

Algoritmos de Vectores de Soporte

🐮 Grupo 3 🐄

Contenidos de la Presentación

01

Perceptrón

Ejercicio 1.a a 1.c

02

SVM

Ejercicio 1.d y 2

03

Conclusiones

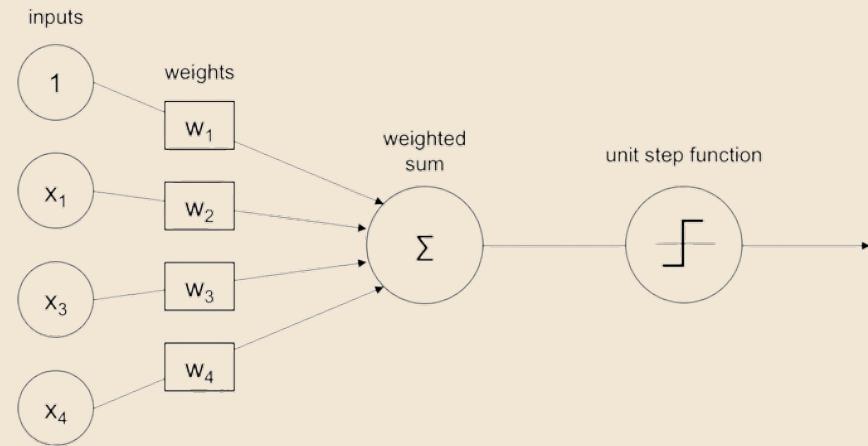
Puntos interesantes para
destacar

01

Perceptrón

Perceptrón Simple

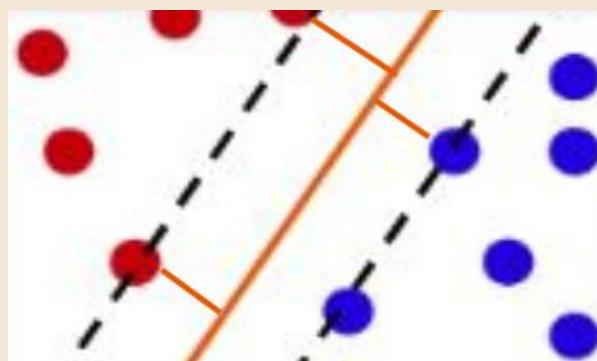
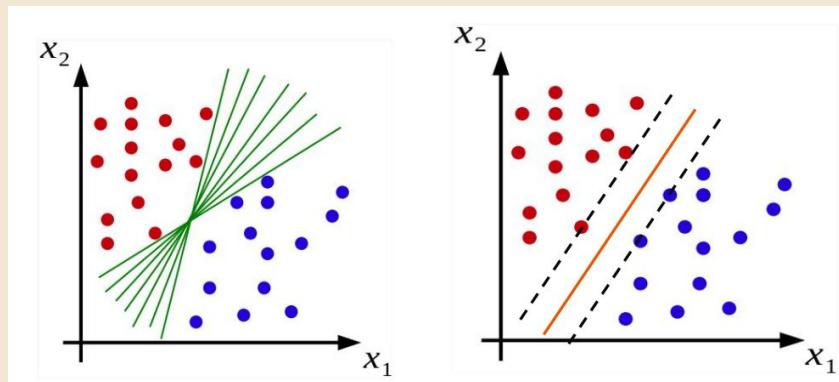
- Ajusta **iterativamente** los pesos de las características para minimizar errores y clasificar correctamente los datos de entrenamiento
- **Parámetros** → Pesos y *Learning rate*
- **Input** → elementos del conjunto
- **Función de activación** → escalón
- **Output** → 1 o -1



$$\sum_{i=1}^n w_i x_i + w_0 = 0$$

Maximizar margen para la recta

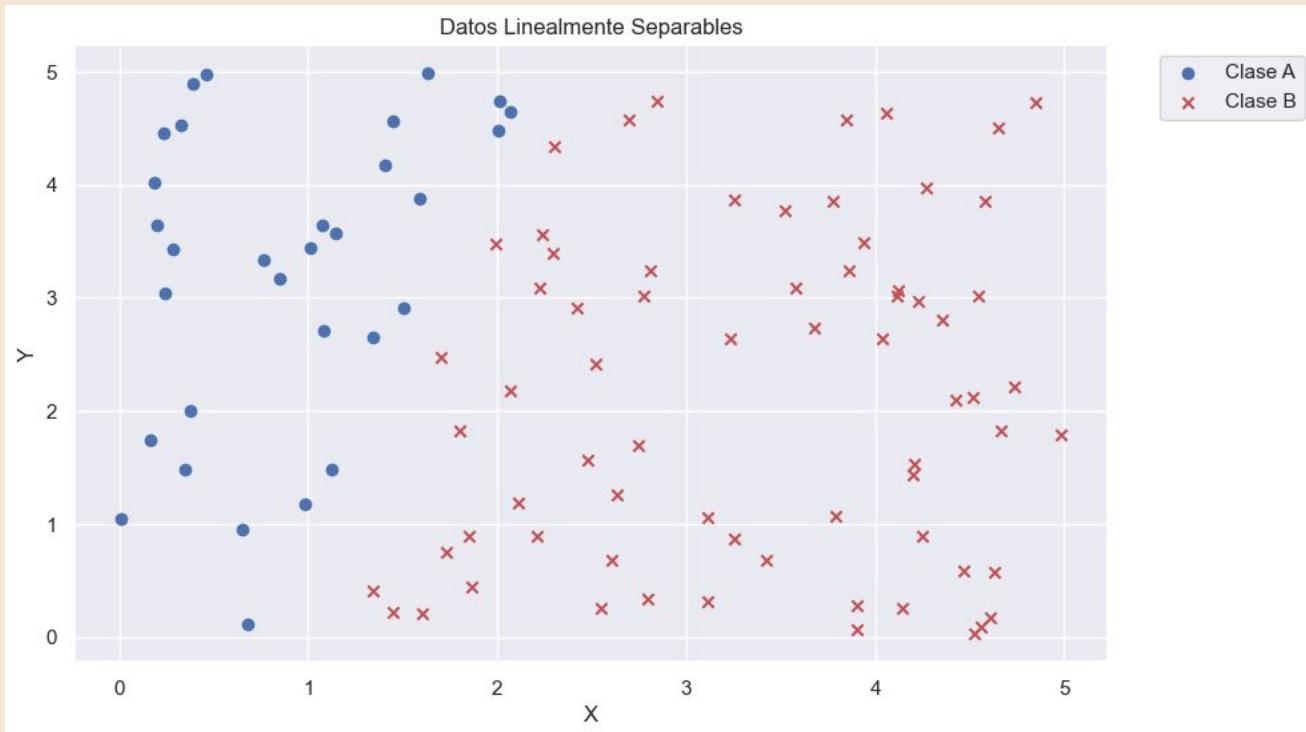
1. Obtener la ecuación de la recta
2. Obtener los n puntos más cercanos a la recta (al menos uno de cada clase)
3. Obtener todas las combinaciones de rectas entre esos n puntos (2 de una clase, 1 de la otra)
4. Evaluar el margen de cada una y tomar la que obtiene mayor margen clasificando los puntos bien



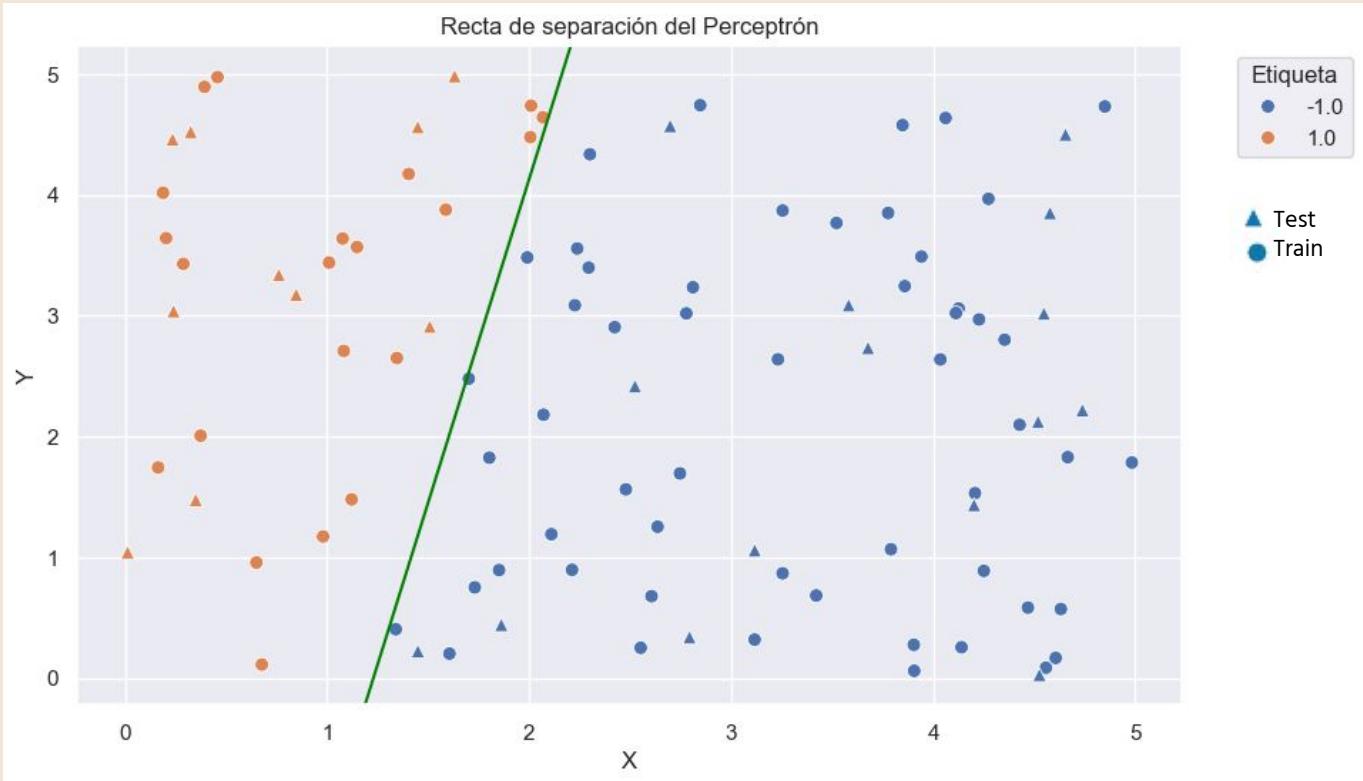
Ejercicio 1.A

- Crear un conjunto linealmente separable
- Entrenar un perceptrón con dicho conjunto
- Verificar optimalidad de la recta obtenida

Ejercicio 1.A



Ejercicio 1.A



División con
K-Fold Cross
Validation con K=4

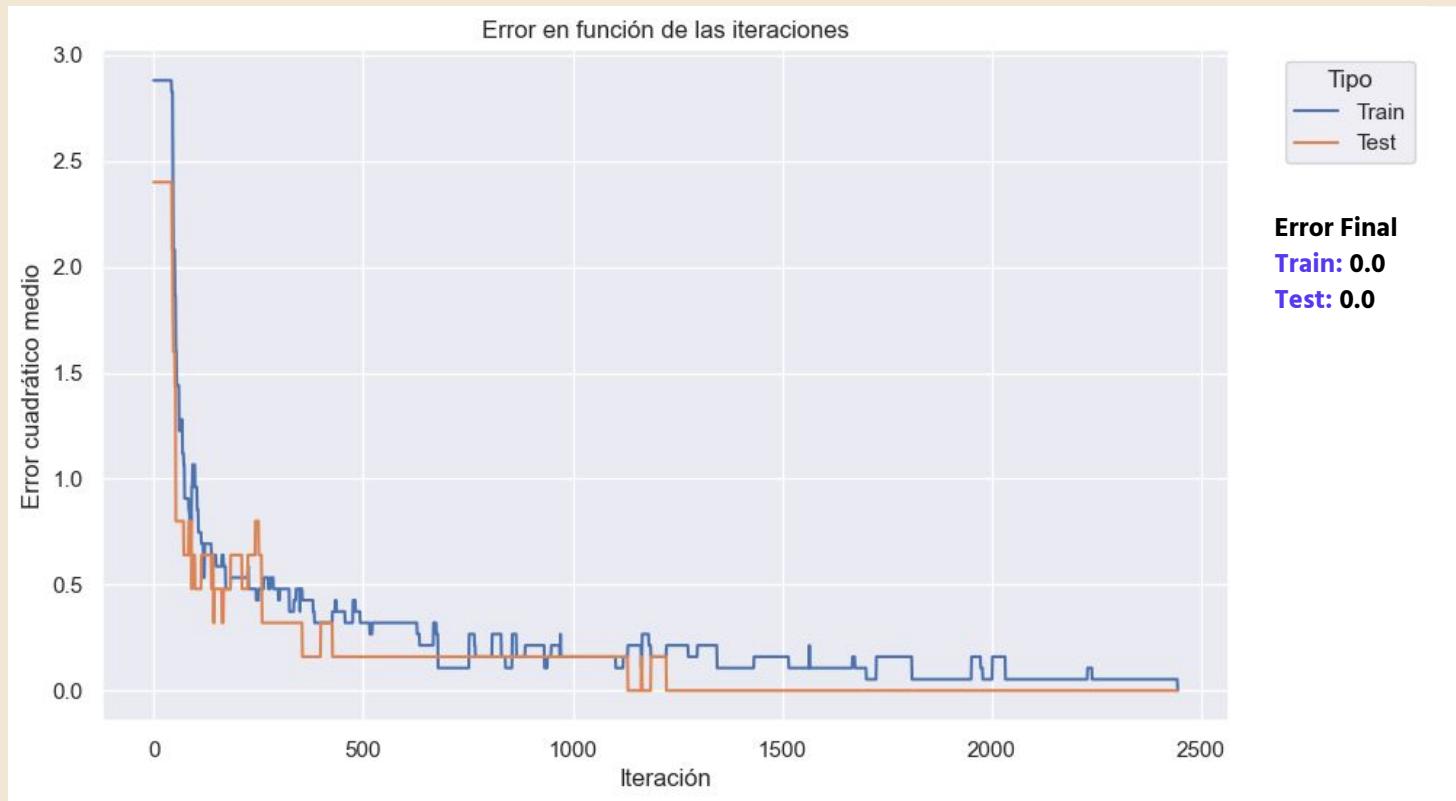
Gradiente
Estocástico

Inicialización
random entre [-1; 1]

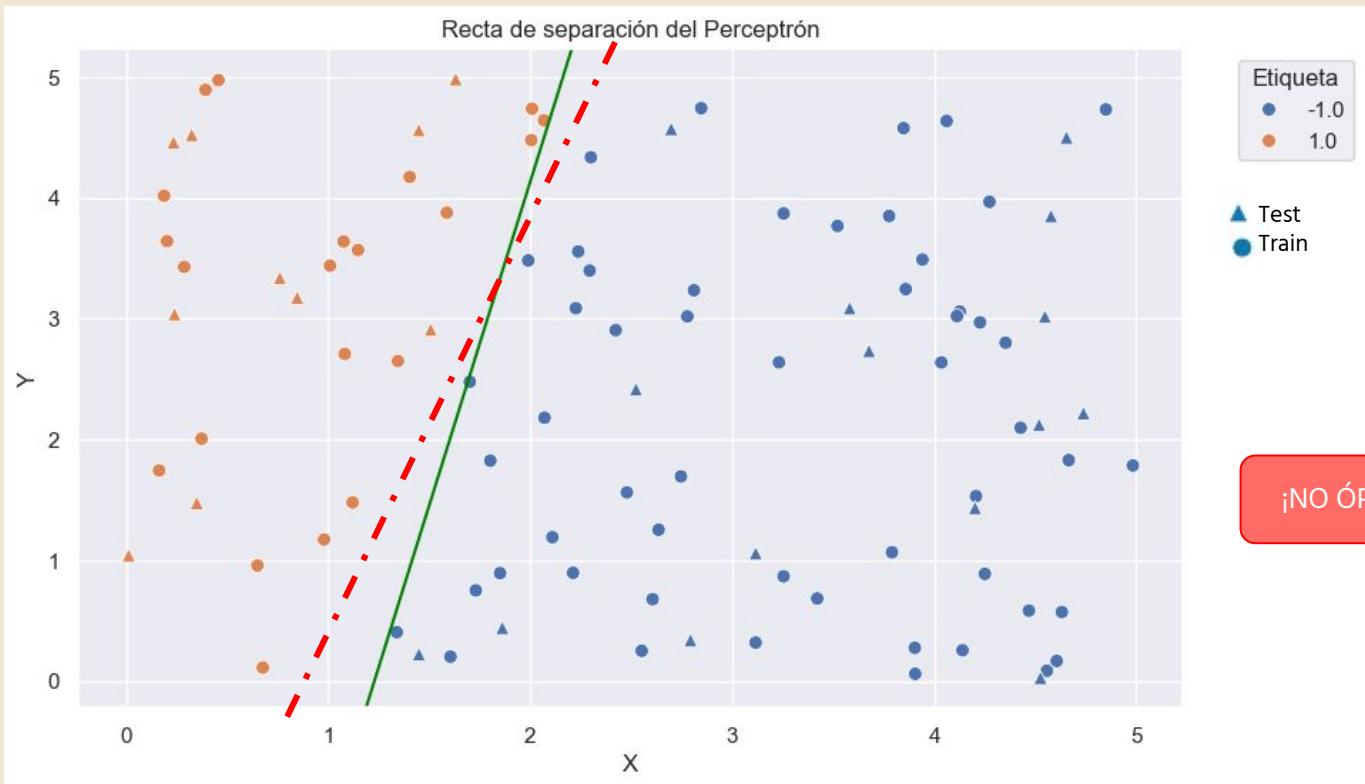
Minimización del
error cuadrático
medio

Learning rate
0.001

Ejercicio 1.A



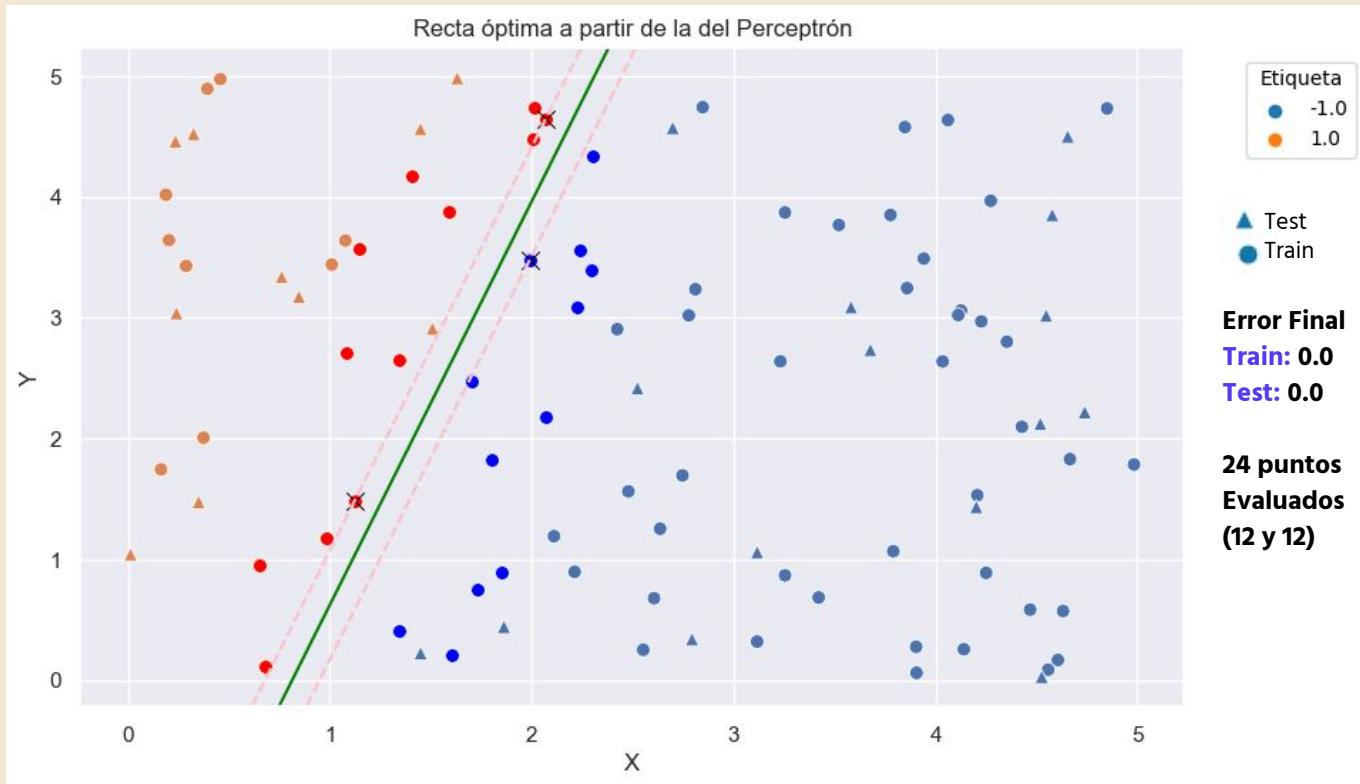
Ejercicio 1.A



Ejercicio 1.B

**Encontrar la recta óptima
a partir de la del Perceptrón**

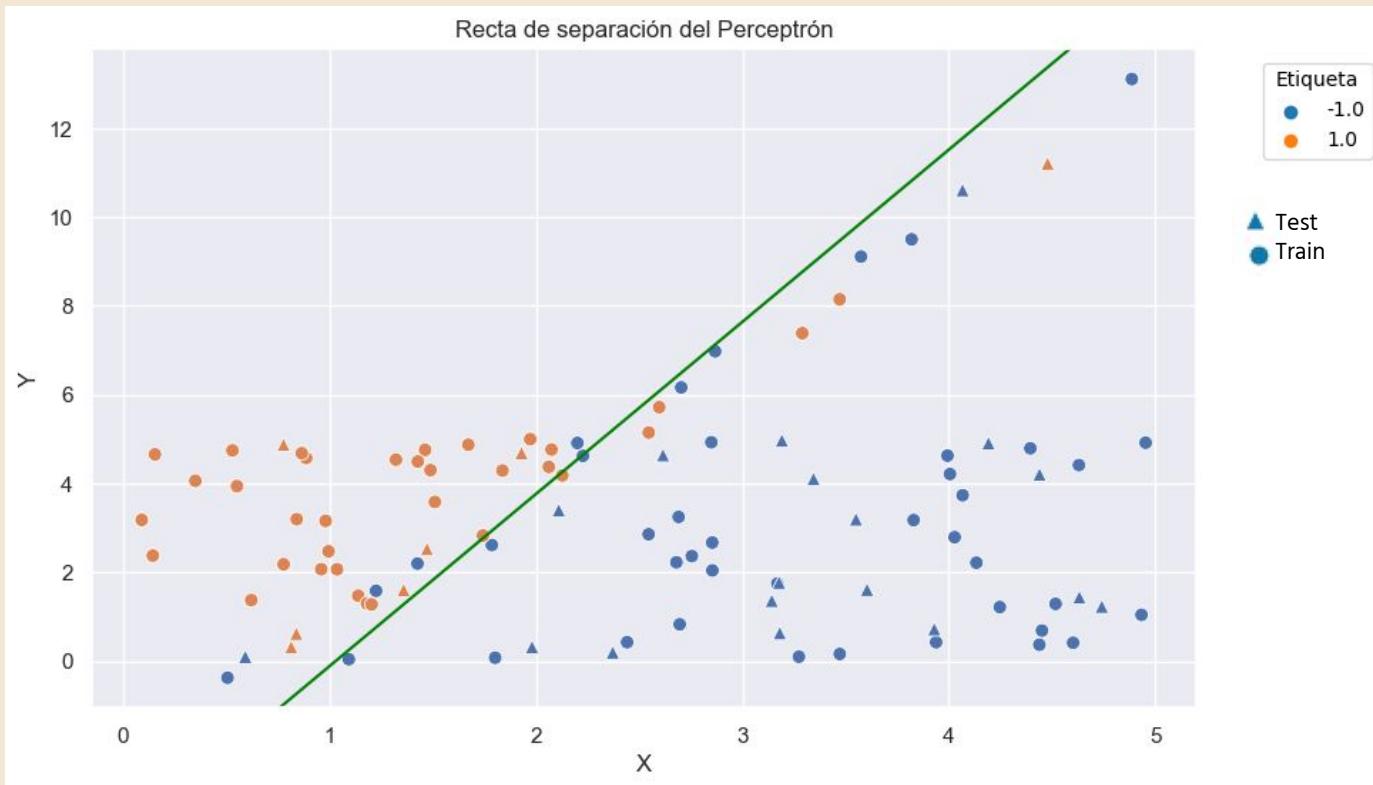
Ejercicio 1.B



Ejercicio 1.C

**Entrenar el Perceptrón con un conjunto
linealmente NO separable**

Ejercicio 1.C





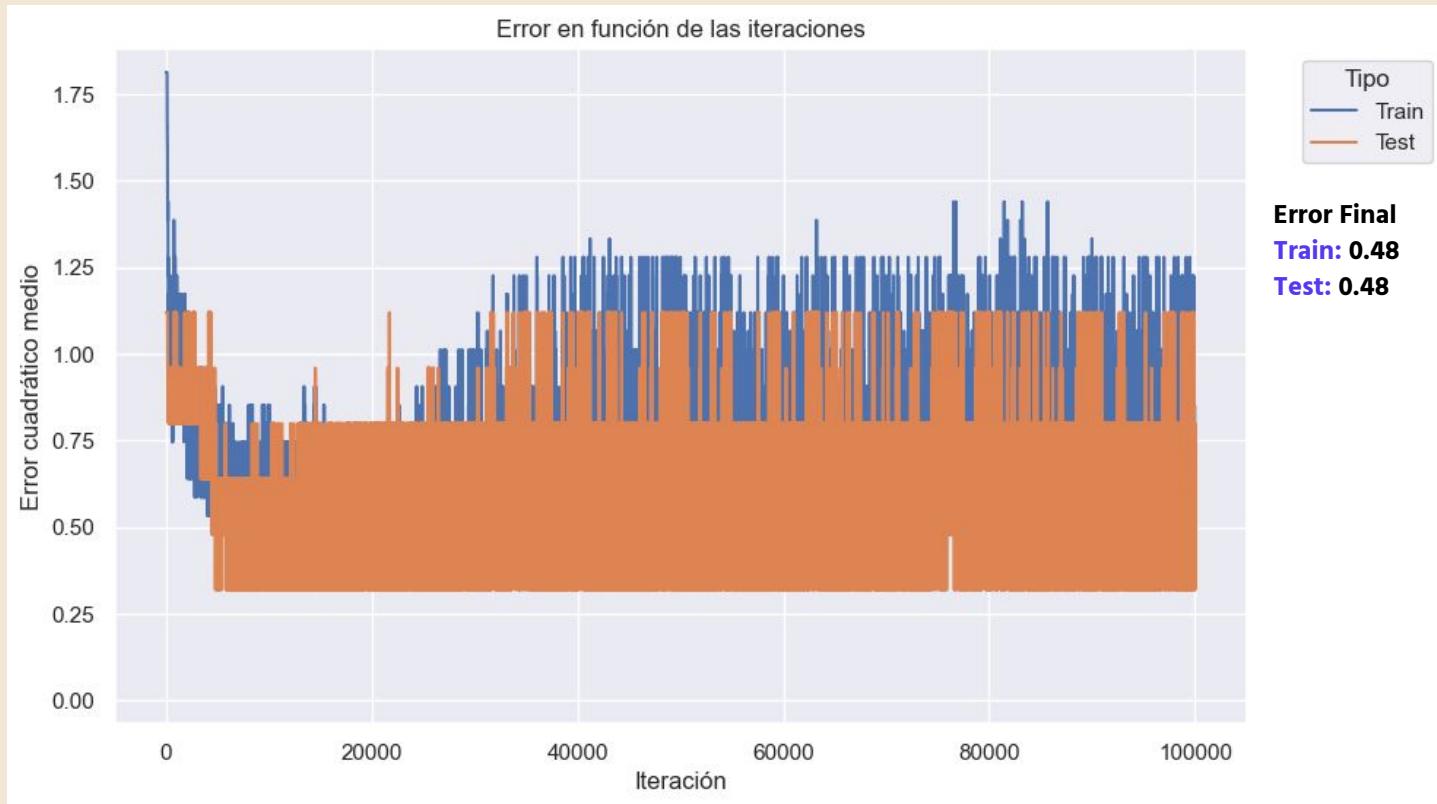
Ejercicio 1.C

¡NO ES POSIBLE SEPARAR LAS CLASES!

- Existirán en toda separación ejemplos mal clasificados
- No hay margen, se debe introducir alguna tolerancia
 - Distancias negativas al hiperplano de separación
- Nuestro método para obtener el hiperplano de margen maximal con el perceptrón ya no nos es útil



Ejercicio 1.C





02

SVM

Support Vector Machine

- Algoritmo de **aprendizaje supervisado** para **clasificación y regresión**.
- Búsqueda del **hiperplano óptimo** que mejor **separe** las distintas clases.
 - El mejor hiperplano será el que tenga la mayor distancia o **margen** entre sí mismo y los puntos más cercanos de cada clase, en cada lado.
 - Si las clases tienen outliers, pueden utilizarse **márgenes tolerantes** regulados por un **parámetro C** que limita cuántas violaciones se permiten a la separación del hiperplano.
 - Mayor C: márgenes más chicos y mejor clasificación
 - Menor C: márgenes más grandes y peor clasificación
- Para **límites de decisión no lineales**, se implementan distintos núcleos que se adapten al espacio de los ejemplos.

SVM Lineal

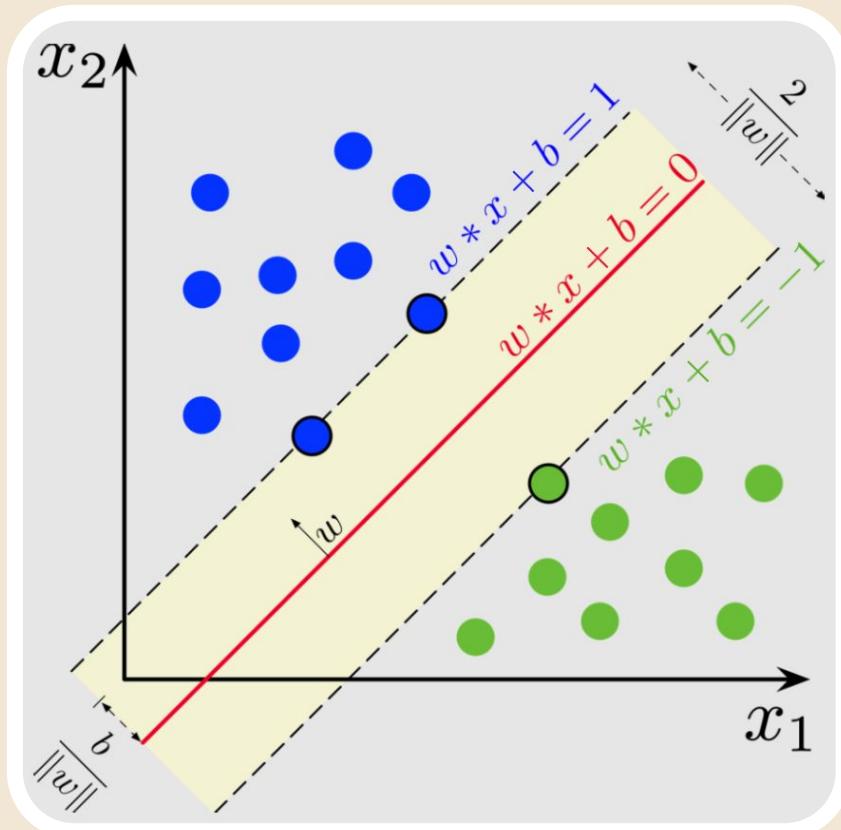
$$y_i(w \cdot x_i + b) \geq 1$$

siendo $y \in \{-1, 1\}$

Hinge Loss

$$l = \begin{cases} 0 & \text{si } t \geq 1 \\ 1 - t & \text{sino} \end{cases}$$

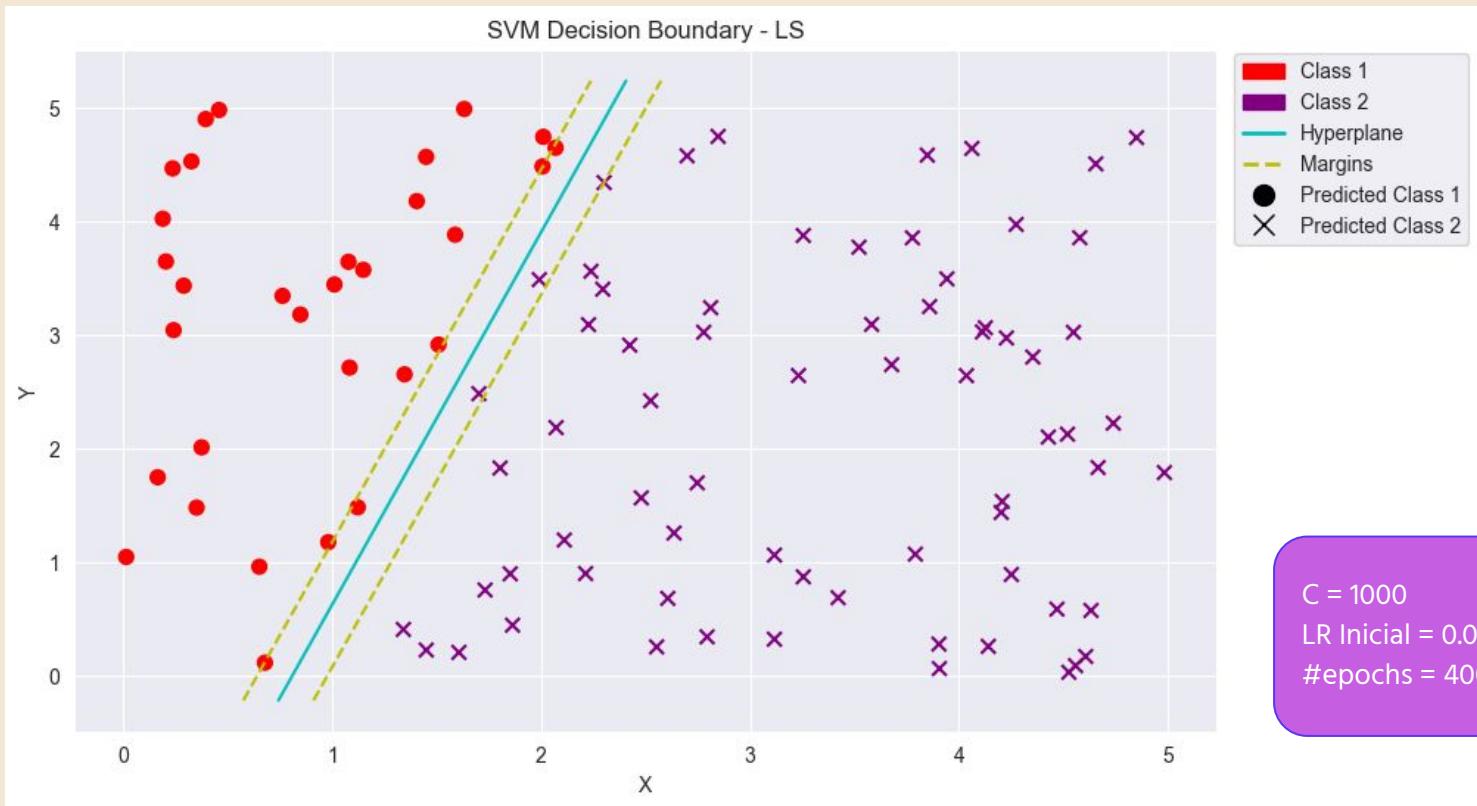
siendo $t = w \cdot x_i + b$



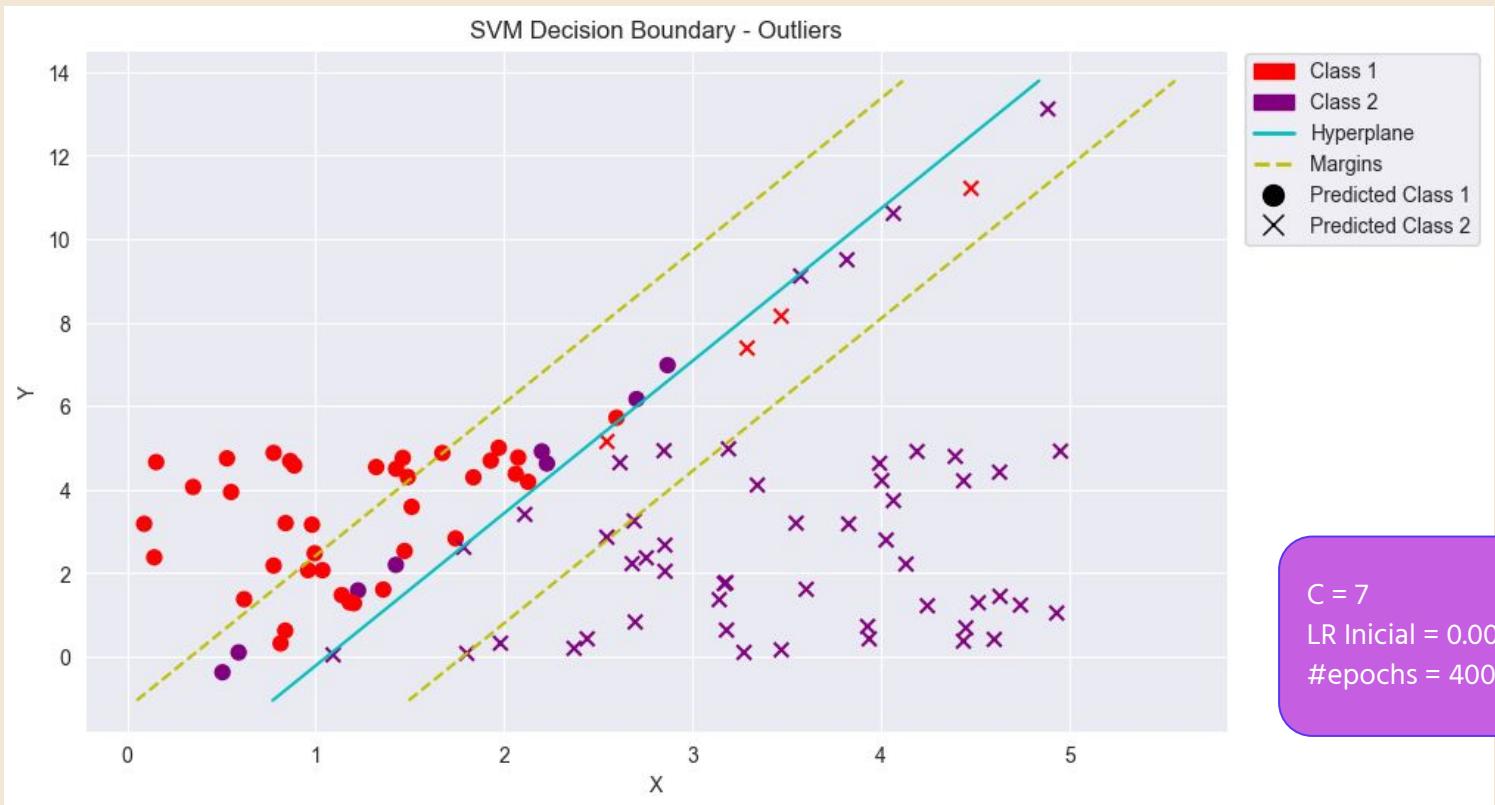
Ejercicio 1.D

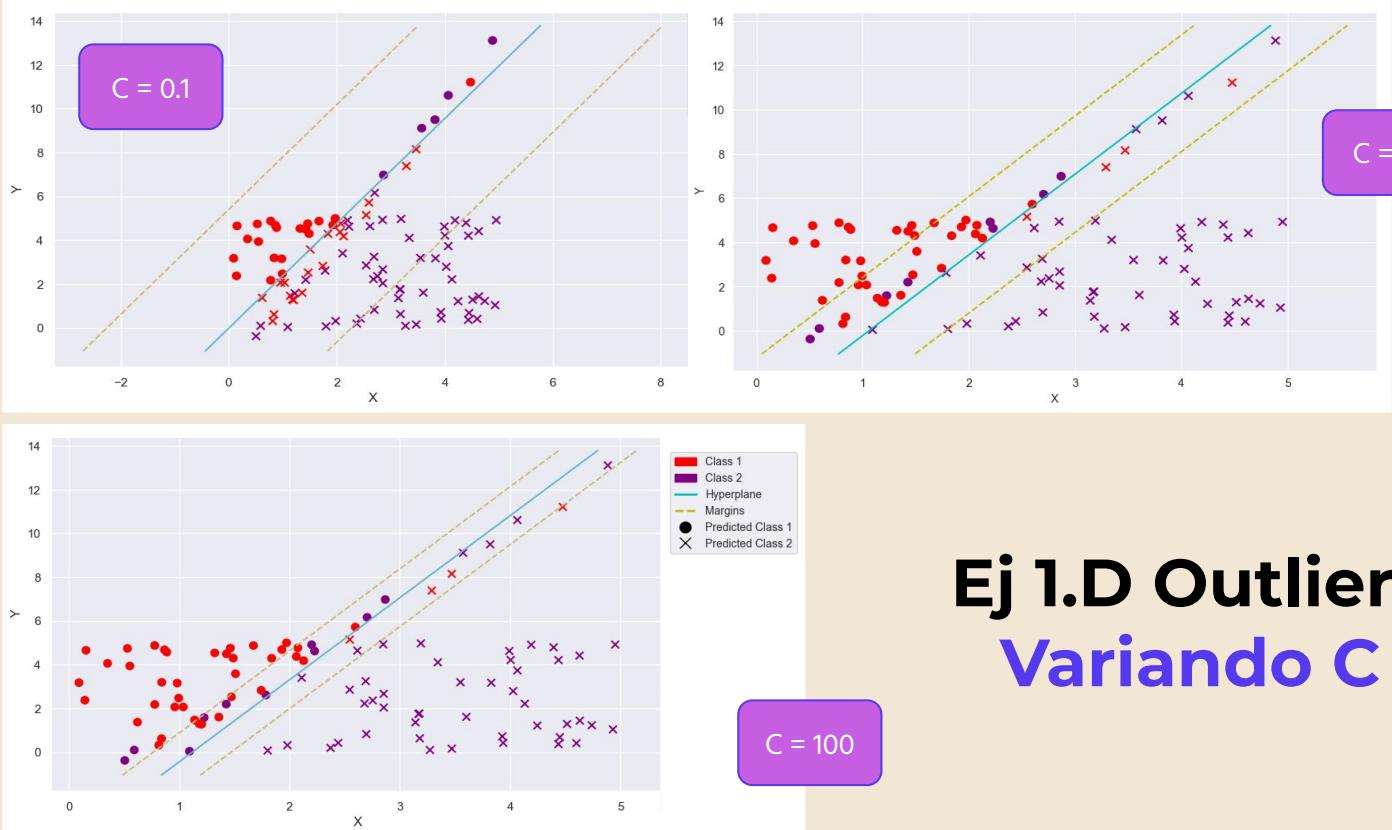
Utilizar los conjuntos anteriores para entrenar una SVM

Ej 1.D Linealmente separable



Ej 1.D Outliers

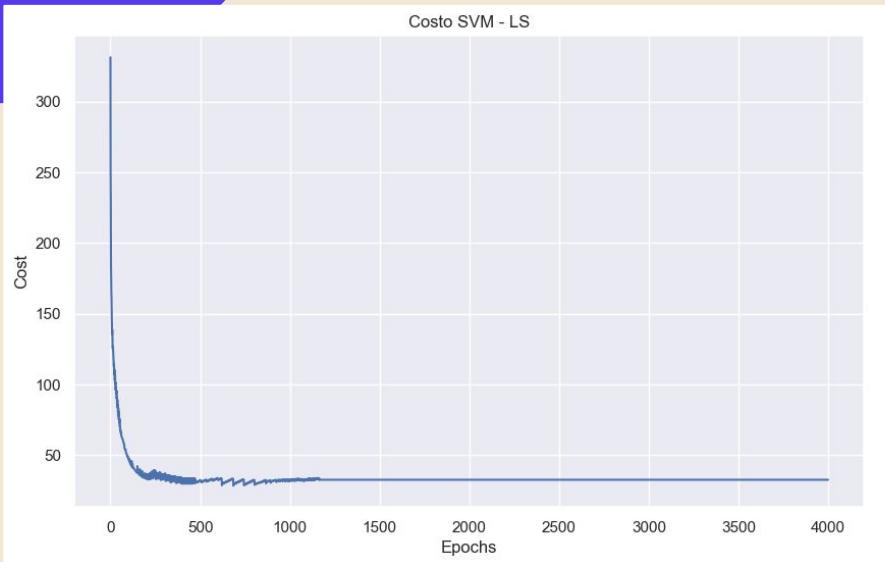




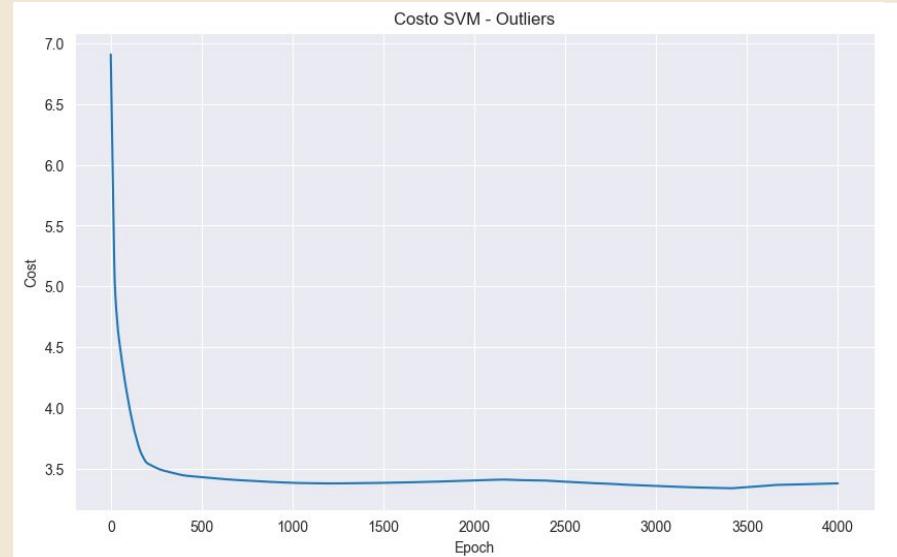
Ej 1.D Outliers Variando C

$C = 100$

Ej 1.D Costo



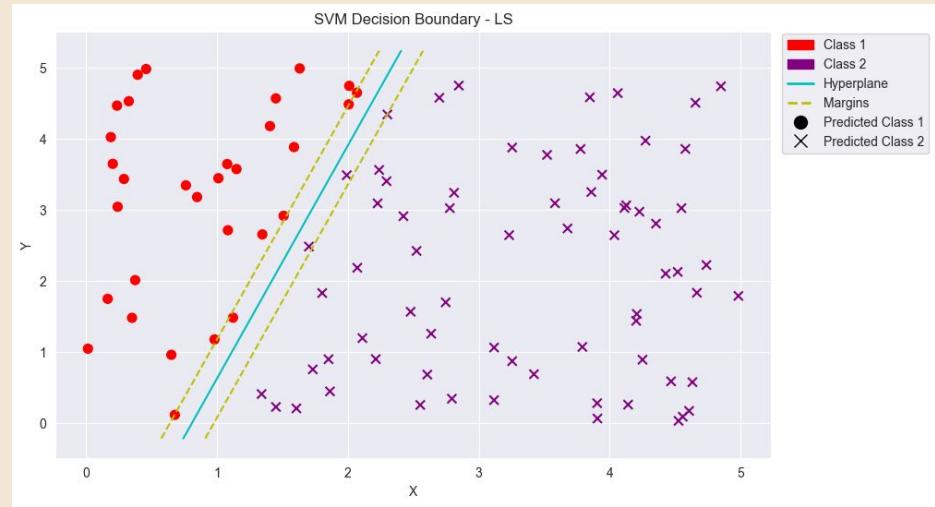
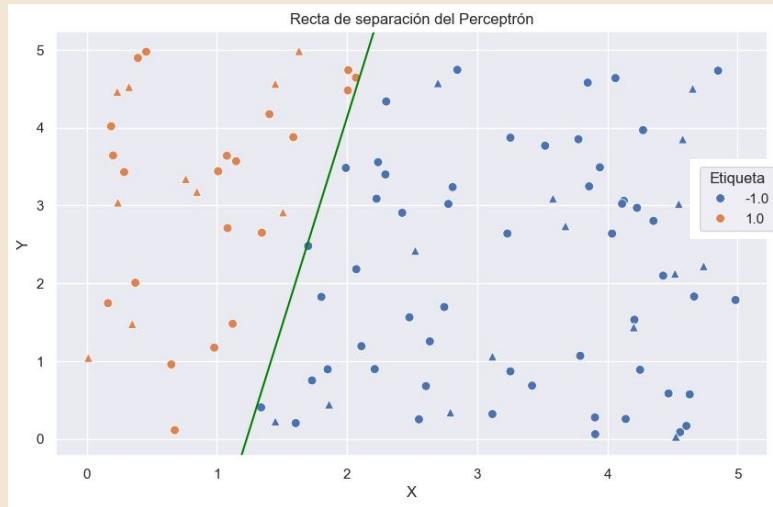
Learning rate inicial = 0.00001
C = 1000



Learning rate inicial = 0.00001
C = 7

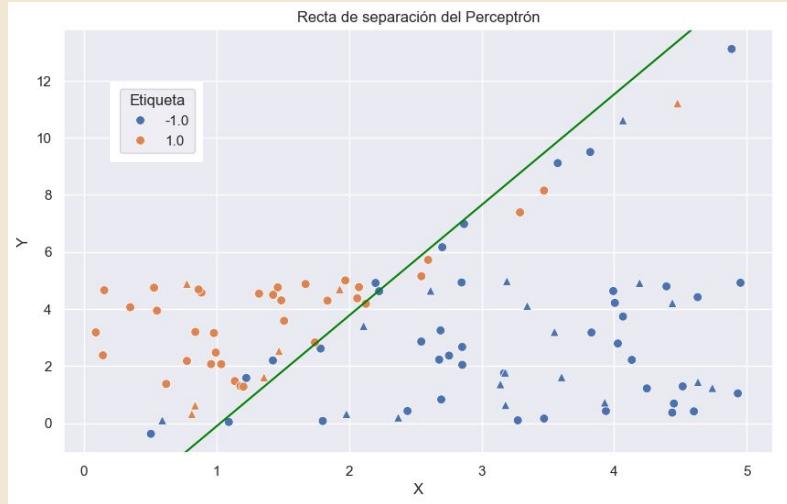
Promedio de la suma de las distancias que están dentro de los márgenes.

Comparación de modelos



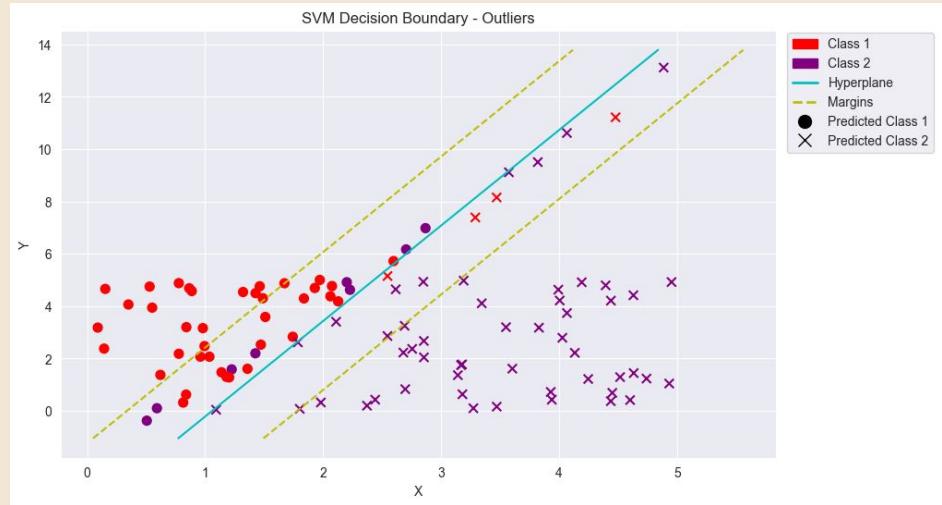
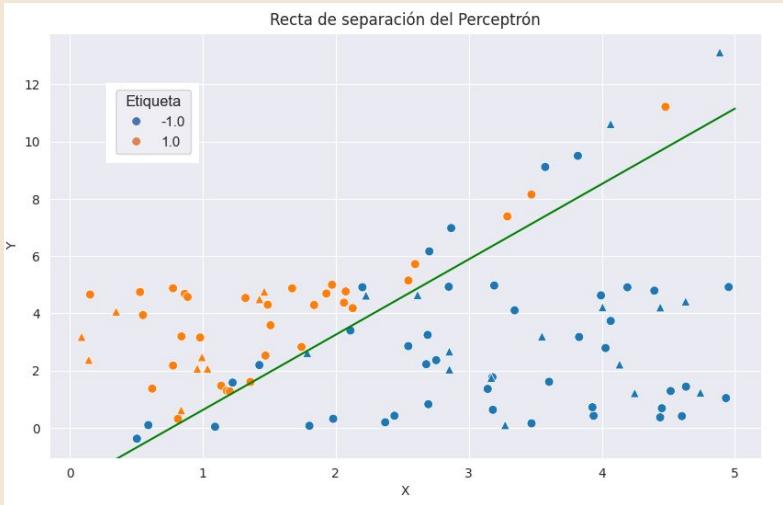
Se ve claramente una mejor predicción generada por el SVM en comparación al perceptrón.

Comparación de modelos



Ambas rectas producen la misma cantidad de predicciones erróneas.
Se generan, en total, once puntos mal clasificados.

Comparación de modelos

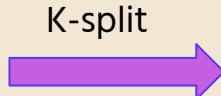


La recta producida por el perceptrón oscila iteración a iteración,
SVM siempre produce la misma

Ejercicio 2

Armar un **dataset** a partir de las imágenes:

- Vaca.jpg
- Cielo.jpg
- Pasto.jpg



- Training set
- Test set

Categorizar los píxeles de distintas imágenes, mostrando a qué clase pertenecen.

Imágenes/píxeles a Vectores

- Utilizando *PIL → Python Imaging Library*
- Se utiliza el método `img.getdata()` y se guarda dentro de un array de numpy
- Se pasa el array a un DataFrame se dividen los valores en columnas RGB.
- Ejemplo:



	R	G	B
0	225	152	99
1	237	164	113
2	223	152	100
3	206	134	84
4	215	145	94

Obtención de Kernel y C óptimo

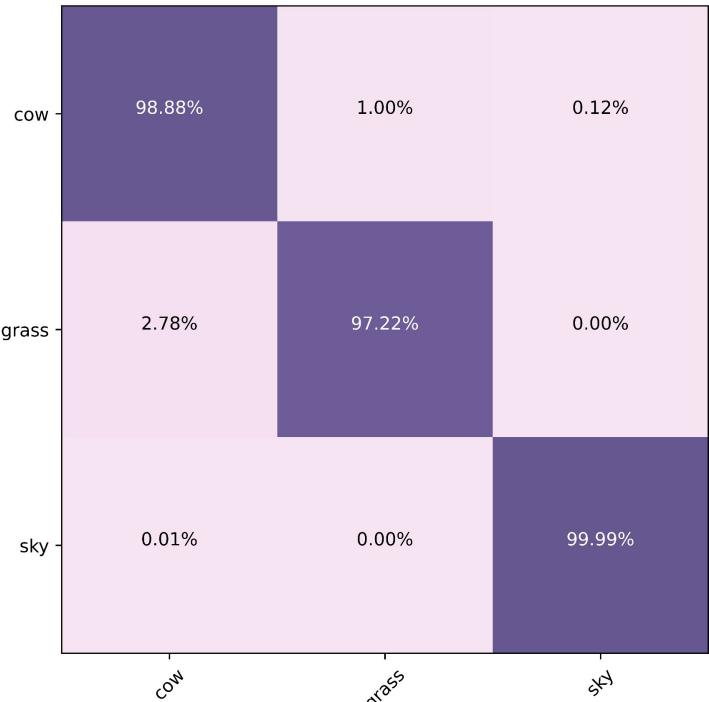
- Utilizando *Cross Validation 80/20* y Todos contra Todos
- Kernels:
 - linear kernel, "*linear*"
 - sigmoid kernel, "*sigmoid*"
 - radial basis function kernel, "*rbf*"
 - polynomial kernel, "*poly*" ($d = 3$)
 - *coeficiente de kernel* = $1/n^{\circ}\text{feats}*\text{var}$
- Inicialmente $C \in [0.5, 50]$
 - con $\text{steps} = 0.25$ hasta 2
 - con $\text{steps} = 1.00$ hasta 50 ← excesivamente lento

Mostraremos a continuación únicamente cuatro variaciones de C.



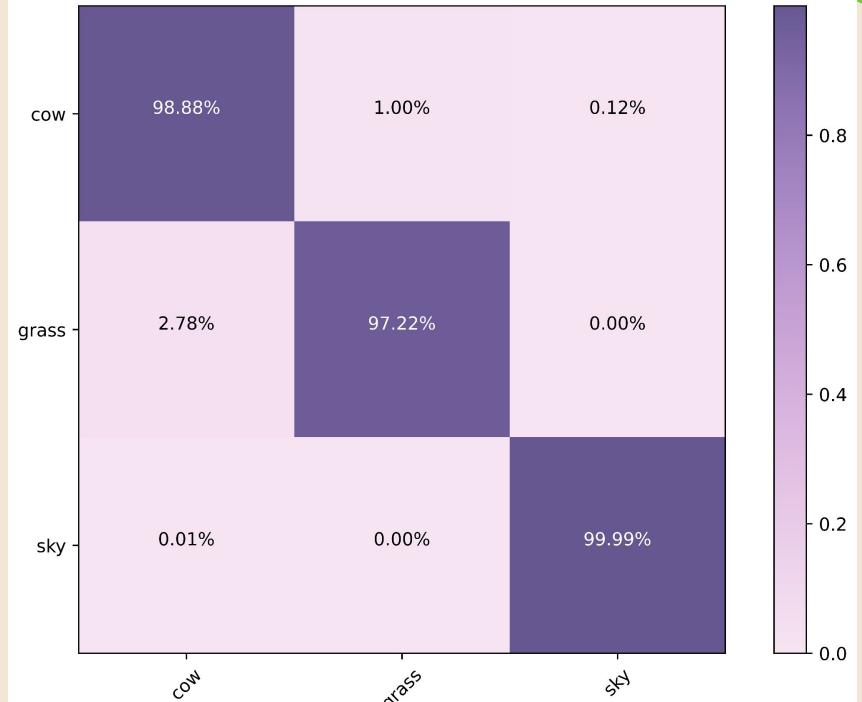
Kernel Lineal

Confusion Matrix for linear kernel, $c = 0.5$



$c = 0.5$, precisión = 0.986, accuracy = 0.986

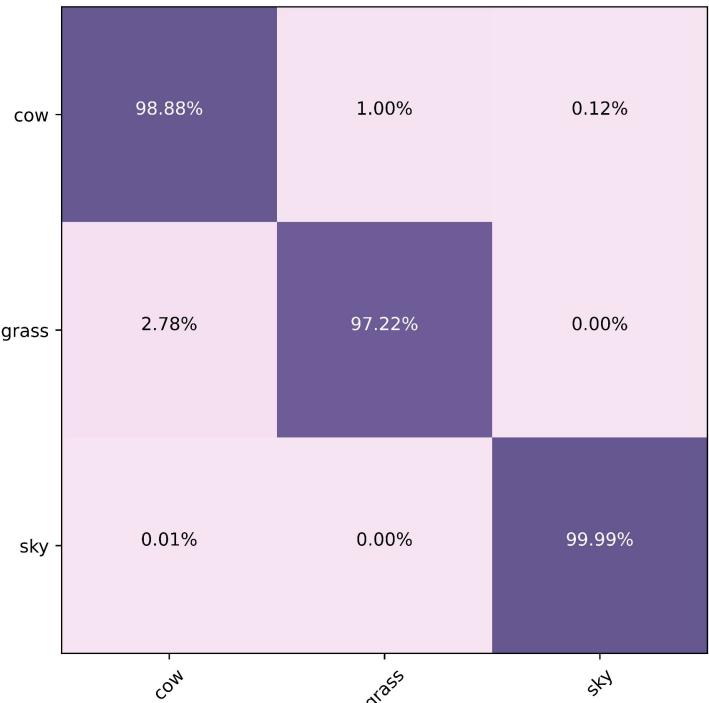
Confusion Matrix for linear kernel, $c = 1.0$



$c = 1$, precisión = 0.987, accuracy = 0.987

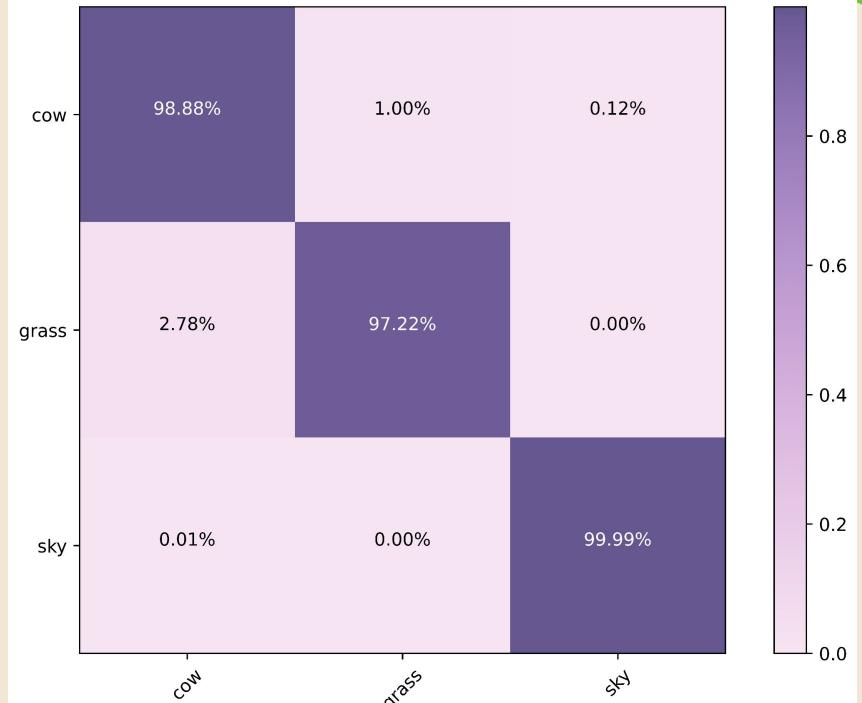
Kernel Lineal

Confusion Matrix for linear kernel, $c = 5.0$



$c = 5$, precisión = 0.987, accuracy = 0.987

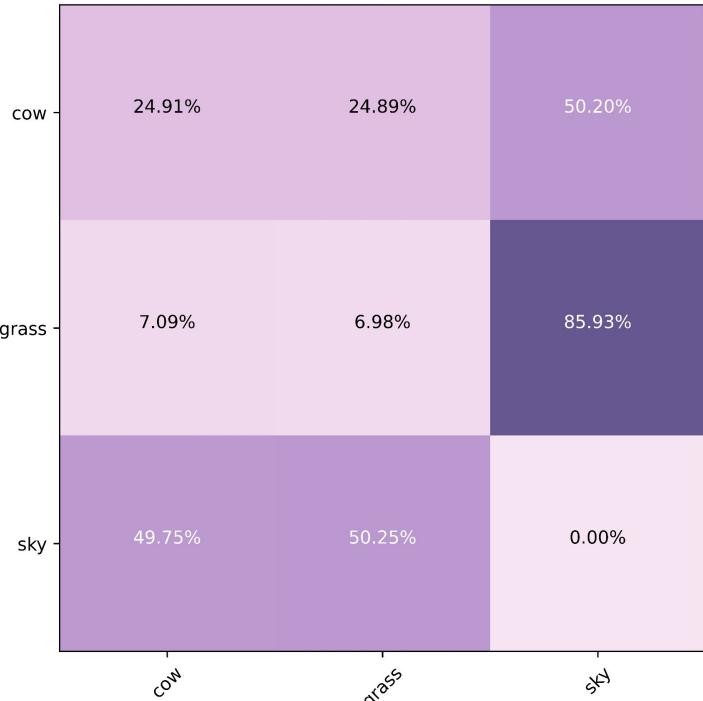
Confusion Matrix for linear kernel, $c = 10.0$



$c = 10$, precisión = 0.987, accuracy = 0.987

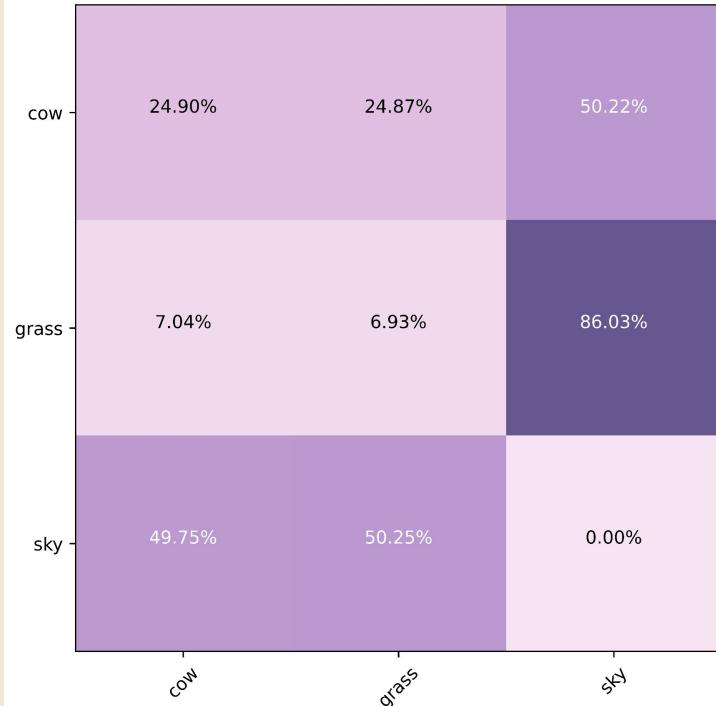
Kernel Sigmoideo

Confusion Matrix for sigmoid kernel, $c = 0.5$



$c = 0.5$, precisión = 0.199, accuracy = 0.127

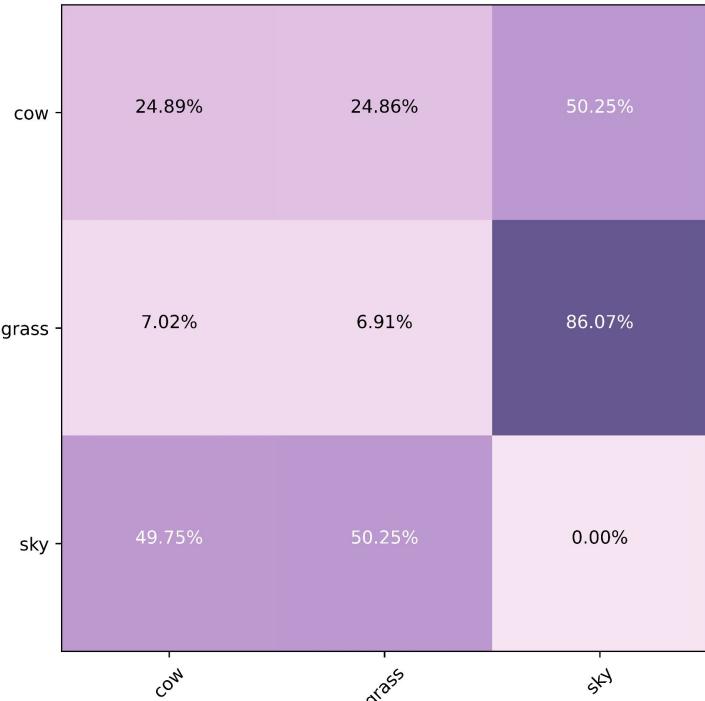
Confusion Matrix for sigmoid kernel, $c = 1.0$



$c = 1$, precisión = 0.199, accuracy = 0.127

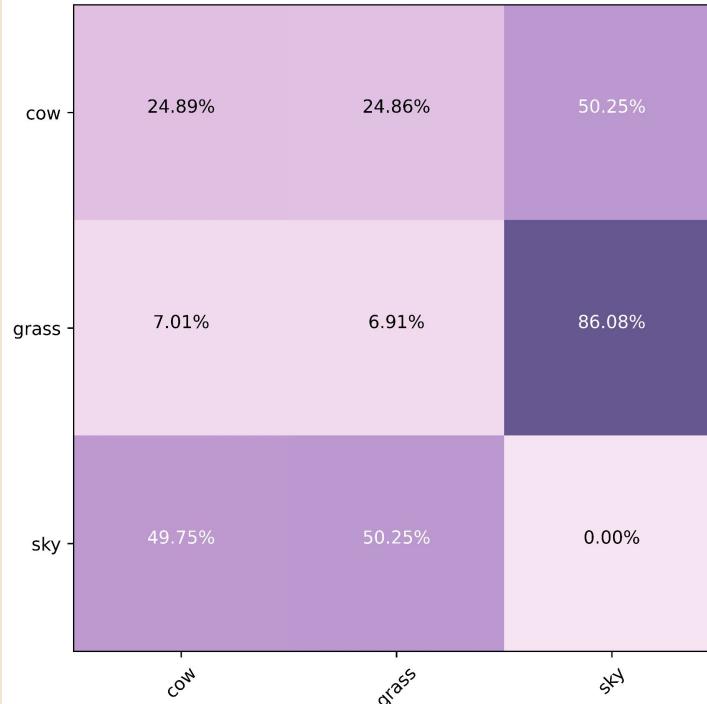
Kernel Sigmoideo

Confusion Matrix for sigmoid kernel, $c = 5.0$



$c = 5$, precisión = 0.199, accuracy = 0.127

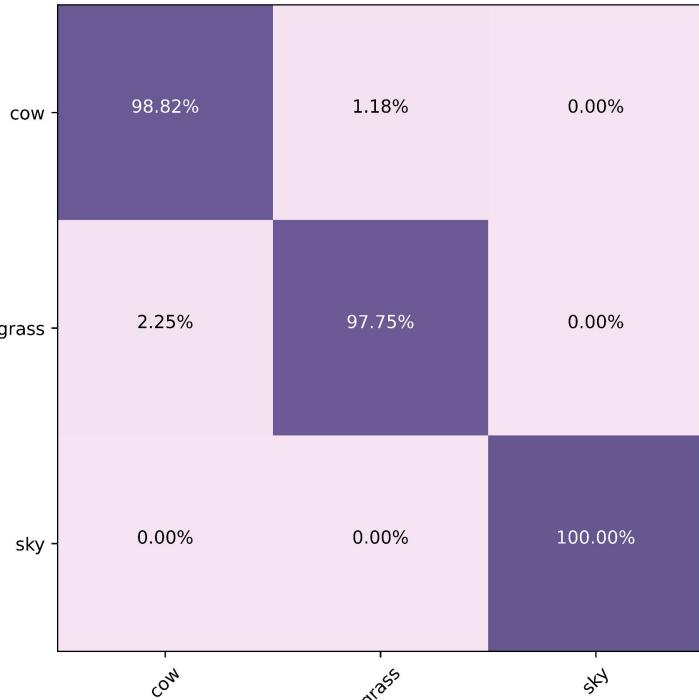
Confusion Matrix for sigmoid kernel, $c = 10.0$



$c = 10$, precisión = 0.199, accuracy = 0.127

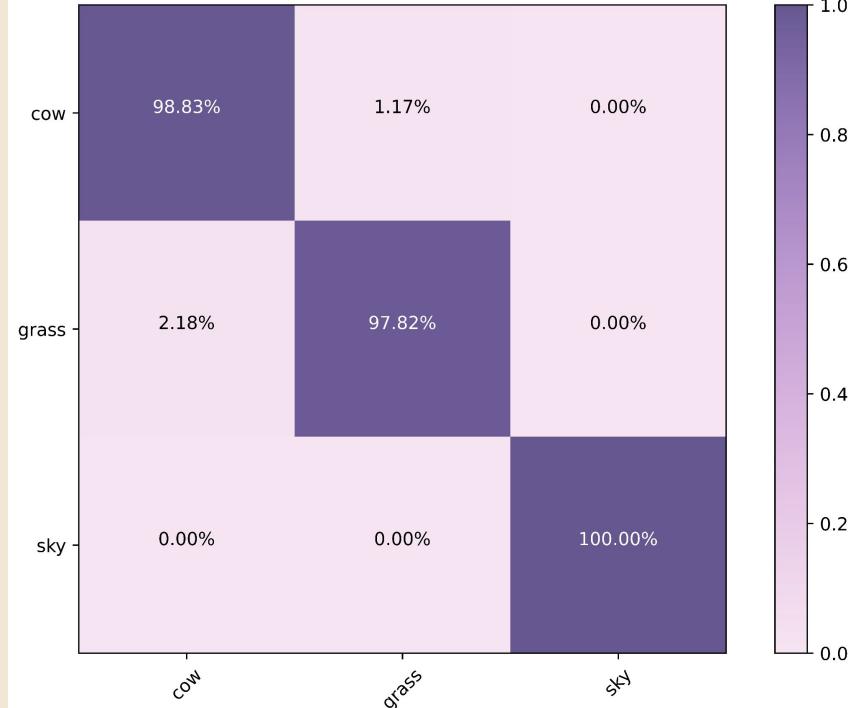
Kernel RBF

Confusion Matrix for rbf kernel, c = 0.5



c = 0.5, precisión = 0.988, accuracy = 0.988

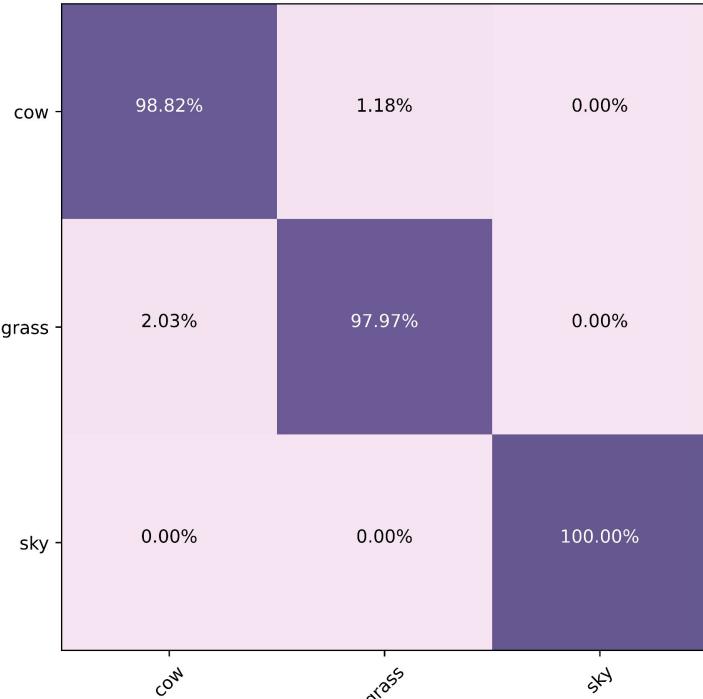
Confusion Matrix for rbf kernel, c = 1.0



c = 1, precisión = 0.988, accuracy = 0.988

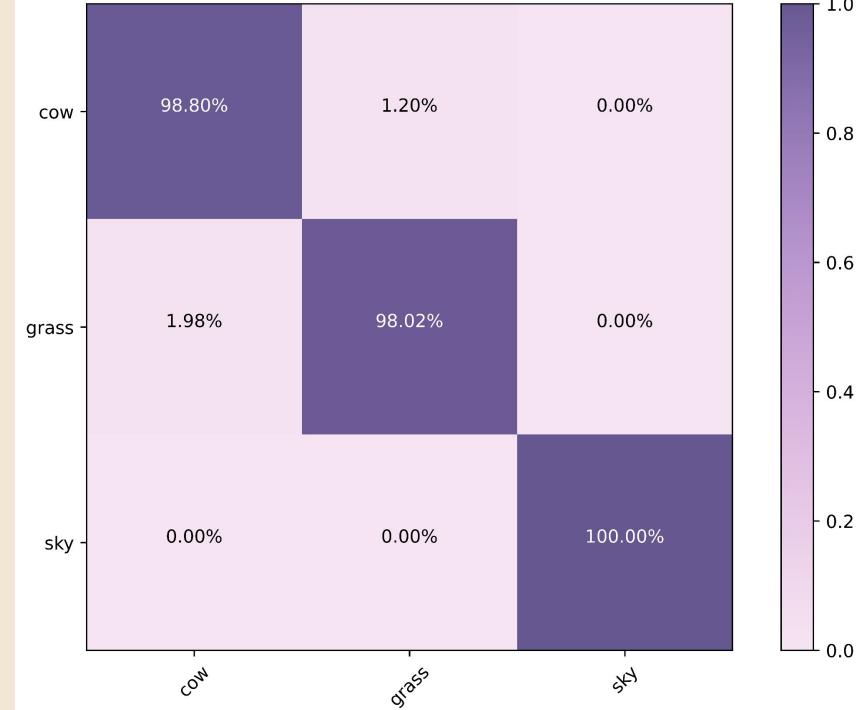
Kernel RBF

Confusion Matrix for rbf kernel, c = 5.0



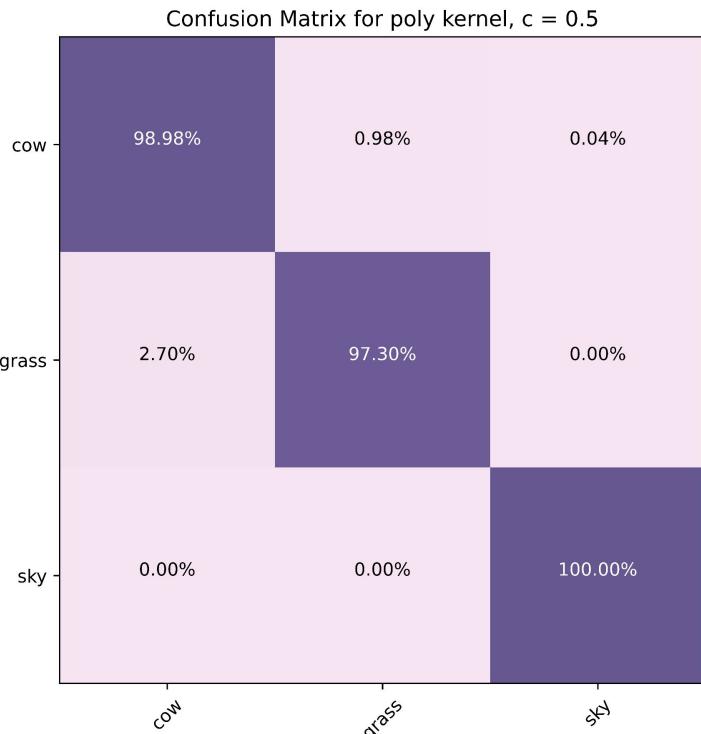
c = 5, precisión = 0.989, accuracy = 0.989

Confusion Matrix for rbf kernel, c = 10.0

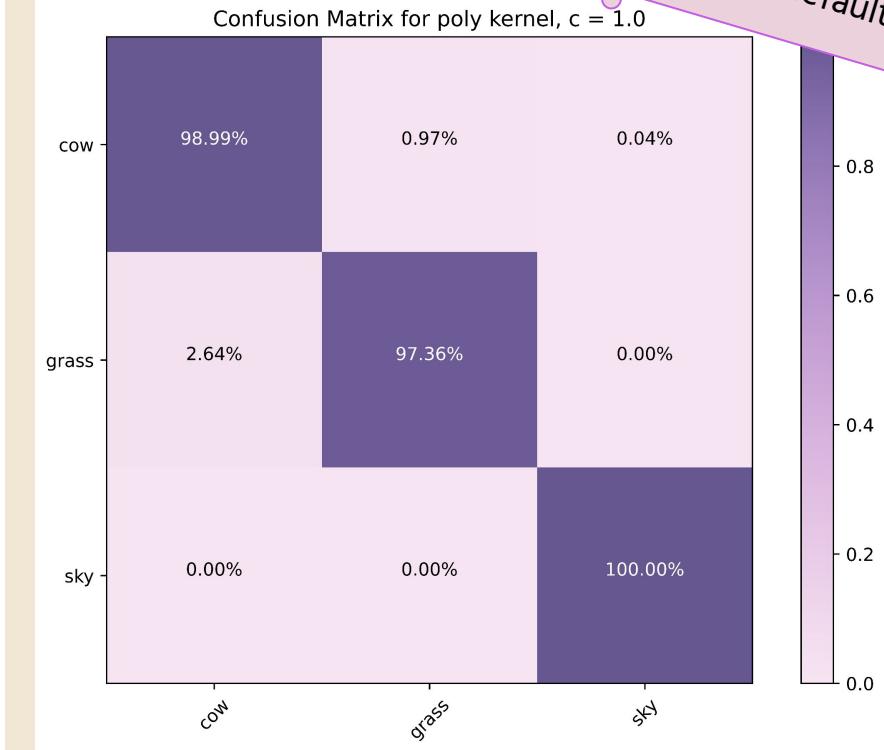


c = 10, precisión = 0.989, accuracy = 0.989

Kernel Polinómico



c = 0.5, precisión = 0.987, accuracy = 0.987

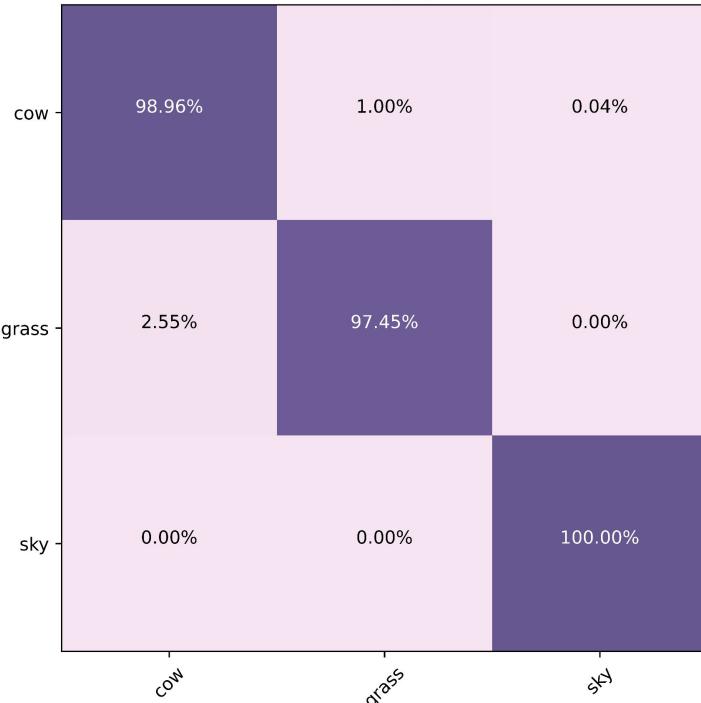


c = 1, precisión = 0.988 , accuracy = 0.988

Degree default 3!

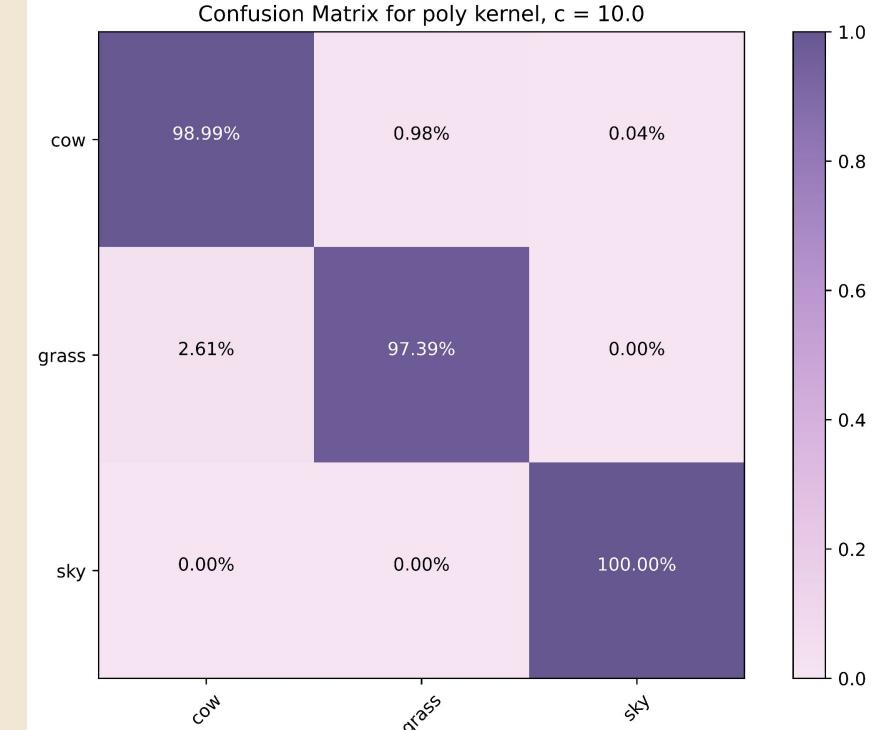
Kernel Polinómico

Confusion Matrix for poly kernel, c = 5.0



c = 5, precisión = 0.988 , accuracy = 0.988

Confusion Matrix for poly kernel, c = 10.0



c = 10, precisión = 0.988 , accuracy = 0.988



¿Cuál es el Kernel y C óptimo?

Nuestra idea:

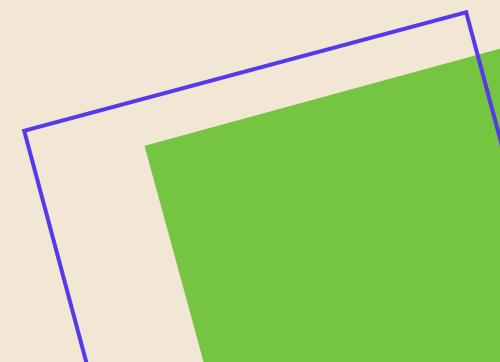
Buscar un punto medio entre:

- Precisión y *accuracy*
- Tiempo Total de Ejecución

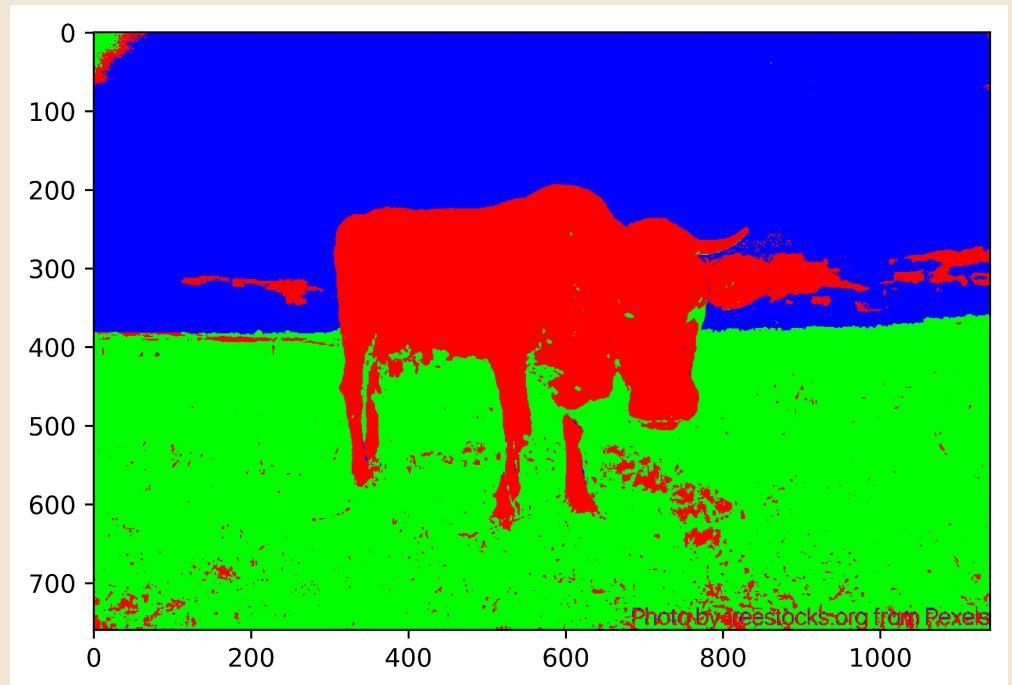
Observaciones:

- Lineal → lento, buena precisión
- Sigmoideo → **excesivamente** lento y baja precisión
- Polinómico → rápido y buena precisión
- RBF → **el mejor**: muy rápido y excelente precisión

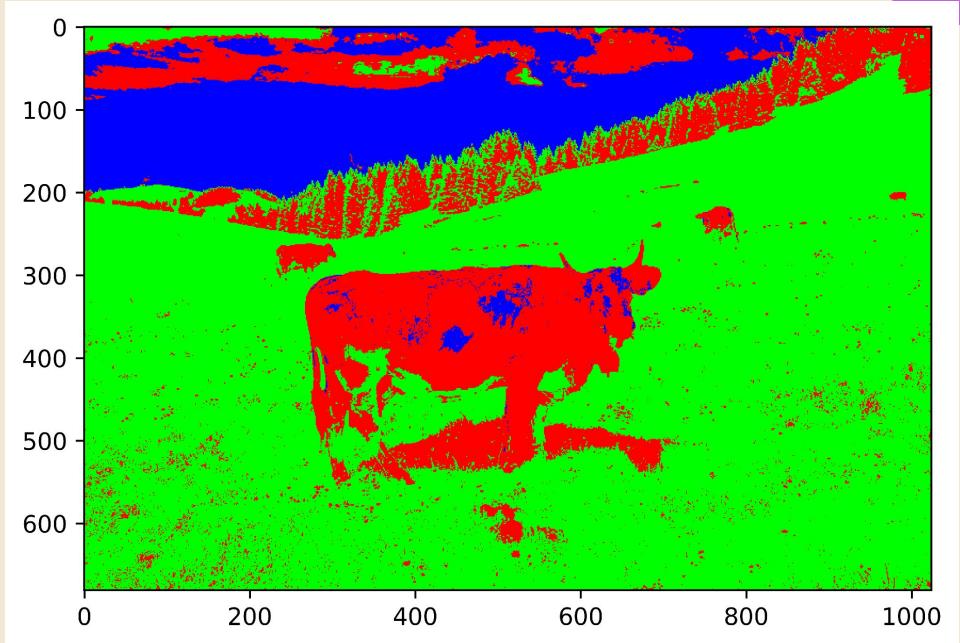
¡Utilizaremos el kernel RBF con C = 5!



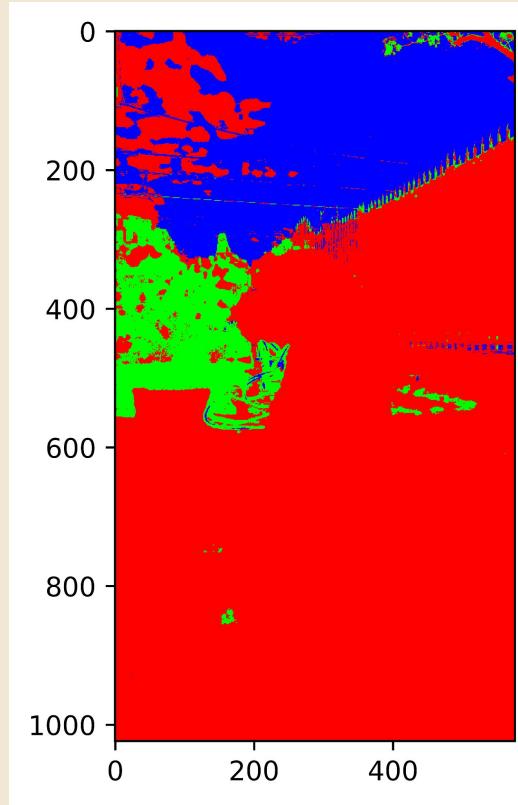
Clasificación de Imagen original



Clasificación de Imagen similar: Milka



Clasificación de Imagen alternativa: Sirio



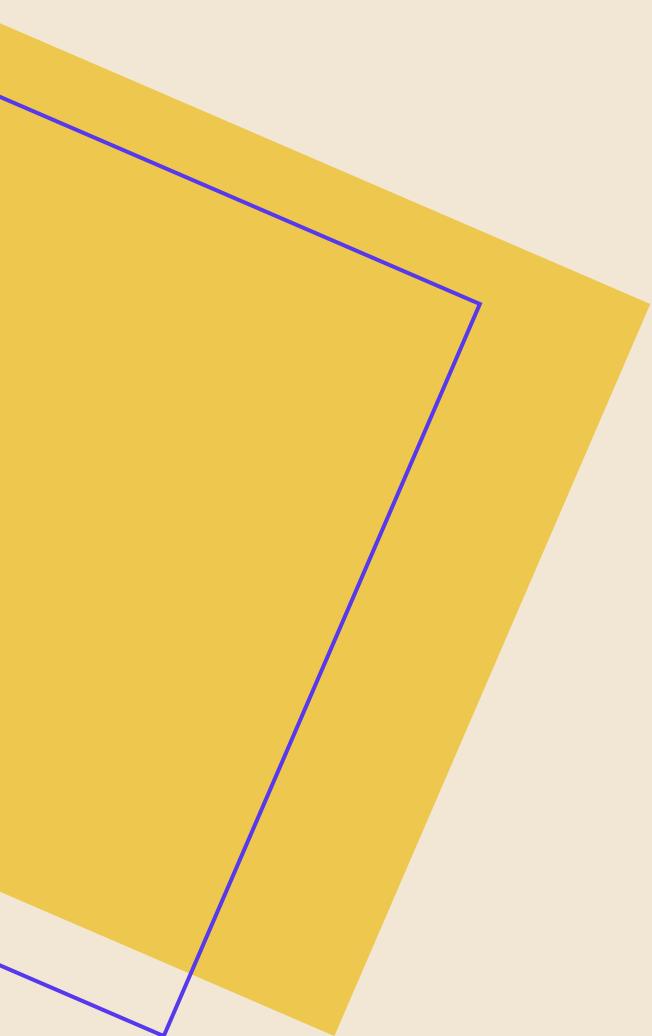
¿Es el resultado esperado?

¡SÍ!

Mapea a los colores no conocidos con aquellos más cercanos en su RGB:

- rojo/rosa → vaca
- negro → vaca
- violeta → vaca

Clasifica correctamente el verde, celeste y marrón, pero es malo extrapolando lo que no conoce.

A large, solid yellow parallelogram is positioned in the upper-left corner of the slide. It is tilted at approximately a 45-degree angle and has a thin blue outline. It overlaps the background color.

03

Nuestras Conclusiones



Nuestras Conclusiones

- SVM es más robusto que perceptrón simple frente a conjuntos con *outliers* ya que se basa en los ejemplos más cercanos (vectores de soporte)
- Observación  : a mayor C, mayor el tiempo de ejecución de SVM.
- Podemos inferir que la información del Ejercicio 2 es linealmente separable y posiblemente muy separados en el espacio, dado que utilizando un kernel lineal el error de predicción no cambia (por consiguiente, se encontró el hiperplano óptimo) independientemente del valor de C que se utilice.

Nuestras Conclusiones

- SVM no será bueno haciendo inferencias de cosas que no conoce: lo mapeará al valor más cercano de aquello conocido.
- El perceptrón simple escalón no produce una recta que maximice el margen.
- El algoritmo de maximización de recta no funciona bien para conjuntos no linealmente separables
- En el perceptrón, los puntos candidatos a vectores de soporte no necesariamente son los más representativos de su clase → hay que elegir una buena cantidad para que recubran correctamente a su clase.



¡Gracias!

¿Preguntas?

🐮 Grupo 3 🐄

Mila Langone
Paula Oseroff
Luciana Diaz Kralj
Paula Domingues