

# **Support Vector Machines**

Machine Learning and Deep Learning 2022

---

Aniello Panariello, Lorenzo Bonicelli, Matteo Boschini

October 14th, 2022

University of Modena and Reggio Emilia

# Today's agenda

**SVM basics**

**PEGASOS**

**Exercise**

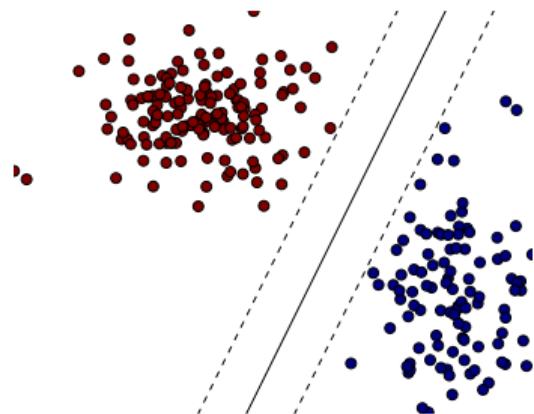
## SVM basics

---

# Support Vector Machines (SVM)

Very famous supervised learning algorithm:

- performs **binary** classification (can be adapted for multiclass problems)
- is **linear** (in its original formulation)
- main feature: chooses the decision boundary that maximizes the margin between positive and negative examples



How do you find such line?

## SVM: problem setting

- we have a set of examples  $\{\vec{x}_i\}_{i=1}^N$ , with label  $\{y_i\}_{i=1}^N$ 
  - $\vec{x}_i$  is a feature vector
  - $y_i \in \{-1, 1\} \quad \forall i = 1, \dots, N$
- the line that maximizes the margin (hyperplane) is identified by
  - the vector  $\vec{w}$  orthogonal to it
  - the intercept  $b$

## SVM: hard margin optimization

It can be shown that solving this constrained optimization problem maximizes the margin:

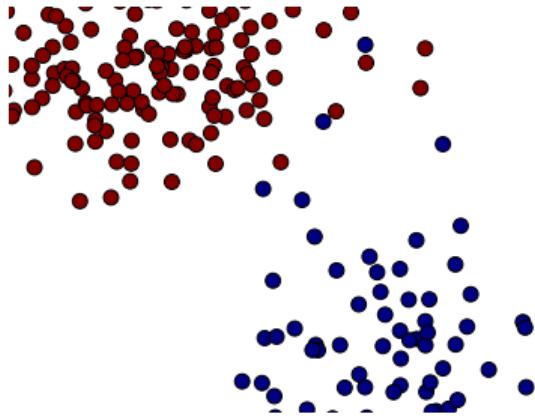
$$\begin{aligned} \min_{\vec{w}, b} \quad & \frac{1}{2} \|\vec{w}\|^2 \\ \text{s.t.} \quad & y_i(\langle \vec{w}, \vec{x}_i \rangle + b) > 1 \quad \forall i = 1, \dots, N \end{aligned}$$

After estimating  $\vec{w}$  and  $b$ , the decision rule for an unknown example  $\vec{u}$  is simply:

$$f(\vec{u}) = \text{sign}(\langle \vec{w}, \vec{u} \rangle + b)$$

## SVM: non separable data

What if you cannot find a linear decision boundary?



Two solutions:

- introduce slack variables  $\xi_i$  (soft margin solution)
- kernel trick

## SVM: soft margin optimization

Introduce a slack variable  $\xi_i$  for every training example

$$\begin{aligned} \min_{\vec{w}, b} \quad & \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(\langle \vec{w}, \vec{x}_i \rangle + b) > 1 - \xi_i \quad \forall i = 1, \dots, N \\ & \xi_i > 0 \quad \forall i = 1, \dots, N \end{aligned}$$

where  $C$  is a tradeoff between the margin maximization and the slack variable minimization.

The decision rule for an unknown example  $\vec{u}$  does not change:

$$f(\vec{u}) = \text{sign}(\langle \vec{w}, \vec{u} \rangle + b)$$

## SVM: kernels

Math guys find out that if you employ Lagrangian multipliers to solve the optimization problem, you need to find the extrema of the following function ( $\{\alpha\}_{i=1}^N$  are the lagrangian multipliers of each training example):

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \vec{x}_i, \vec{x}_j \rangle$$

(1)



## SVM: kernels (2)

You can replace that dot product with a kernel  $K(\vec{x}_i, \vec{x}_j) = \langle \phi(\vec{x}_i), \phi(\vec{x}_j) \rangle$ . Here,  $\phi$  is a non linear mapping to a new space, but you don't really need to compute it!

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j) \quad (2)$$

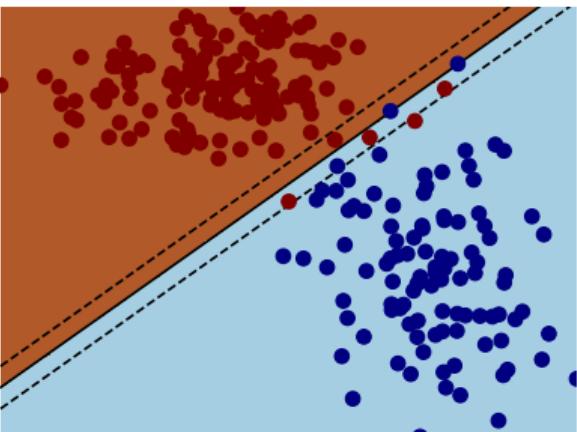
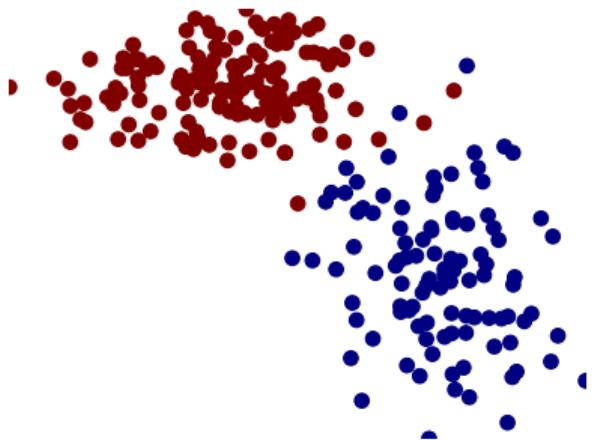
Choose a kernel:

- linear:  $K(\vec{x}_i, \vec{x}_j) = \langle \vec{x}_i, \vec{x}_j \rangle$
- polynomial:  $K(\vec{x}_i, \vec{x}_j) = (\langle \vec{x}_i, \vec{x}_j \rangle + c)^d$
- gaussian:  $K(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right)$

This way, you draw the hyperplane in a transformed space, leading to non linear decision boundaries in the original space!

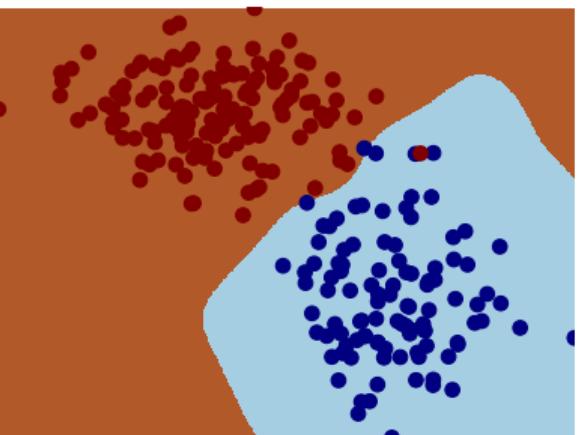
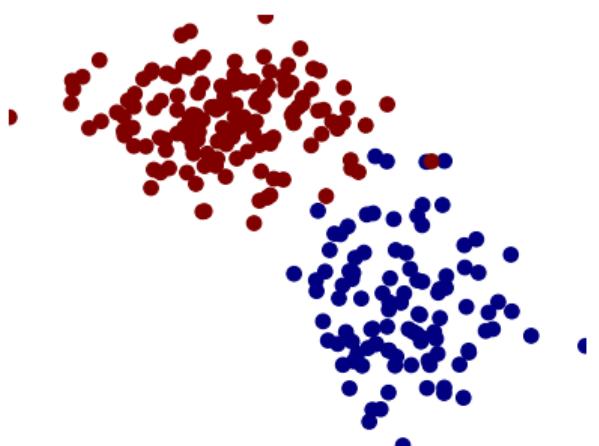
# SVM example

Using a linear kernel:



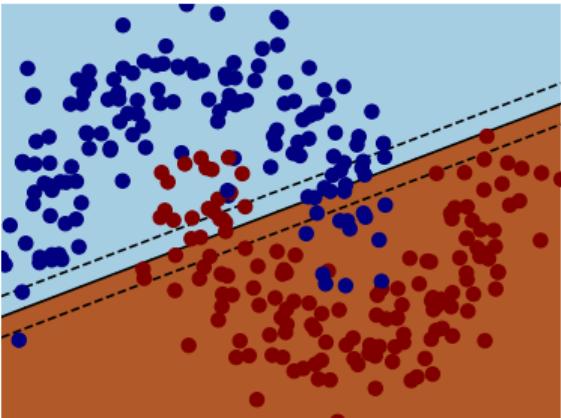
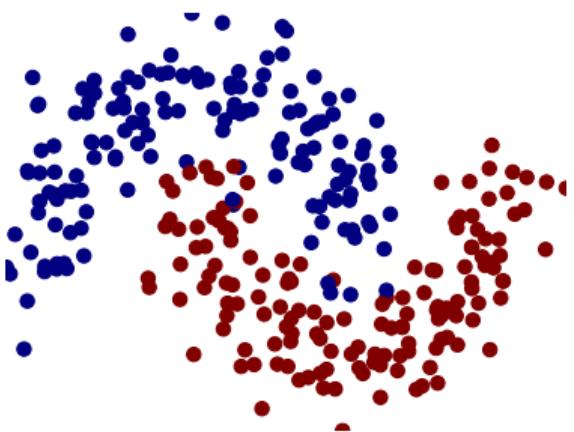
## SVM example

Using a rbf (gaussian) kernel:



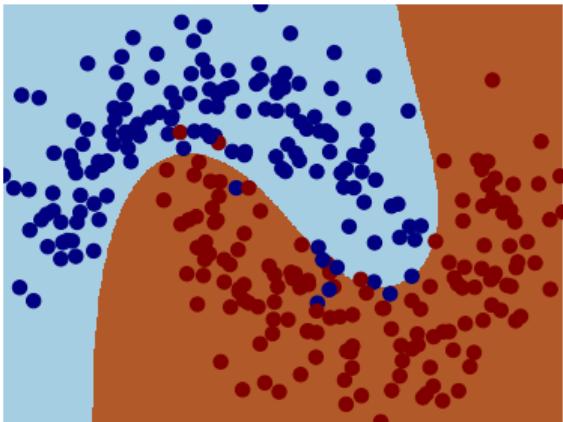
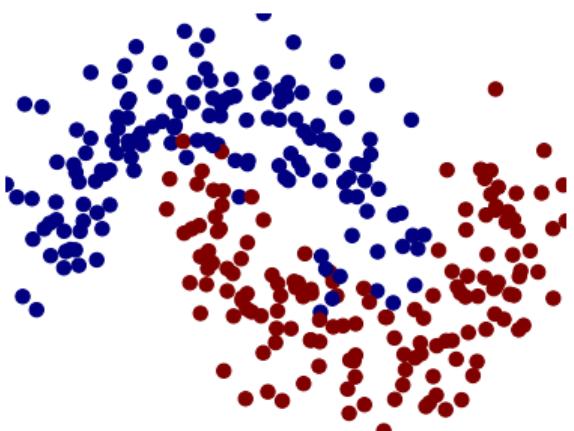
## SVM example (2)

Using a linear kernel:



## SVM example (2)

Using a rbf (gaussian) kernel:



# **PEGASOS**

---

Pegasos is a training algorithm for SVMs allowing to solve the optimization problem in its **primal form** by gradient descent.

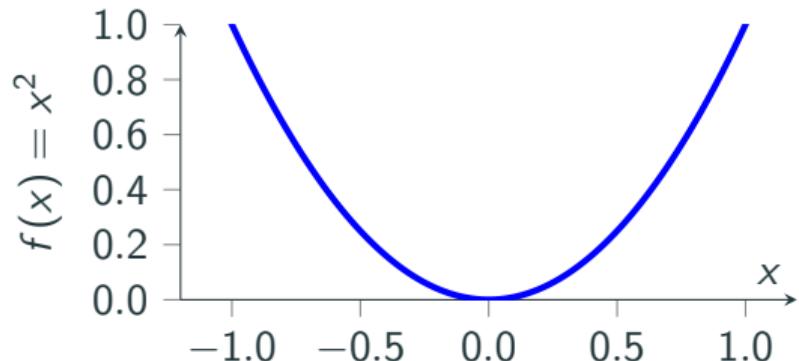
It starts from the soft margin formulation:

$$\begin{aligned} \min_{\vec{w}, b} \quad & \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(\langle \vec{w}, \vec{x}_i \rangle + b) > 1 - \xi_i \quad \forall i = 1, \dots, N \\ & \xi_i > 0 \end{aligned}$$

And rewrites it as:

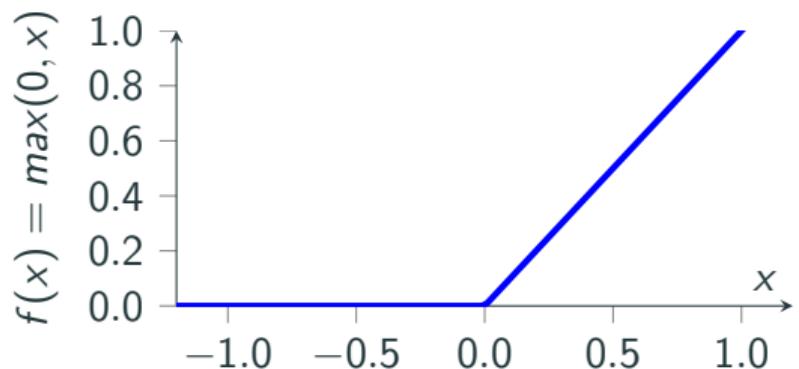
$$\min_{\vec{w}} L(\vec{w}) = \underbrace{\frac{\lambda}{2} \|\vec{w}\|^2}_{\text{regularization}} + \underbrace{\frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i \langle \vec{w}, \vec{x}_i \rangle)}_{\text{hinge loss}}$$

## Subgradient for non differentiable functions



$$f(x) = x^2$$

$$\nabla f(x) = 2x$$



$$f(x) = \max(0, x)$$

$$\nabla f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

# Optimizing with PEGASOS

We can now employ gradient descent to optimize

$$L(\vec{w}) = \frac{\lambda}{2} \|\vec{w}\|^2 + \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i \langle \vec{w}, \vec{x}_i \rangle)$$

## Algorithm 1 Gradient descent

---

```

1:  $\vec{w}^0 = \vec{0}$ 
2: for  $t = 0, \dots, n\_steps-1$  do
3:    $\vec{w}^{t+1} = \vec{w}^t - \eta^t \nabla L(\vec{w}^t)$ 
4: end for
```

---

Where:

- $\eta^t = \frac{1}{t\lambda}$  (step size)
- $\nabla L(\vec{w}^t) = \lambda \vec{w}^t - \frac{1}{N} \sum_{y_i \langle \vec{w}^t, \vec{x}_i \rangle < 1} y_i \vec{x}_i$

# PEGASOS algorithm

---

## Algorithm 2 PEGASOS

---

```
1:  $\vec{w}^0 = \vec{0}$ ,  $t = 0$ 
2: for epoch in  $1, \dots, n_{\text{epochs}}$  do
3:   for  $j = 1, \dots, N$  do
4:      $t = t + 1$ 
5:      $\eta^t = \frac{1}{t\lambda}$ 
6:     if  $y_j \langle \vec{w}_t \vec{x}_j \rangle < 1$  then
7:        $\vec{w}^{t+1} = (1 - \eta^t \lambda) \vec{w}^t + \eta^t y_j x_j$ 
8:     else
9:        $\vec{w}^{t+1} = (1 - \eta^t \lambda) \vec{w}^t$ 
10:    end if
11:  end for
12: end for
```

## **Exercise**

---

# SVM for people detection

We will discriminate between people and non people

people



non people



## (Optional) on-the-shelf SVM

Pipeline for people vs non people classification with **HOG features**:

- load training examples (images, HOG features, labels)
- load test examples (images, HOG features, labels)
- choose a kernel (linear or rbf) and a value for C and initialize a SVM model
  - use `sklearn.svm.SVC`
- fit a SVM model on training examples
  - use `SVC.fit()`
- use model for computing predictions on test examples
  - use `SVC.predict()`