



Trabajo Práctico Nro. 3: Algoritmos de Vectores de Soporte

Ezequiel Agustin Perez, Gonzalo Nahuel Baliarda, Lucas Agustin Vittor

Instituto Tecnológico de Buenos Aires

{eperez, gbaliarda, lvittor}@itba.edu.ar

11 de octubre de 2023

Ejercicio 1

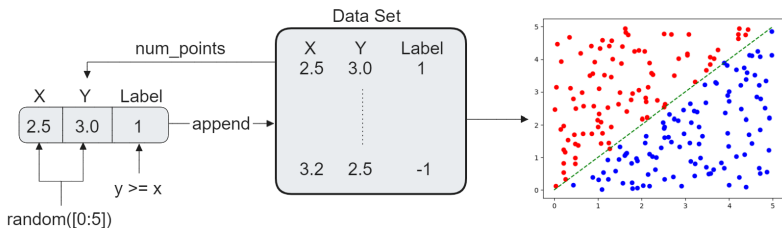


Enunciado

1. Construir un dataset linealmente separable y usar un perceptrón simple para clasificarlo
2. Obtener el hiperplano óptimo a partir del hiperplano del perceptrón
3. Construir un dataset con ejemplos mal clasificados y usar un perceptrón simple para clasificarlo
4. Clasificar los dataset con SVM

Dataset 1

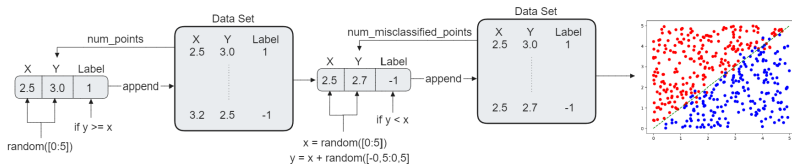
Randomizado y dividido en 80 % train, 20 % test.





Dataset 2

Randomizado y dividido en 80 % train, 20 % test.



Datasets

- ▶ TP3-1: 200 puntos
- ▶ TP3-2: 400 puntos bien clasificados y 40 puntos clasificados aleatoriamente.

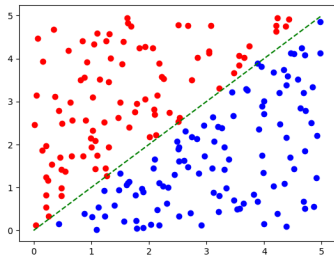


Figura: TP3-1

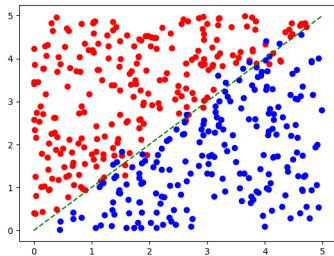


Figura: TP3-2



Perceptrón Simple: modelo

Aprendizaje

$$w_i^{nuevo} = w_i^{viejo} + \Delta w_i \quad \text{con } \Delta w_i = 2\eta \xi_i^\mu \zeta_i^\mu$$

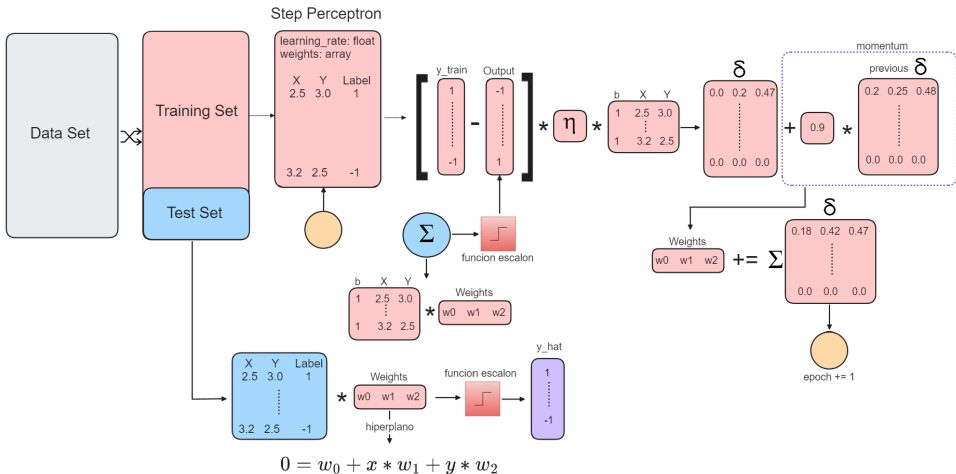
Función escalón

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{sino} \end{cases}$$

Momentum

$$\Delta = \Delta + \mu * \Delta_{prev}$$

Perceptrón Simple: implementación





SVM: modelo

Hinge Loss

$$l = \frac{1}{2}w^2 + \Sigma[y_i(wx_i + b)]$$

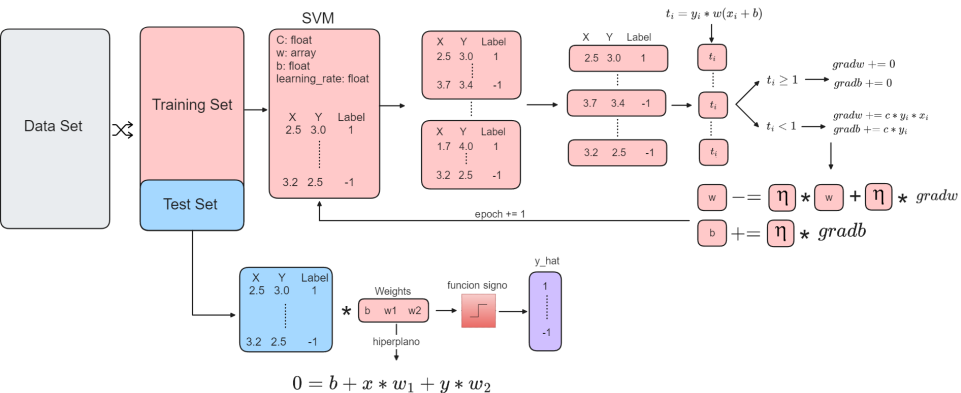
Gradiente descendente

$$w = \begin{cases} w + C \sum_{i=1}^p (-1)y_i x_i & t < 1 \\ w + C \sum_{i=1}^p 0 = w & t \geq 1 \end{cases}$$

$$b = \begin{cases} C \sum_{i=1}^p (-1)y_i & t < 1 \\ C \sum_{i=1}^p 0 = 0 & t \geq 1 \end{cases}$$



SVM: implementación



Resultados: entrenamiento perceptrón

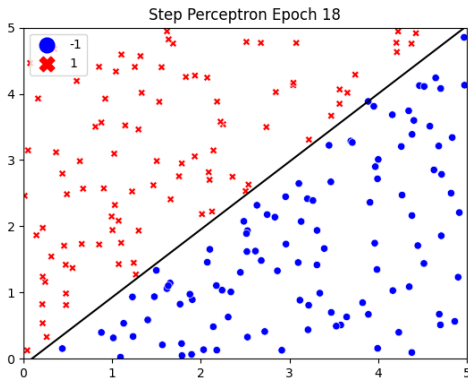


Figura: Step Perceptron

Resultados: hiperplano óptimo

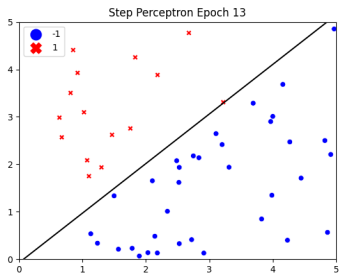


Figura: Inicio

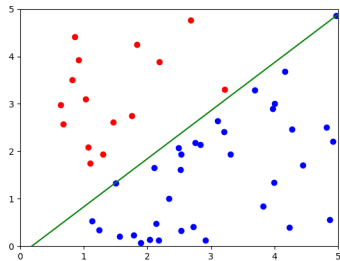


Figura: Dirección

Resultados: hiperplano óptimo

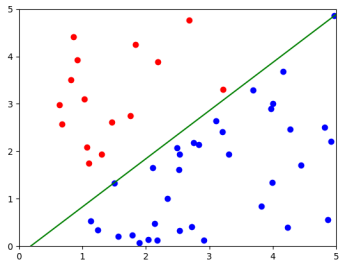


Figura: Dirección

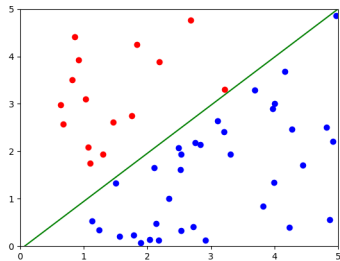


Figura: Traslación

Resultados: test perceptron

► $\eta = 0.1$; $\mu = 0,9$; epochs = 50

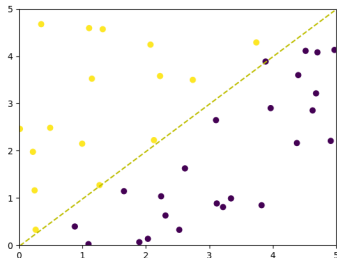


Figura: Test TP3-1

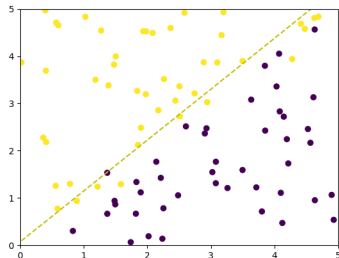


Figura: Test TP3-2

Resultados: test SVM

► $C = 2$; $\eta = 0.001$; epochs = 1000

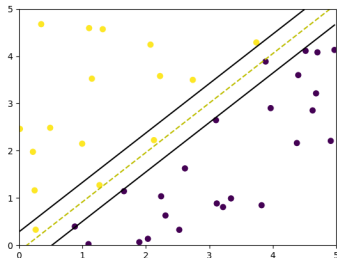


Figura: Test TP3-1

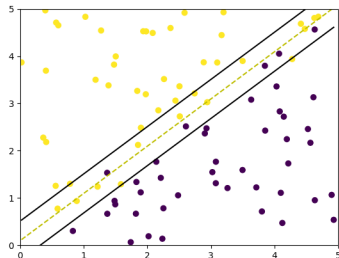


Figura: Test TP3-2

Resultados: precisión sobre testing set

- ▶ Mismos parámetros anteriores.
- ▶ 10 experimentos ; error = std.

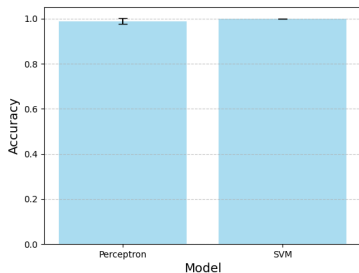


Figura: Precisión TP3-1

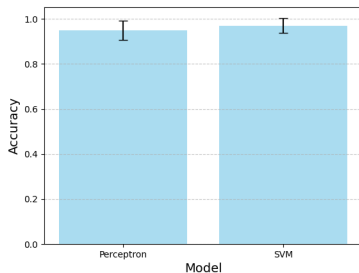


Figura: Precisión TP3-2



Conclusiones

- ▶ SVM logra una mayor precisión que el perceptrón escalón.
- ▶ Para el dataset linealmente separable, la diferencia de precisión es ínfima.

Ejercicio 1
○○○○○
○○○○
○○○○○○○

Ejercicio 2
●○○○
○○○○
○○○○○○○
○

Ejercicio 2



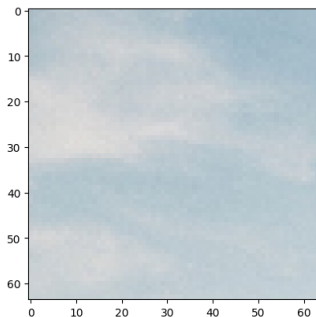
Enunciado

A partir de las imagenes dadas:

1. Crear conjunto de entrenamiento (imagenes a pixeles)
2. Dividir dataset en training y testing set.
3. Aplicar hyperparameter tuning para SVM. Agregar matrices de confusion.
4. Colorear *cow.jpg* usando SVM y 3 colores.
5. Clasificar todos los pixeles de *cow.jpg*
6. Clasificar todos los pixeles de otra imagen

Dataset

- ▶ Tomamos cada imagen en formato RGB
- ▶ Hacemos un resize de la imagen
- ▶ Pasamos la imagen a un vector donde cada elemento es de la forma (R, G, B)



Dataset

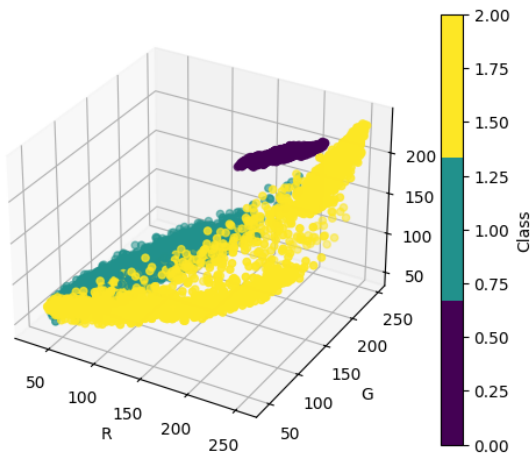


Figura: Visualización de los datos



SVM: kernels

- ▶ Nos permitirán crear clasificadores no lineales.

Kernel polinómico:

$$poly(x_i, x_j) = (x_i \cdot x_j + r)^d$$

Kernel radial:

$$rbf(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$



SVM: kernels

- ▶ El kernel radial transforma los datos a un espacio de dimensión infinita, mientras que el polinómico los transforma a uno de dimension específica dada por el grado del polinomio.
- ▶ El kernel radial es más flexible y puede manejar patrones más complejos.



Búsqueda de hiperparámetros

- ▶ GridSearchCV (*scoring = accuracy*, *cv = 5*)

kernel *poly*

degree [2, 3, 4]

C [0,1, 1, 10, 100]

kernel *rbf*

gamma [10^{-5} , 10^{-4} , 10^{-3} , 10^{-2}]

C [0,1, 1, 10, 100]



Parámetros óptimos

- ▶ 28 combinaciones analizadas (5 folds por cada una).

kernel *rbf*

gamma 10^{-4}

C 10,0

- ▶ $accuracy = 0,9901322482197354$.

Matriz de confusión

Kernel: poly, degree: 2, C: 1.0

vaca	844	0	0
cielo	0	791	19
pasto	0	8	796
	vaca	cielo	pasto

Kernel: poly, degree: 3, C: 1.0

vaca	844	0	0
cielo	0	790	20
pasto	0	10	794
	vaca	cielo	pasto

Kernel: poly, degree: 4, C: 1.0

vaca	844	0	0
cielo	0	785	25
pasto	0	10	794
	vaca	cielo	pasto

Kernel: rbf, gamma: 1e-05, C: 1.0

vaca	844	0	0
cielo	0	793	17
pasto	2	16	786
	vaca	cielo	pasto

Kernel: rbf, gamma: 0.0001, C: 1.0

vaca	844	0	0
cielo	0	795	15
pasto	0	17	787
	vaca	cielo	pasto

Kernel: rbf, gamma: 0.001, C: 1.0

vaca	844	0	0
cielo	0	800	10
pasto	0	15	789
	vaca	cielo	pasto

Kernel: poly, degree: 2, C: 10.0

vaca	844	0	0
cielo	0	796	14
pasto	0	14	790
	vaca	cielo	pasto

Kernel: poly, degree: 3, C: 10.0

vaca	844	0	0
cielo	0	793	17
pasto	0	12	792
	vaca	cielo	pasto

Kernel: poly, degree: 4, C: 10.0

vaca	844	0	0
cielo	0	792	18
pasto	0	11	793
	vaca	cielo	pasto

Kernel: rbf, gamma: 1e-05, C: 10.0

vaca	844	0	0
cielo	0	795	15
pasto	1	16	787
	vaca	cielo	pasto

Kernel: rbf, gamma: 0.0001, C: 10.0

vaca	844	0	0
cielo	0	799	11
pasto	0	14	790
	vaca	cielo	pasto

Kernel: rbf, gamma: 0.001, C: 10.0

vaca	844	0	0
cielo	0	799	11
pasto	0	17	787
	vaca	cielo	pasto



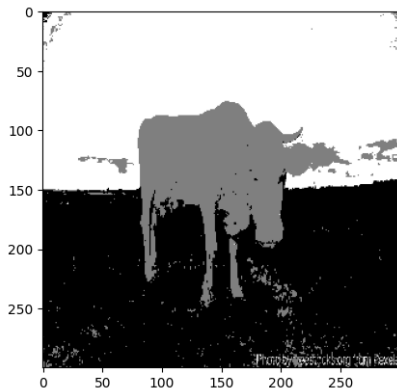
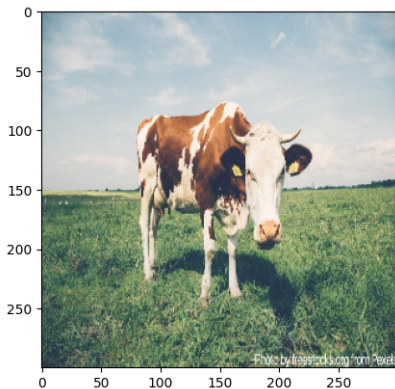
Colores

Cielo Blanco (255, 255, 255)

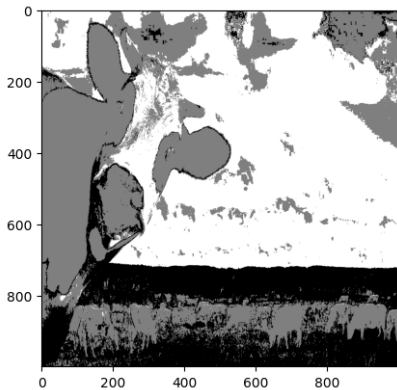
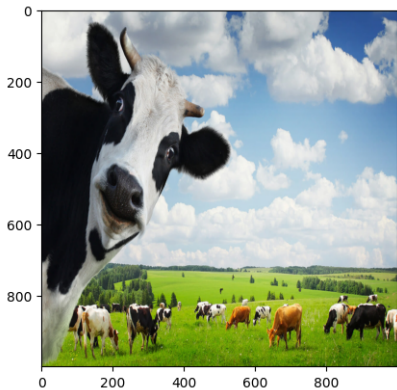
Pasto Negro (0, 0, 0)

Vaca Gris (127, 127, 127)

Resultados



Resultados





Conclusiones

- ▶ No se apreciaron grandes cambios en la precisión del modelo al variar los hiperparámetros en los rangos propuestos.
- ▶ La precisión del modelo sobre una imagen nueva no fue del todo buena.



Reflexiones

- ▶ Como el conjunto de entrenamiento era escaso (3 fotos), podriamos haber hecho data augmentation (rotar las imagenes, zoom in, zoom out, cambiar de RGB a CMYK, cambiar brillo de la imagen, etc).
- ▶ Grid search para ajustar hiperparámetros es bastante costoso en general, pero para nuestro caso fue util por la poca cantidad de hiperparametros a probar.
- ▶ Hubiese sido interesante separar la data en training, validation y test sets.



Muchas gracias