear classifier

Question



Question

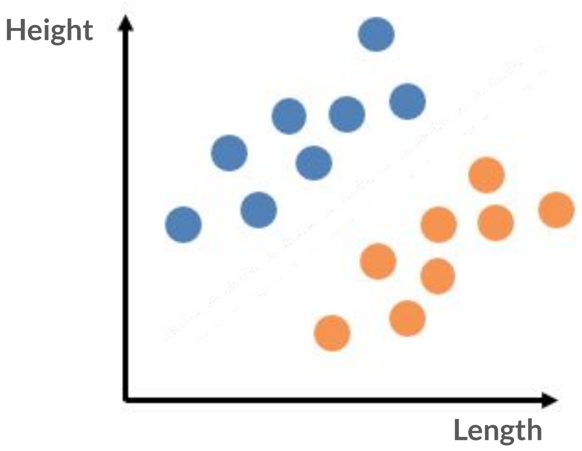
2 **breeds** of dogs:

- Blue:

  tall legs & short body

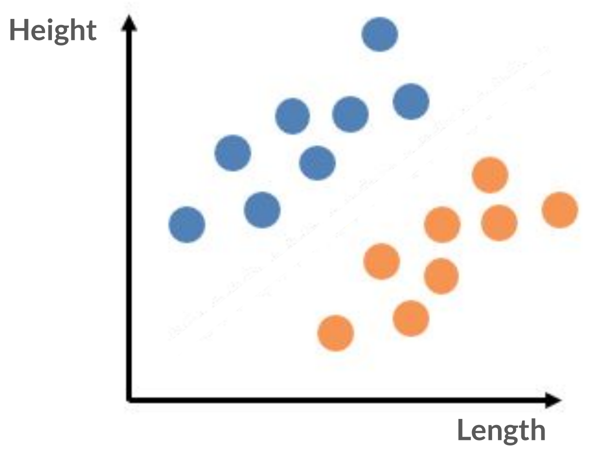
- Orange:

  short legs & long body



Question

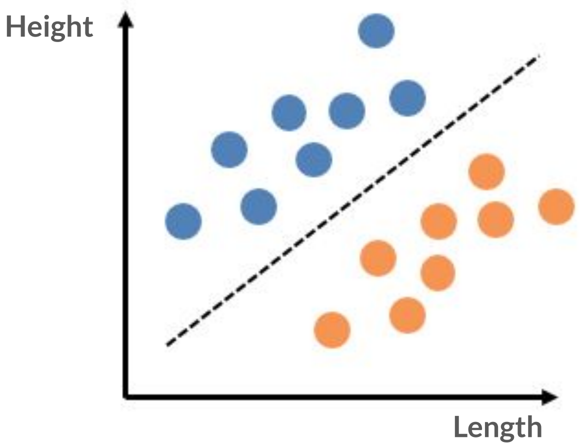
Can we separate the two classes?



Definition

The dataset is **linearly separable**

Linearly separable =:
a dataset is said to be linearly separable if it is possible to draw a straight line (or a hyperplane in higher dimensions) that separates the different classes in the dataset

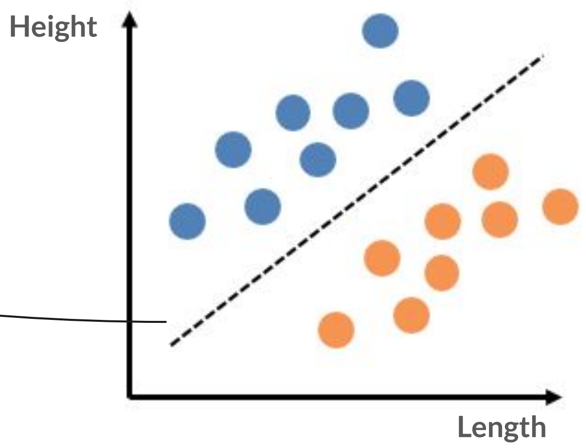


Definition

The dataset is **linearly separable**

Linearly separable =:
a dataset is said to be linearly separable if it is possible to draw a straight line (or a hyperplane in higher dimensions) that separates the different classes in the dataset

$y = m*x + b$



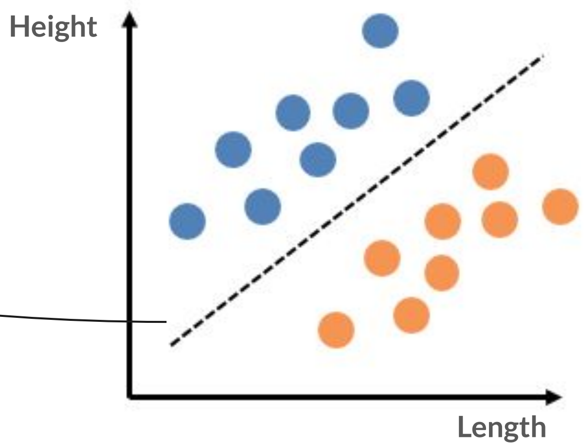
Problem

We need a formalized way to find:

$y = m^*x + b \Longrightarrow y = x + 1$

Can we name some features of this line wrt the data?

$y = m^*x + b$



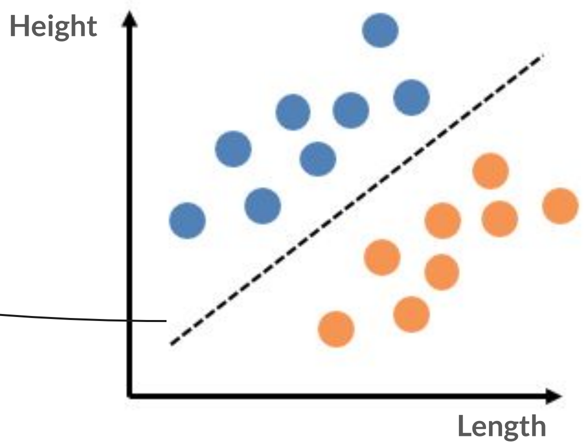
Problem

Can we name some features of this line wrt the data?

All points of class **'Blue'** are **on top of** the line, while all points of class **'Orange'** are **under** the line.

Let's formalize!

y = m*x + b



Solution

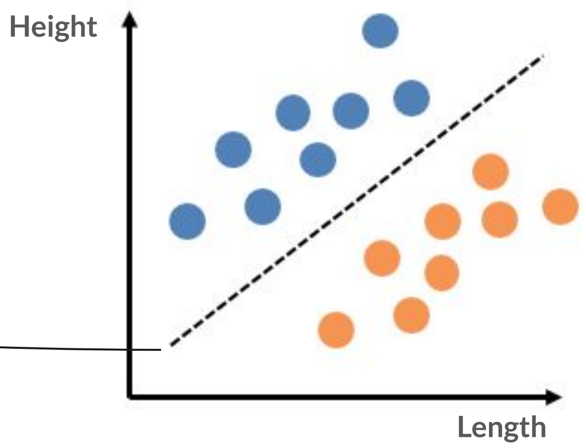
$Ax + By + C = 0 \Leftrightarrow$

$-By = Ax + C \Leftrightarrow$

$y = Ax + C / -B \Leftrightarrow$

$y = (-A/B) * x + (-C/B) \Leftrightarrow$

$y = mx + b$



Solution

Ax + By + C = 0 ⇔

-By = Ax + C ⇔

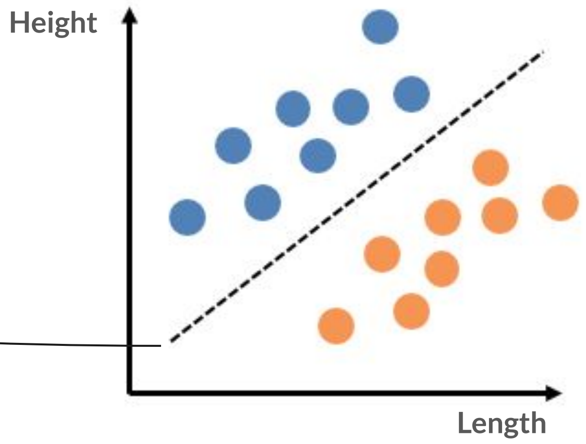
y = Ax + C / -B ⇔

y = (-A/B) * x + (-C/B) ⇔

y = mx + b

Absolute distance

$$d = \frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$



Solution

$Ax + By + C = 0 \Leftrightarrow$

$-By = Ax + C \Leftrightarrow$

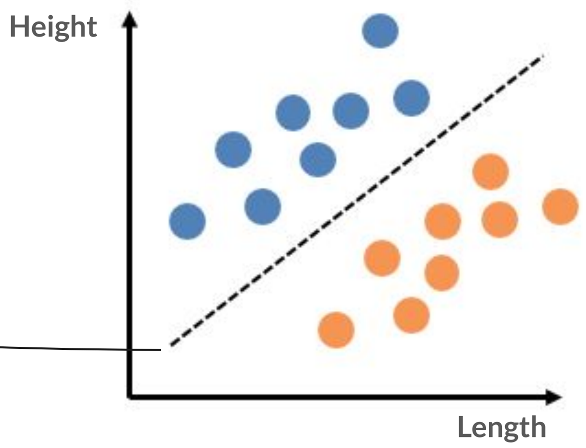
$y = Ax + C / -B \Leftrightarrow$

$y = (-A/B) * x + (-C/B) \Leftrightarrow$

$y = mx + b$

Signed distance

$$d = \frac{Ax_0 + By_0 + C}{\sqrt{A^2 + B^2}}$$



Model

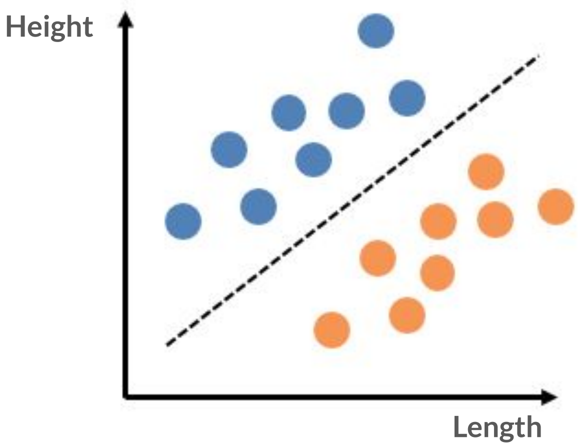
$Ax + By + c = 0$

$g(x) = Ax + By + c$

$g(x) = w_1x_1 + w_2x_2 + \dots + w_nx_n + w_0$ *(n dimensions)*

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

$g(x) = \mathbf{a}^T * \mathbf{y}$ *(bias trick)*



Model

g(x) is **proportional** to the signed distance!

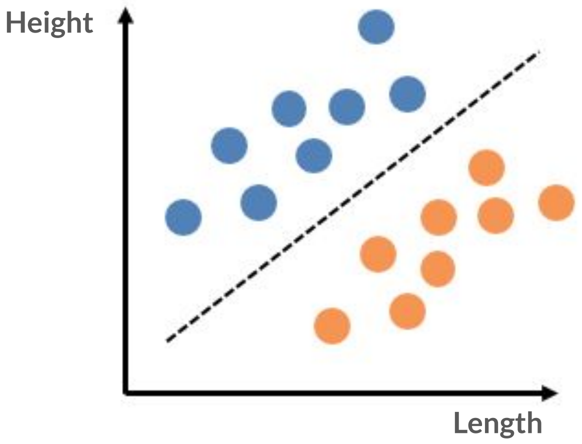
$Ax + By + c = 0$

$g(x) = Ax + By + c$

$g(x) = w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0w$ *(n dimensions)*

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

$g(x) = \mathbf{a}^T * \mathbf{y}$ *(bias trick)*



Extras

https://github.com/boniolp/MSAD

https://pytorch.org/docs/stable/generated/torch.nn.functional.linear.html

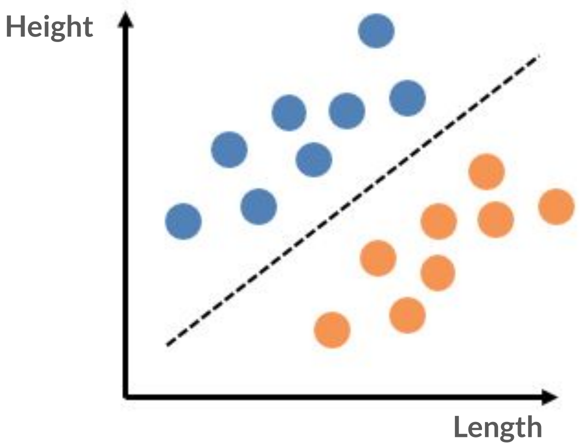
$Ax + By + c = 0$

$g(x) = Ax + By + c$

$g(x) = w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0w$ *(n dimensions)*

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

$g(x) = \mathbf{a}^T * \mathbf{y}$ *(bias trick)*



The training algorithm

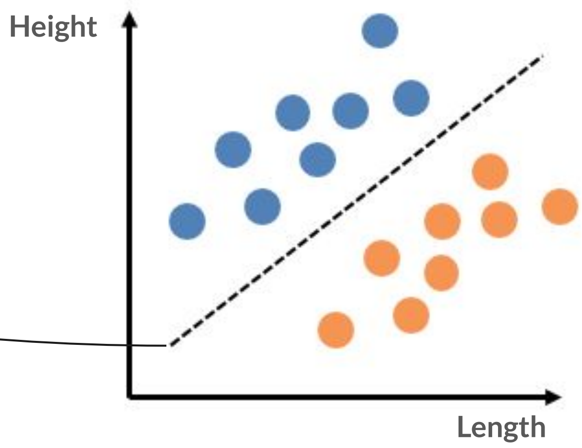
The question remains

We need a formalized way to:

$$y = m^*x + b \Longrightarrow y = x + 1$$

Hint: it's an optimization problem

$y = x + 1$



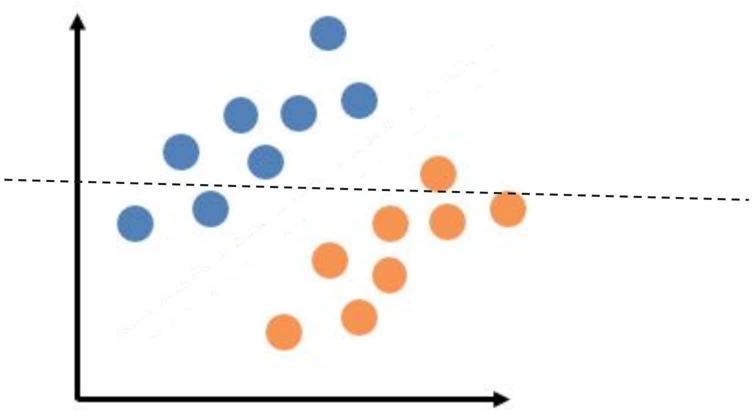
Criterion

We need something to minimize

Try #1:

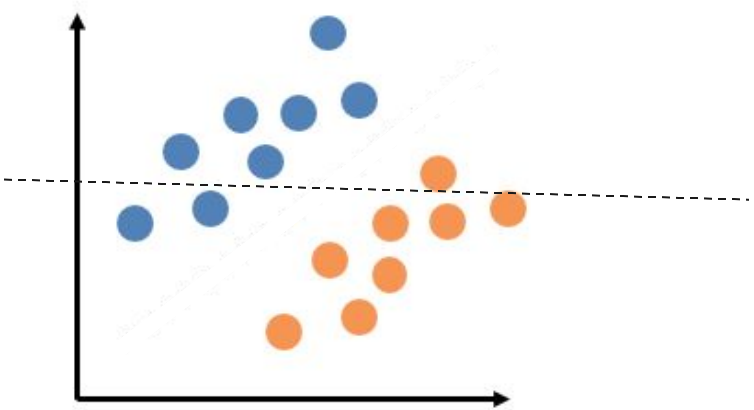
➔ ***The number of misclassified samples***

Is it a good option?



Criterion

- ➔ Good: It makes sense to minimize the number of misclassified samples
- ➔ Bad: The function is non-continuous thus non-differentiable



Criterion

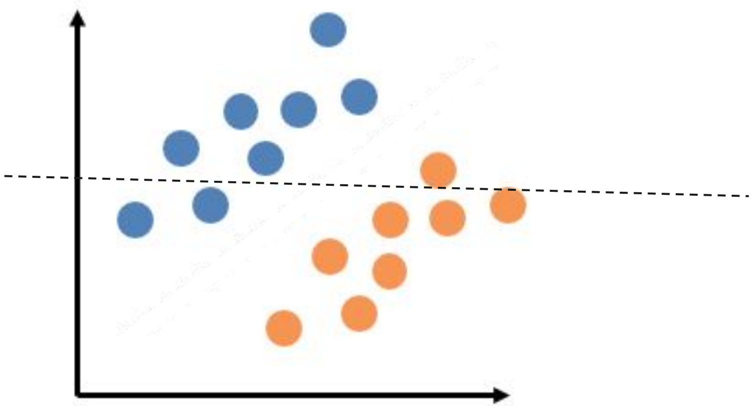
$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} \left(-\mathbf{a}^t \mathbf{y} \right)$$

We need something to minimize

Try #2:

➔ ***The distance of misclassified samples from the classifier***

Is it a good option?

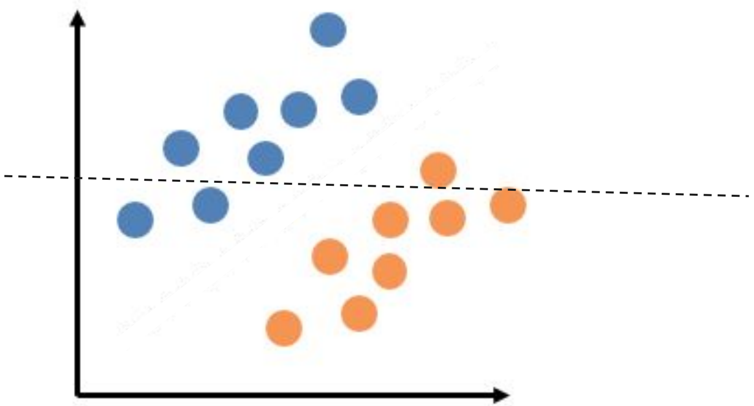


Criterion

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} \left(-\mathbf{a}^t \mathbf{y}\right)$$

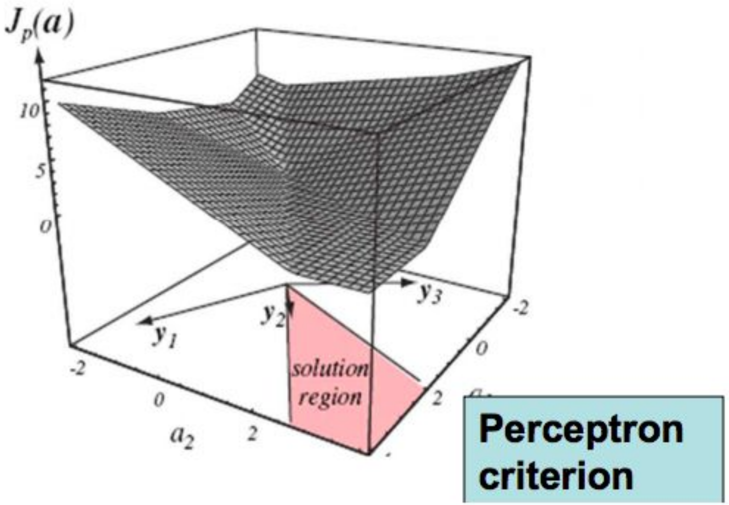
➔ Good: It makes sense to minimize the distance of misclassified samples

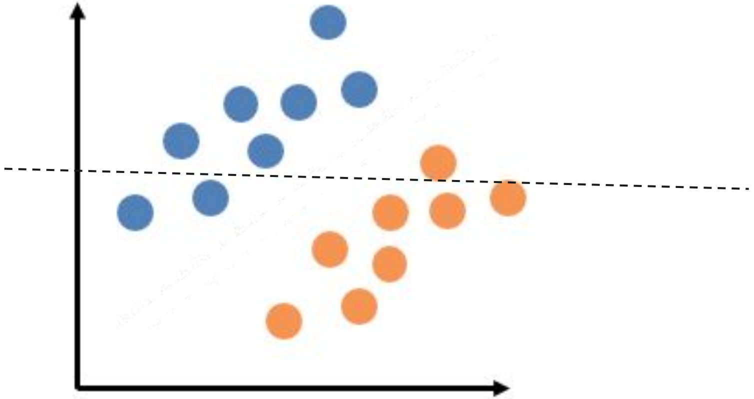
➔ Good: The function is continuous thus differentiable



Criterion (visualization)

$$J_p(\mathbf{a}) = \sum_{\mathbf{y}\in\mathcal{Y}} \left(-\mathbf{a}^t\mathbf{y}\right)$$





Criterion

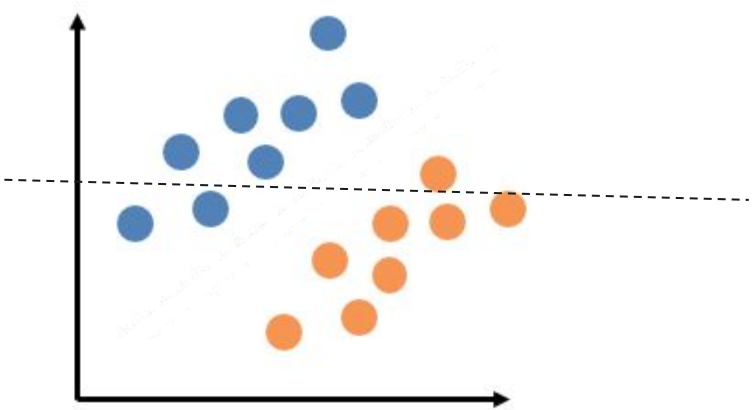
$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in \mathcal{Y}} \left(-\mathbf{a}^t \mathbf{y} \right)$$

What's the derivative:

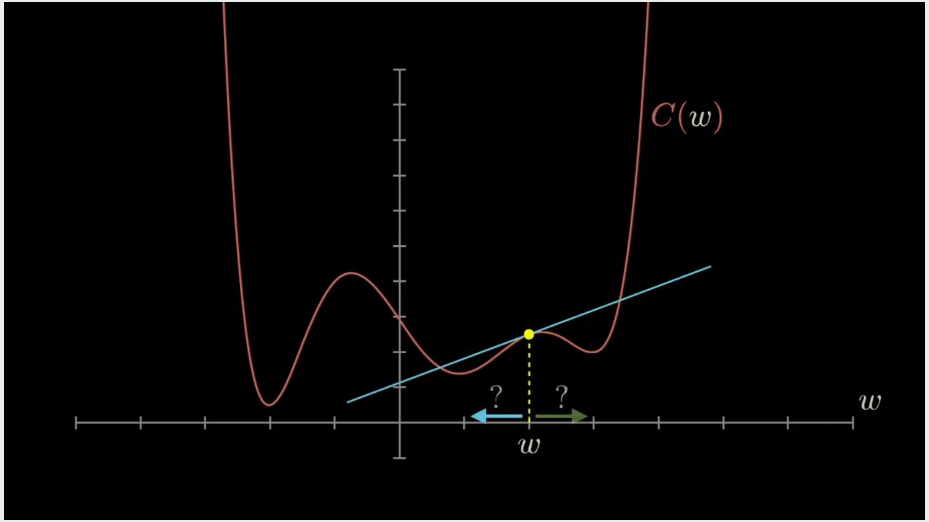
➔ The derivative is -y (look notes)

Question:

Why do we need the derivative :/



Gradient descent

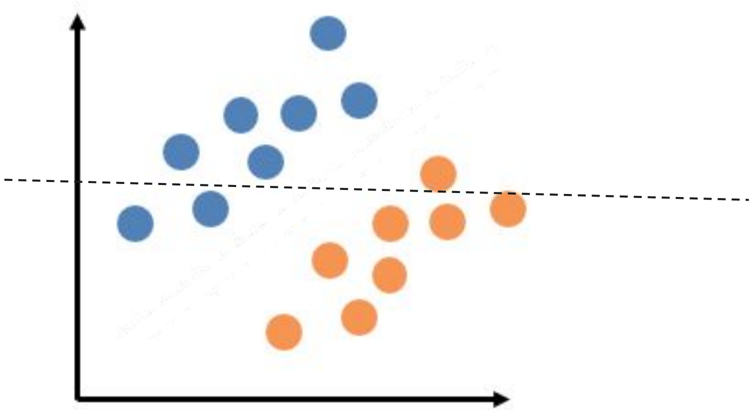


What's the derivative:

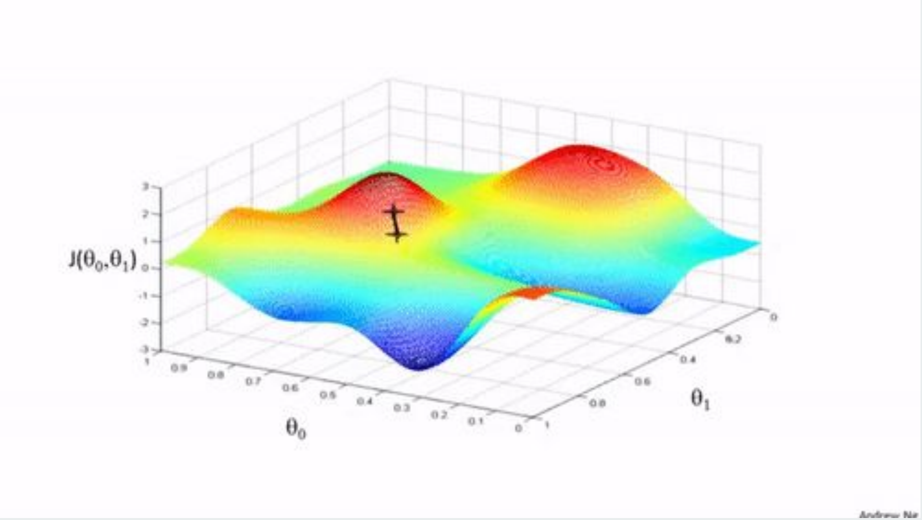
➔ The derivative is -y (look notes)

Question:

Why do we need the derivative :/



Gradient descent

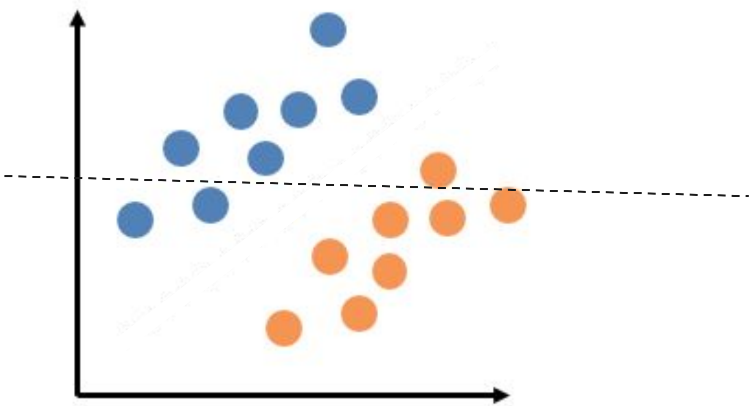


What's the derivative:

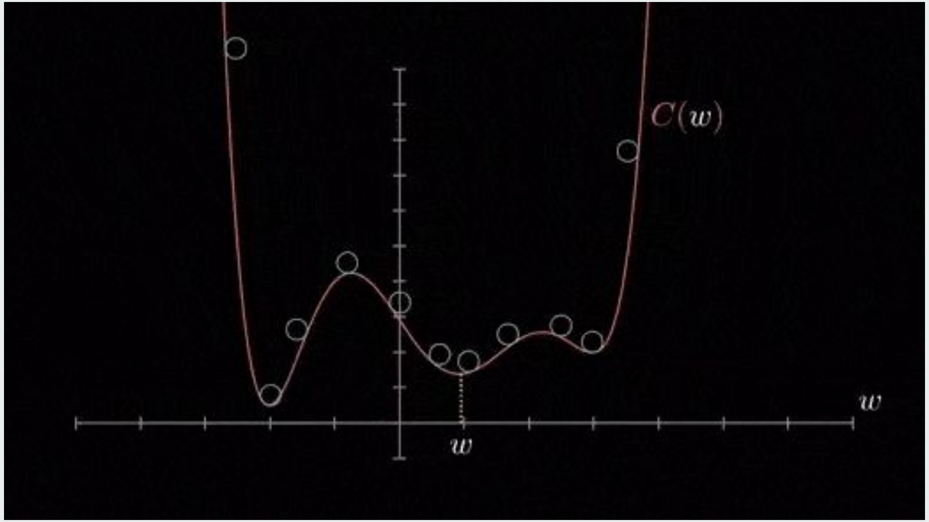
➔ The derivative is -y (look notes)

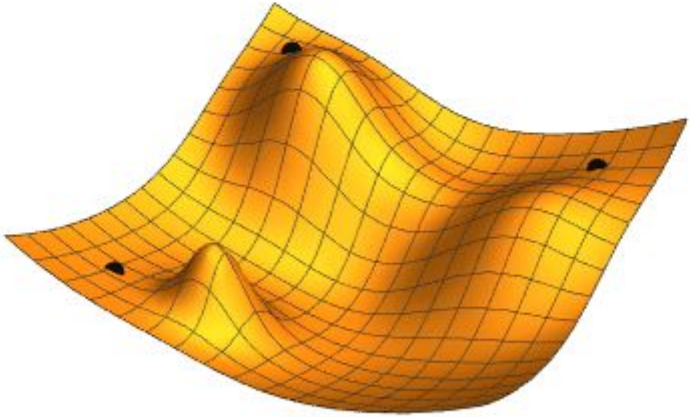
Question:

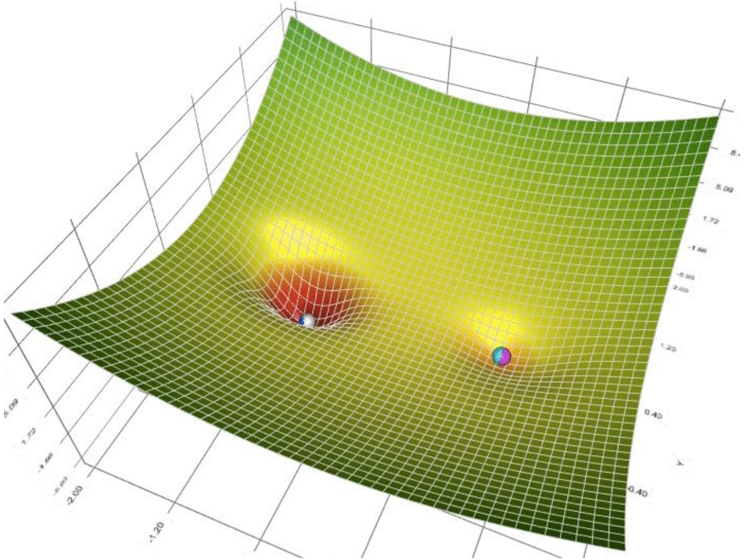
Why do we need the derivative :/



Gradient descent







What you will implement

Fixed Increment Single-Sample Perceptron

Algorithm 4 (Fixed-increment single-sample Perceptron)

begin initialize $\mathbf{a}, k = 0$

  do $k \leftarrow (k+1) \bmod n$

    if $\mathbf{y}_k$ is misclassified by $\mathbf{a}$ **then** $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{y}_k$

  until all patterns properly classified

 return $\mathbf{a}$

end

Fixed/Variable Increment Batch Perceptron

Algorithm 3 (Batch Perceptron)

begin initialize $\mathbf{a}, \eta(\cdot)$, criterion $\theta, k = 0$

  do $k \leftarrow k + 1$

    $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$

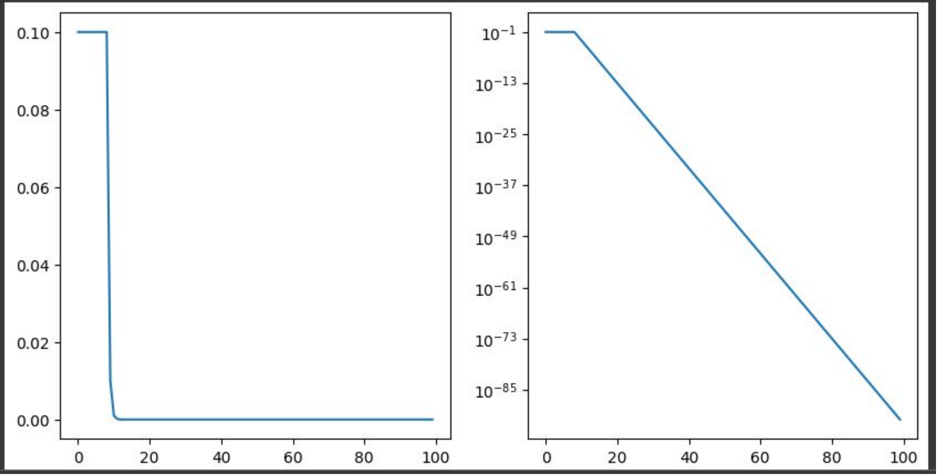
  until $\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} < \theta$

 return $\mathbf{a}$

end

Scheduler

```
class DummyScheduler:
    def __init__(self, lr=0.1, factor=0.1, patience=10):
        self.lr = lr
        self.factor = factor
        self.patience = patience
        self.counter = 0

    def get_next_lr(self):
      self.counter += 1

      if self.counter >= self.patience:
        self.lr *= self.factor

      return self.lr

dummy_scheduler = DummyScheduler()
lr_values = [dummy_scheduler.get_next_lr() for i in range(100)]

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(lr_values)

axs[1].plot(lr_values)
axs[1].set_yscale("log")

plt.show()
```



TQDM lib

```
from tqdm.auto import tqdm
from time import sleep

loop = tqdm(
    range(10000),
    desc="Example",
)

animals = ["cat", "cow", "dog"]
animals_counter = 0
for i in loop:
  sleep(0.01)
  if i % 100:
    loop.set_postfix(animal=animals[animals_counter % len(animals)])
    animals_counter += 1
```

```
Example: 2% ▌ 240/10000 [00:02<01:51, 87.22it/s, animal=cow]
```

Any questions?

hy473-list@csd.uoc.gr
sylligardos@csd.uoc.gr
despargy@csd.uoc.gr