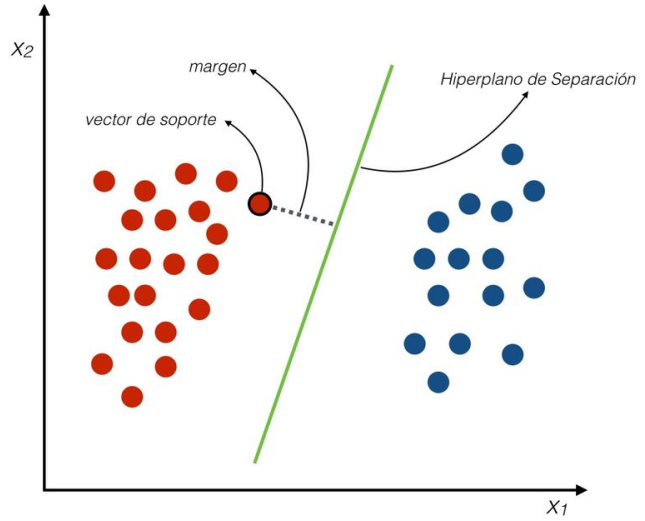
# TP3: Algoritmos de Vectores de Soporte

Dey, Patrick
Lombardi, Matías
Vázquez, Ignacio

# Tecnologías utilizadas



# Ejercicio 1:
# Perceptrón y SVM



# Generación de puntos

- Se generan N puntos (x,y) en un plano de [0,5]x[0,5] de manera aleatoria con distribución uniforme
- Para separarlos, se genera una recta a partir de 2 puntos. La misma se obtiene como:

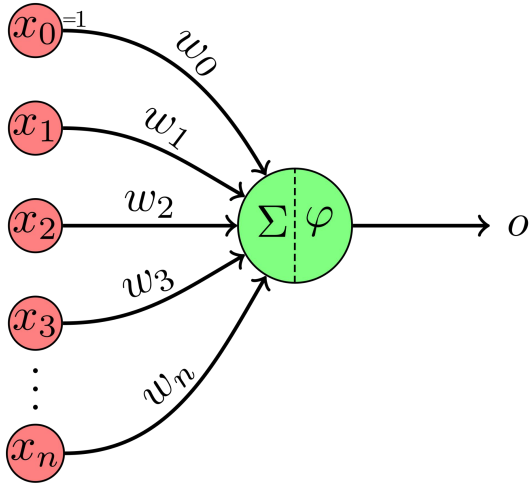
$$m = \frac{y_2 - y_1}{x_2 - x_1} \qquad b = y_1 - m \cdot x_1$$

- Luego clasificamos los puntos generados

$$1 \; si \; y_i \geq m \cdot x_i + b \qquad -1 \; si \; y_i < m \cdot x_i + b$$

- Para generar un dataset que no es linealmente separable, tomamos N puntos más cercanos a la recta separadora e invertimos su clase con una probabilidad del 50%.

# Perceptrón



# Ejercicio 1: primera parte

- Mediante el uso de un perceptrón simple escalón se desea hallar el hiperplano que permite separar un conjunto de datos linealmente separables
- Función de activación:

$$y_{pred} = sign(X_i * W_i)$$

# Implementación: Perceptrón

- Se inicializa el perceptrón con DIM pesos $w_i$ utilizando valores al azar con distribución uniforme entre -0.5 y 0.5 (en este caso DIM = 2 + 1)
- Entrenamos al perceptrón utilizando el siguiente algoritmo (se repiten n iteraciones)
  1. Tomamos un ejemplo al azar del conjunto de entrenamiento
  2. Le agregamos el sesgo 1 a dicha instancia $\xi_i = (x, y, 1)$
  3. Calculamos $\Delta w$ utilizando y actualizamos los pesos con la siguiente función

$$\Delta w = \eta \cdot (y_i - y_{pred}) \cdot \xi_i$$

  4. Calculamos el error utilizando la siguiente ecuación

$$Error = \sum_{i}^{P} (y_i - y_{pred})^2$$

# Implementación: Perceptrón

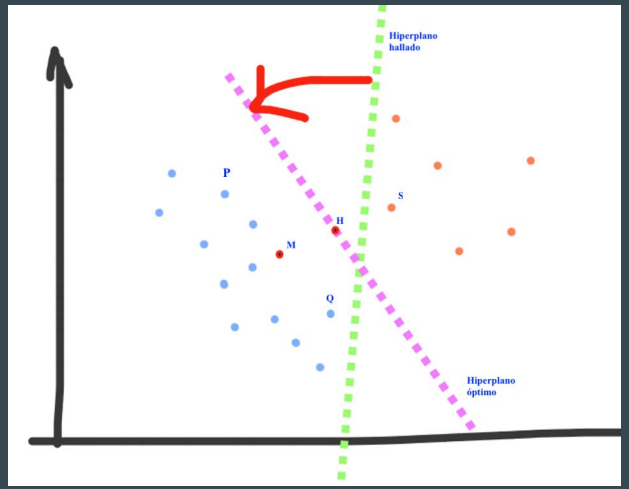
4. Calculamos el error utilizando la siguiente ecuación

$$Error = \sum_{i}^{P} (y_i - y_{pred})^2$$

5. Si el error es menor al mínimo hasta el momento, nos quedamos con estos pesos
6. Retornamos los pesos que minimizan el error

# Obtención del plano óptimo

- Una vez obtenido el hiperplano clasificador, obtenemos
  - 2 puntos más cercanos de la clase que se encuentra por debajo del hiperplano. Los llamamos P y Q.
  - El punto más cercano de la clase que se encuentra por encima del hiperplano. Lo llamamos S.
- Calculamos el punto medio entre P y Q. Lo llamamos M.
- Luego calculamos el punto medio entre M y S. Lo llamamos H
- Hallamos el hiperplano que pasa por P y Q y lo trasladamos utilizando la misma pendiente hacia H

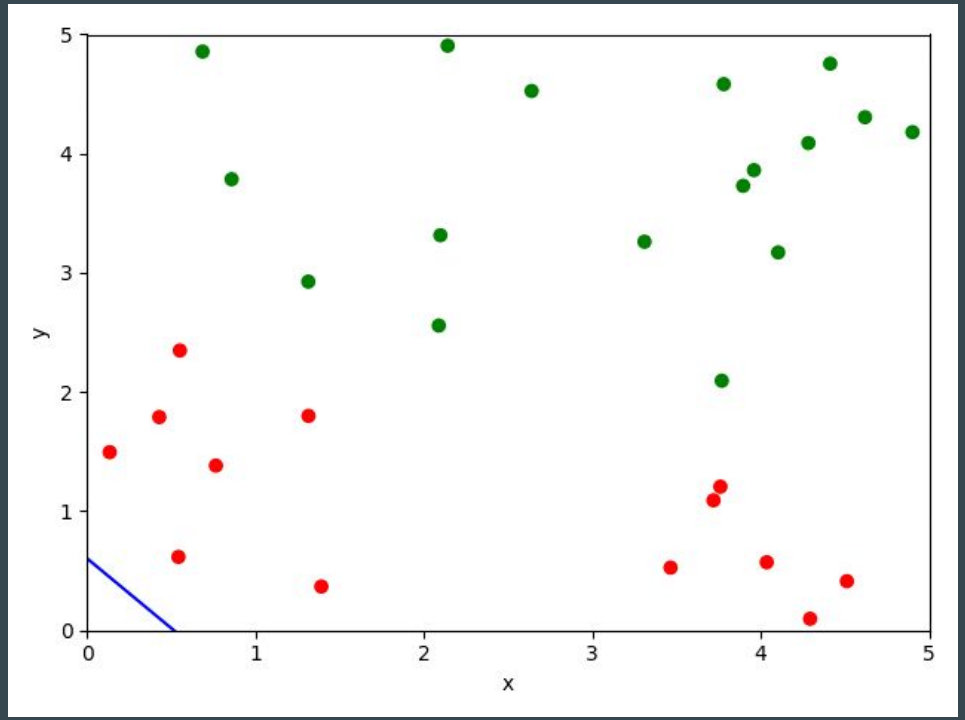


# Resultados

- Cantidad de puntos en el plano: 30, 60, 120
- Cantidad de épocas utilizadas: 1000, 10.000, 100.000
- Learning rates ($\eta$) utilizados: 0.1, 0.01, 0.001

# TP3-1: Perceptrón

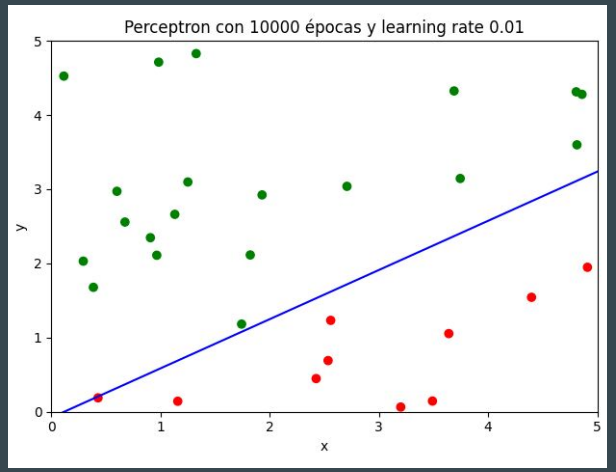
- 10.000 épocas y learning rate = 0.01, 60 puntos



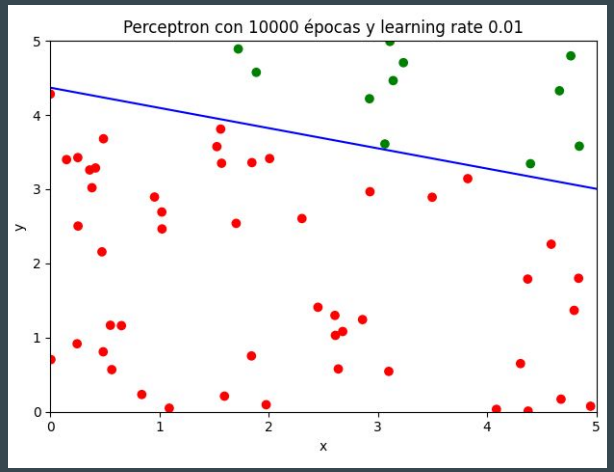
# TP3-1: Perceptrón

- 10.000 épocas y learning rate = 0.01
- Variando cantidad de puntos

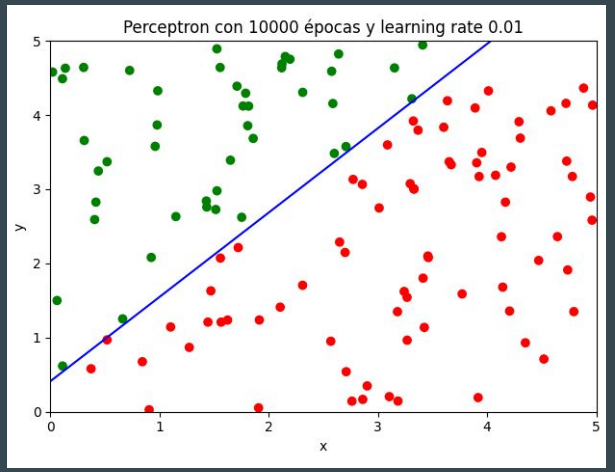
30 puntos




60 puntos



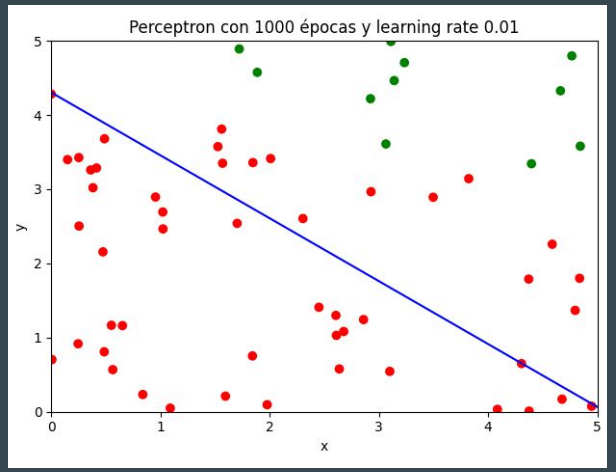

120 puntos



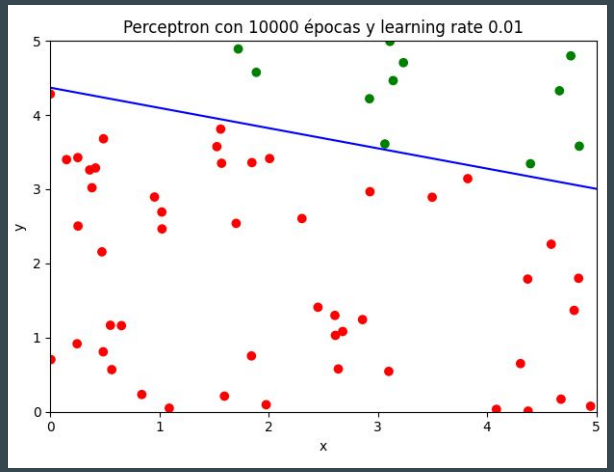
# TP3-1: Perceptrón

- 60 puntos y learning rate = 0.01
- Variando cantidad de épocas

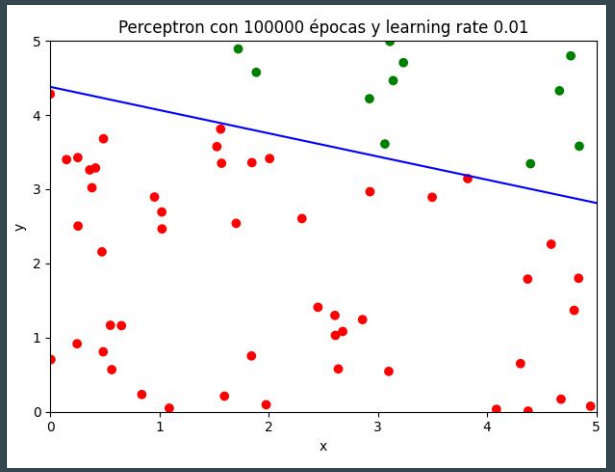
1000 épocas




10.000 épocas



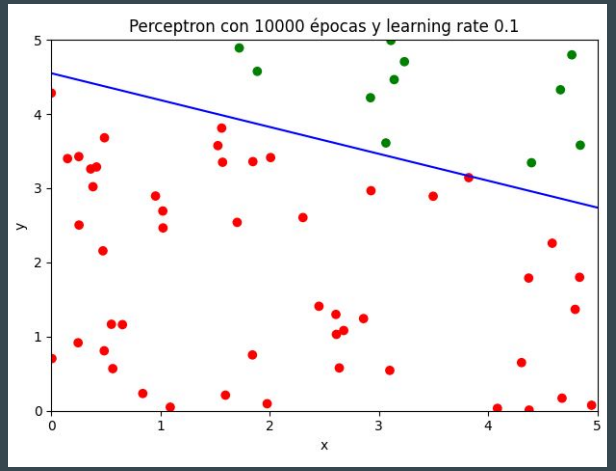

100.000 épocas



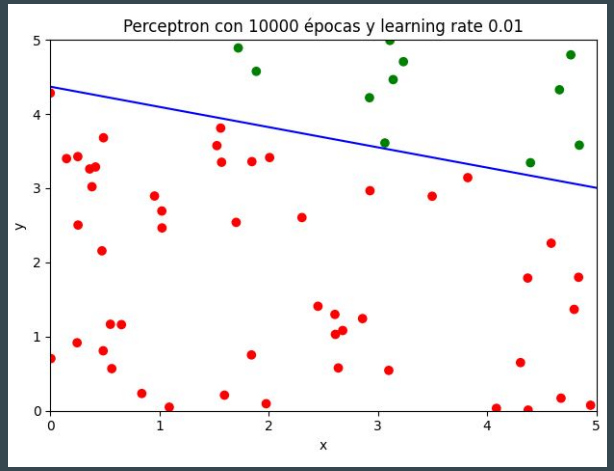
# TP3-1: Perceptrón

- 60 puntos y 10.000 épocas
- Variando learning rate (η)

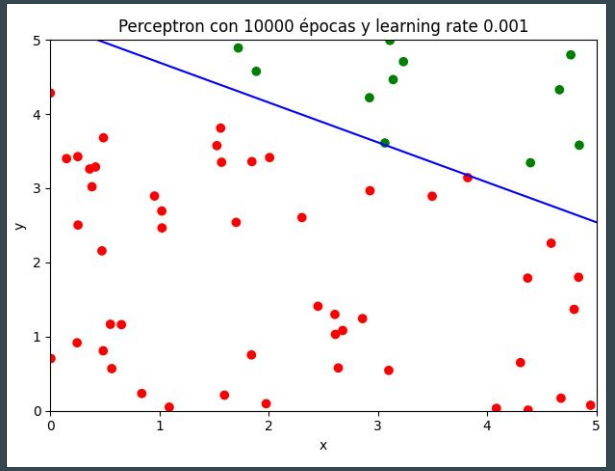
η = 0.1




η = 0.01

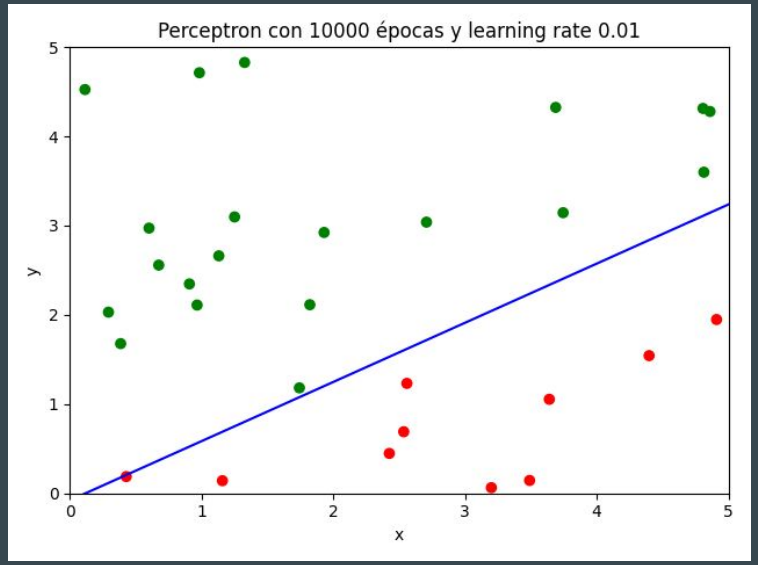


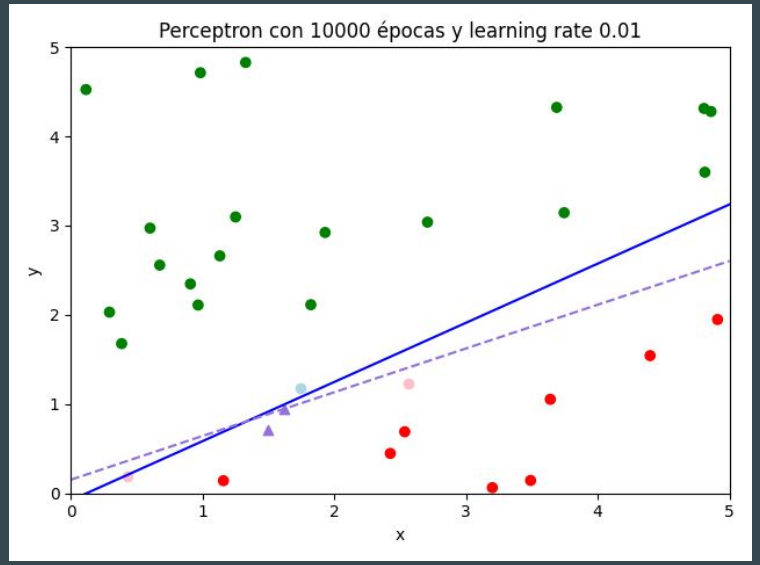

η = 0.001



# TP3-1: Hiperplano óptimo

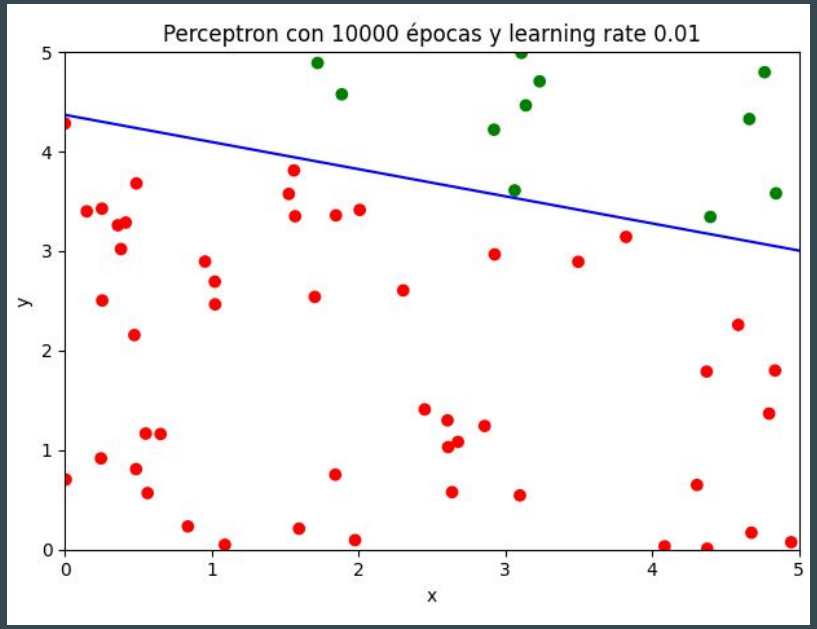
- 30 puntos y 10.000 épocas y η = 0.01

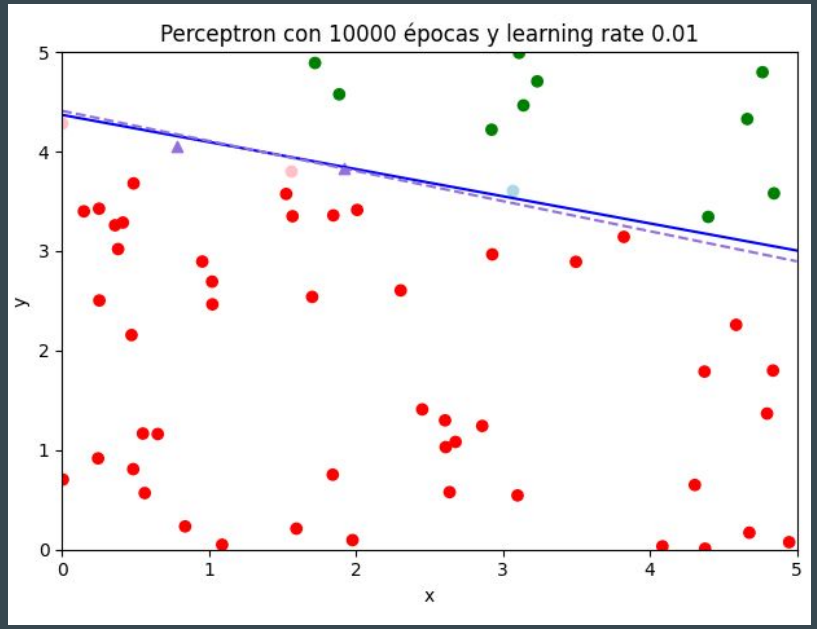


# TP3-1: Hiperplano óptimo

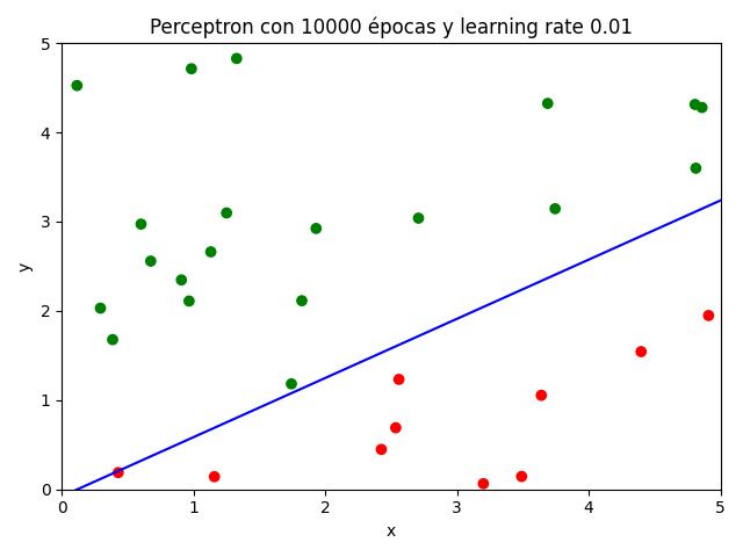
- 60 puntos y 10.000 épocas y $\eta = 0.01$

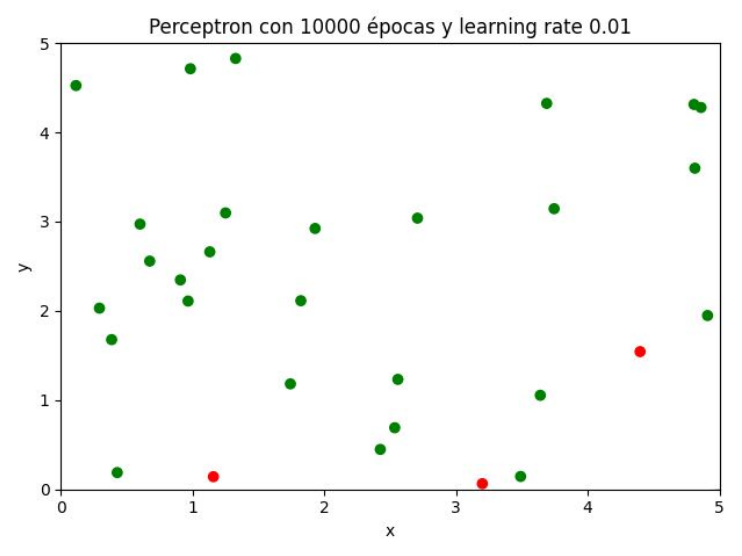


# TP3-1 vs TP3-2

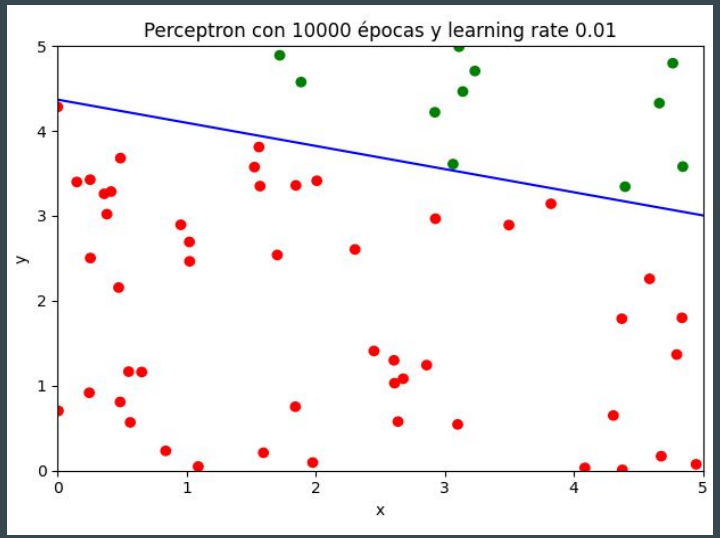
- 30 puntos y 10.000 épocas y η = 0.01

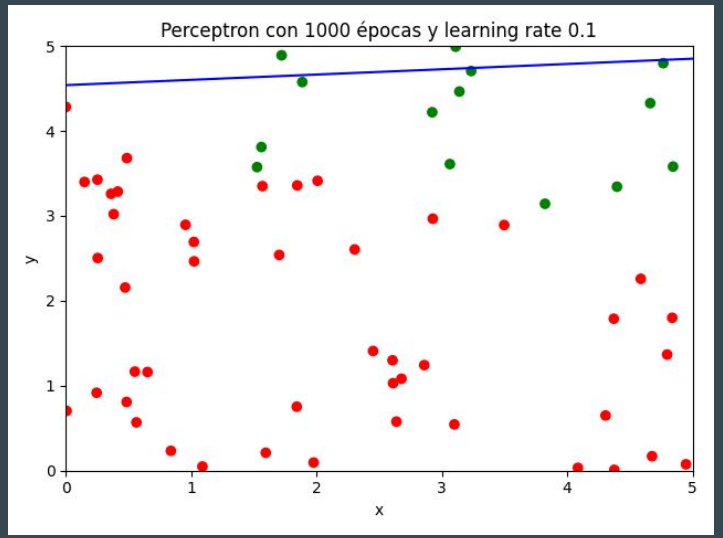


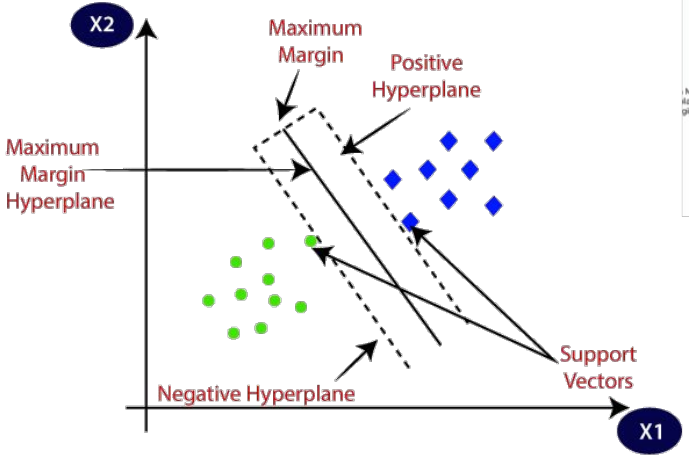
# TP3-1 vs TP3-2

- 60 puntos y 10.000 épocas y $\eta = 0.01$



# Implementación SVM



# Ejercicio 1: segunda parte

- Mediante el uso de un SVM se desea hallar el hiperplano que permite separar un conjunto de datos linealmente separables.
- Para obtener el hiperplano, se utiliza una función de pérdida, en donde se buscan minimizar los errores, sujetos a una tolerancia (C).
- Clasificación:

$$y_{pred} = sign(X_i * W_i + b)$$

# Implementación: SVM

- Se inicializa el SVM con DIM pesos $w_i$ utilizando valores al azar con distribución uniforme entre -0.5 y 0.5 (en este caso DIM = 2) y el valor de b = 0.
- Entrenamos el utilizando el siguiente algoritmo (se repite N iteraciones):
  1. Tomamos una instancia al azar del conjunto de entrenamiento.
  2. Calculamos $t = y_i \cdot (X_i \cdot w_i + b)$
     a. Si t < 1

$$w = w - \eta \cdot (w - C \cdot y_i \cdot X_i)$$
$$b = b + \eta \cdot (C \cdot y_i)$$

     b. Si t ≥ 1

$$w = w - \eta \cdot w$$

# Implementación: SVM

3. Computamos el error.

$$Error = \sum_{i}^{P} (y_i - y_{pred})^2$$

4. Si el error es menor al guardado, actualizamos los mejores w, el mejor b y el error mínimo.
5. Calculamos el nuevo valor de η

$$\eta = \eta \cdot e^{-p \cdot k}, \text{ donde } k \text{ es el número de época y } p \text{ es el valor inicial de } \eta$$

# Elección del factor C

- Factor C: valor constante que controla el equilibrio entre el error y la maximización del margen.
  - Mayor C: márgenes más chicos y mejor clasificación
  - Menor C: márgenes más grandes y peor clasificación
- Se utiliza validación cruzada 90/10 para determinar el mejor factor C.
  - $C \in [0.1, 2]$ con paso 0.2.
  - Calculamos la precisión para cada caso y nos quedamos con la que mayor media nos da.

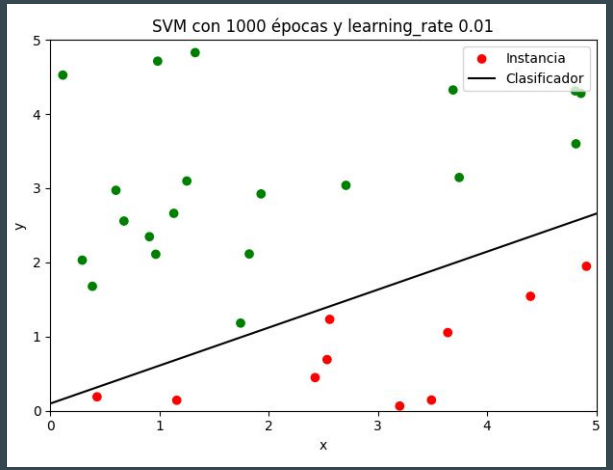
# Resultados

- Cantidad de puntos en el plano: 30, 60, 120
- Cantidad de épocas utilizadas: 100, 1.000, 10.000
- Learning rates ($\eta$) utilizados: 0.1, 0.01, 0.001
- Todos utilizan el C óptimo

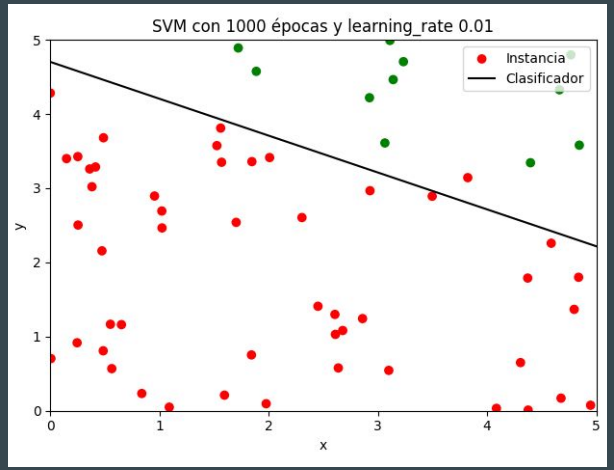
# TP3-1: SVM

- 10.000 épocas y learning rate = 0.01
- Variando cantidad de puntos

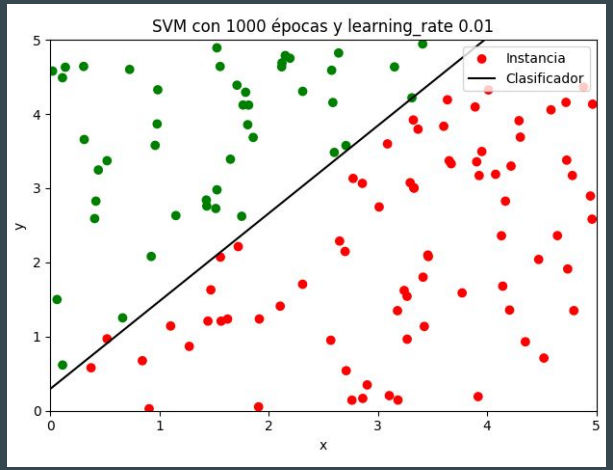
30 puntos




60 puntos



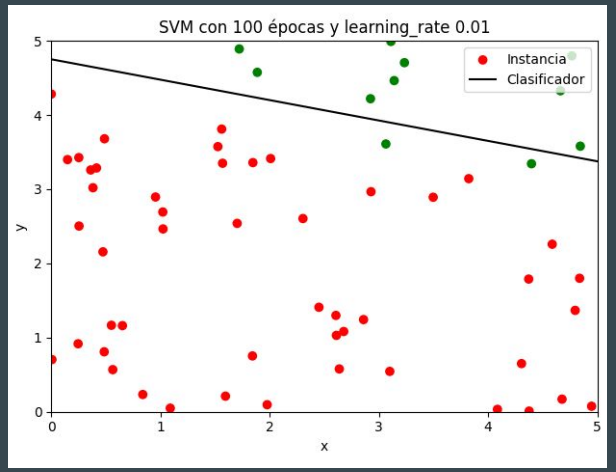

120 puntos



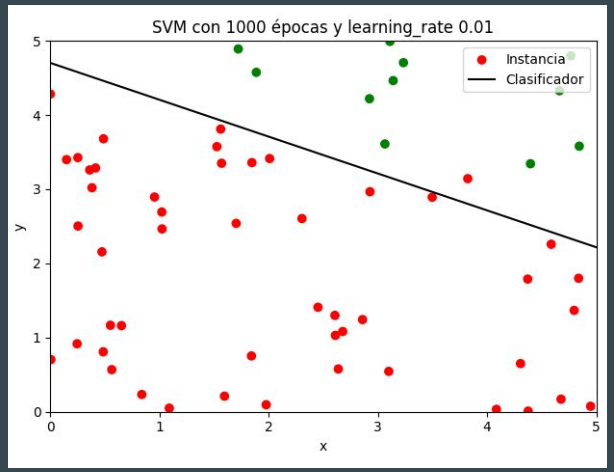
# TP3-1: SVM

- 60 puntos y learning rate = 0.01
- Variando cantidad de épocas

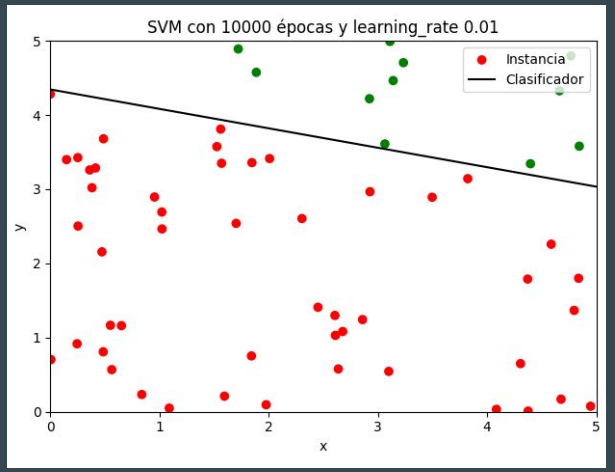
100 épocas




1.000 épocas



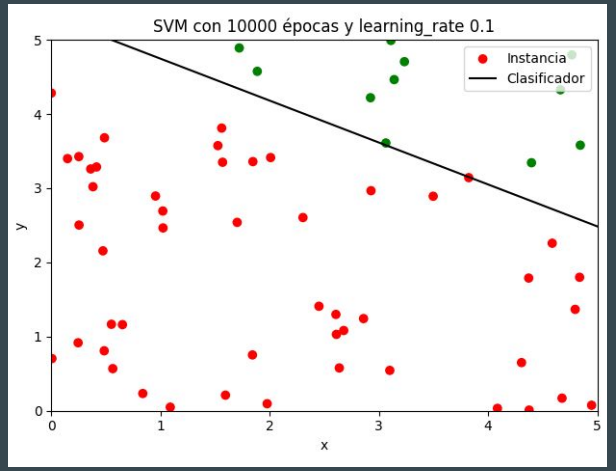

10.000 épocas



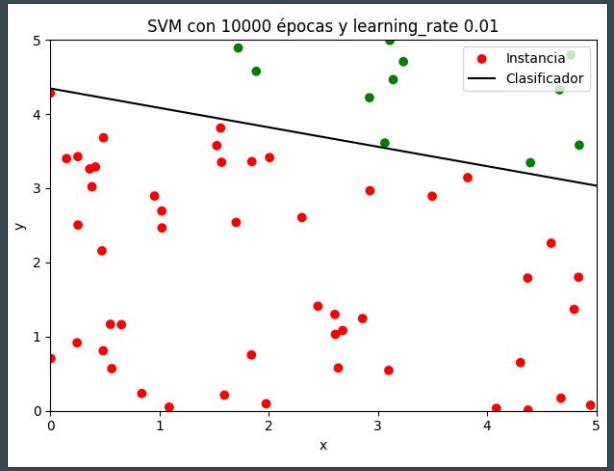
# TP3-1: SVM

- 60 puntos y 10.000 épocas
- Variando learning_rate (η)

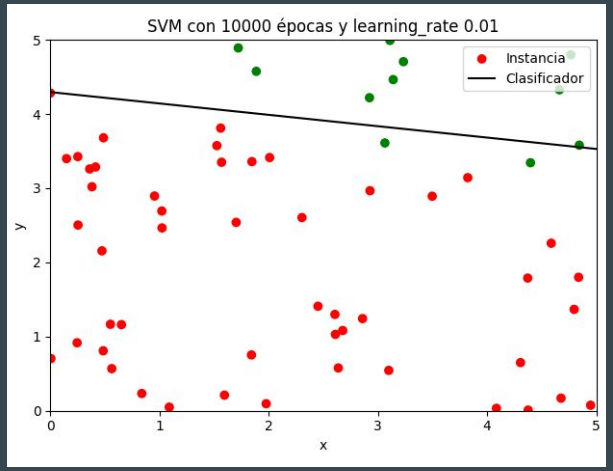
η = 0.1




η = 0.01

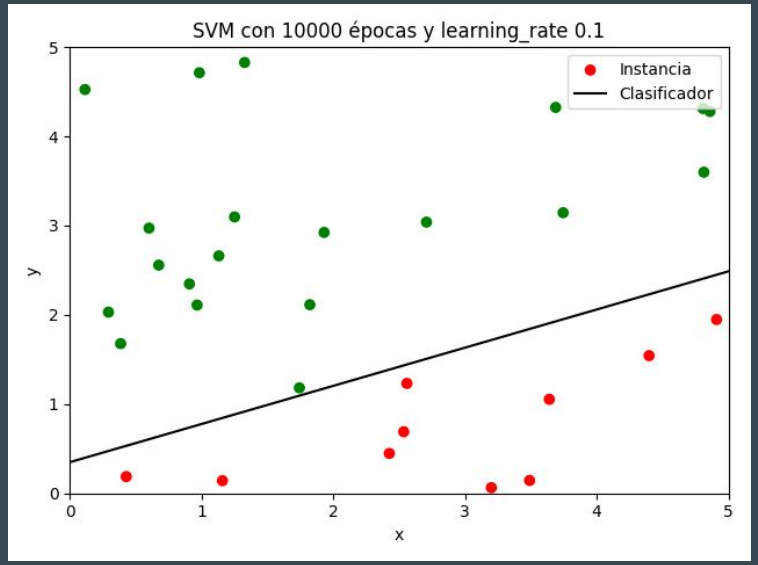


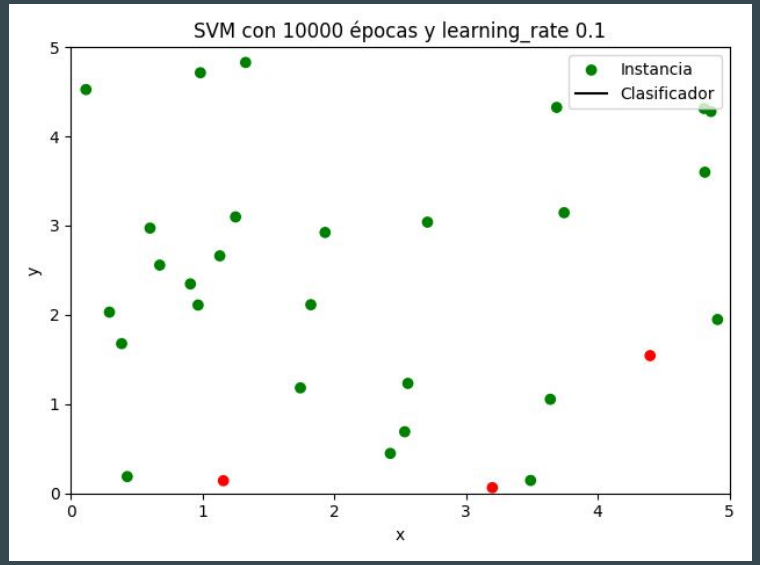

η = 0.001



# TP3-1 vs TP3-2

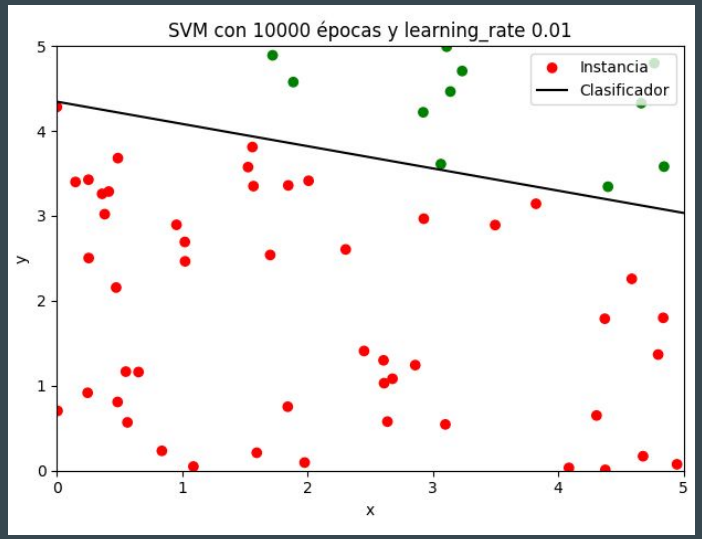
- 30 puntos y 10.000 épocas y η = 0.1

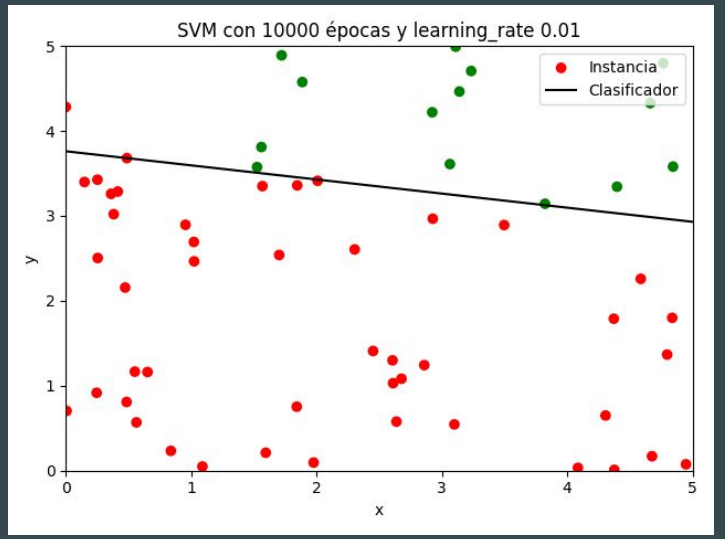


# TP3-1 vs TP3-2

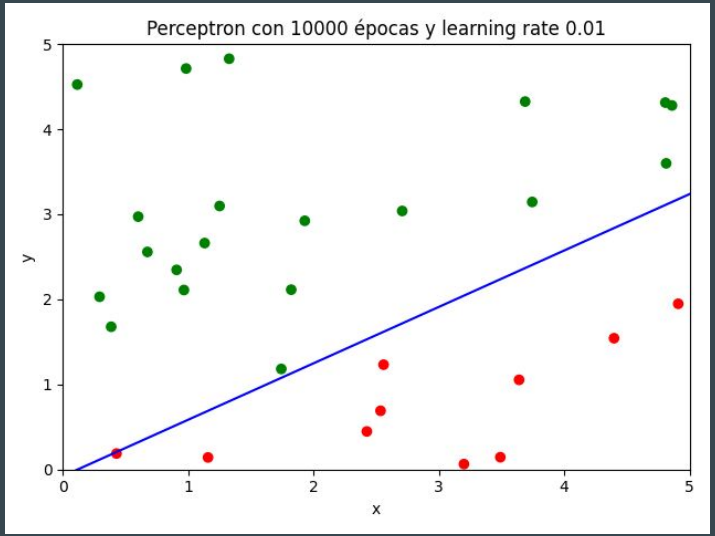
- 60 puntos y 10.000 épocas y $\eta = 0.01$



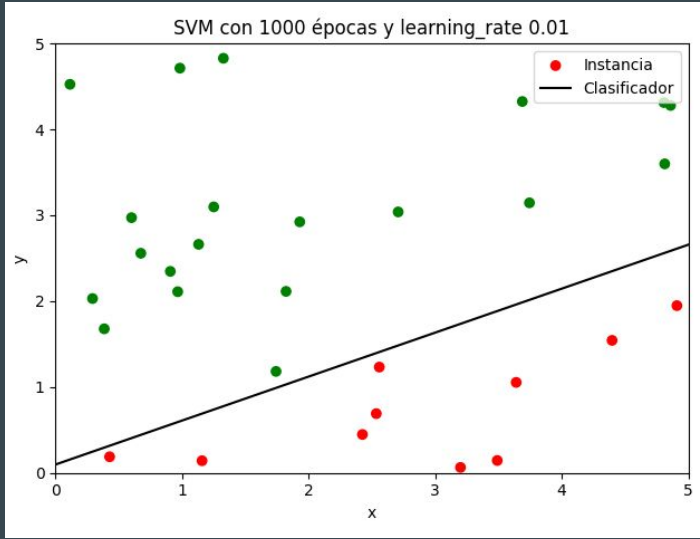



# Comparación Perceptrón vs SVM

- 30 puntos y 10.000 épocas y $\eta = 0.01$



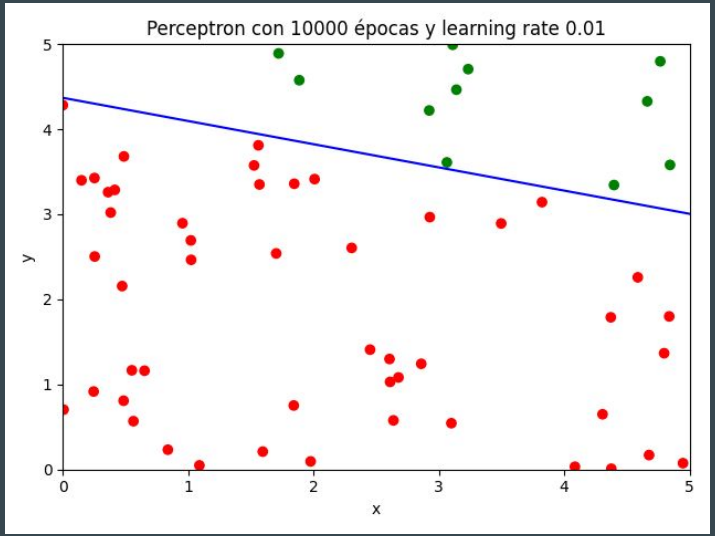

Perceptrón



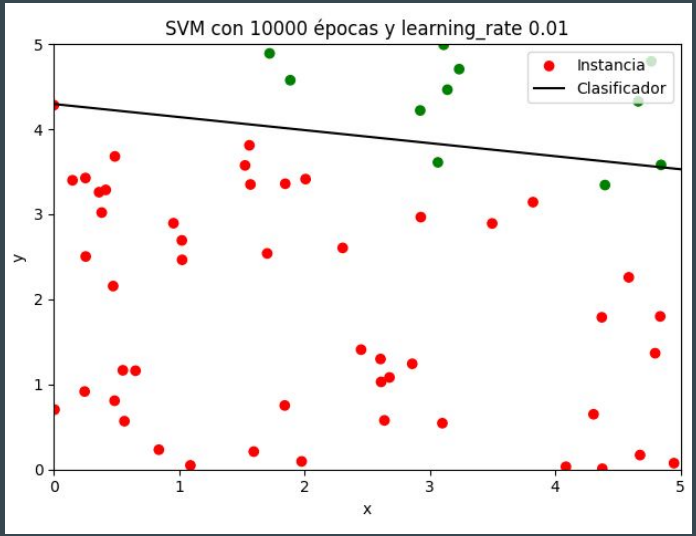

SVM

# Comparación Perceptrón vs SVM

- 60 puntos y 10.000 épocas y $\eta = 0.01$

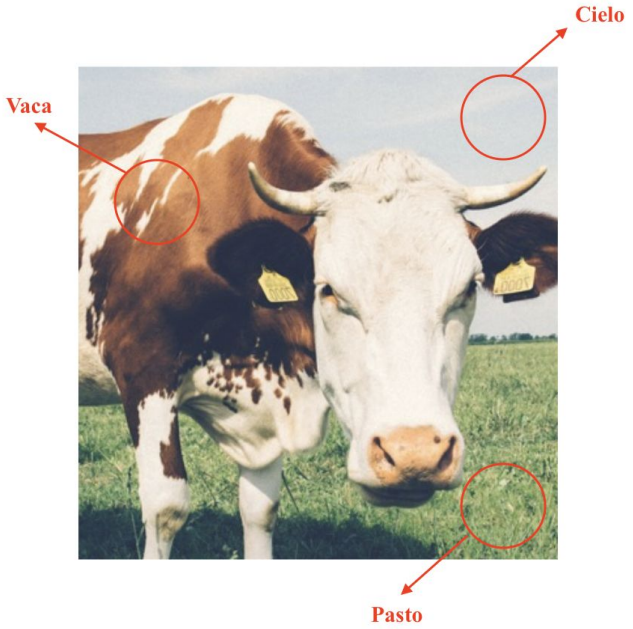



Perceptrón




SVM

# Ejercicio 2: SVM



# Ejercicio 2

- En base un conjunto de imágenes acerca de una vaca, queremos utilizar SVM para clasificar los píxeles de las mismas
- Se utilizan distintos kernels y valores de C
  - Kernel: funciones matemáticas para aumentar la dimensionalidad del input y ayudan a separar linealmente datos que pueden no serlo.
  - Factor C: valor constante que controla el equilibrio entre el error y la maximización del margen.
    - A mayor C: márgenes más chicos y mejor clasificación
    - A menor C: márgenes más grandes y peor clasificación
- Se utiliza validación cruzada 80/20

# Transformación de inputs

- Cada imagen es transformada a un conjunto de 3 matrices R, G y B con los valores para cada píxel.
- A cada píxel (de cada imagen) le asignamos un valor clasificatorio
  - Cielo = -1
  - Pasto = 0
  - Vaca = 1

Cielo

Pasto

Vaca

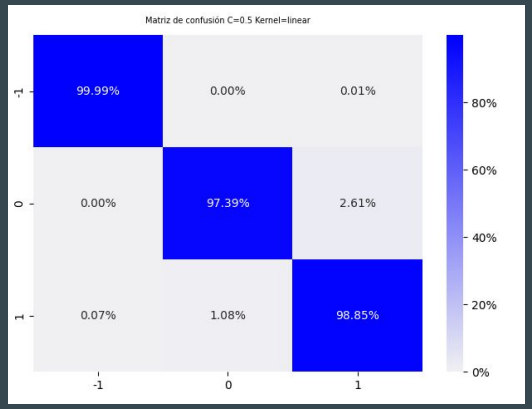
# Entrenamiento

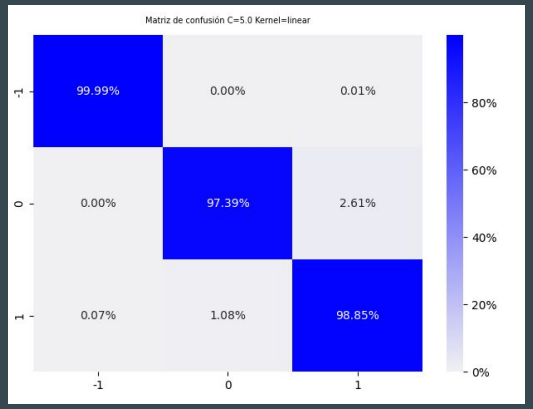
- Utilizamos sklearn.svm.SVC()
- Entrenamos el modelo utilizando distintos valores de C y distintos kernels.
- Generamos la matriz de confusión comparando los valores predichos con los valores de testeo.
- Kernels: linear, polynomial, rbf, sigmoid
- $C \in [0.2, 2]$ con paso 0.2.

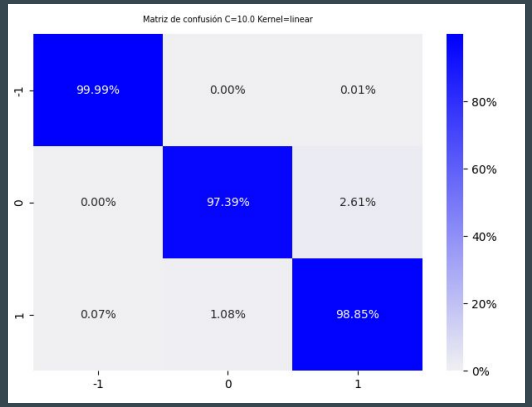
# Clasificación de nuevas imágenes

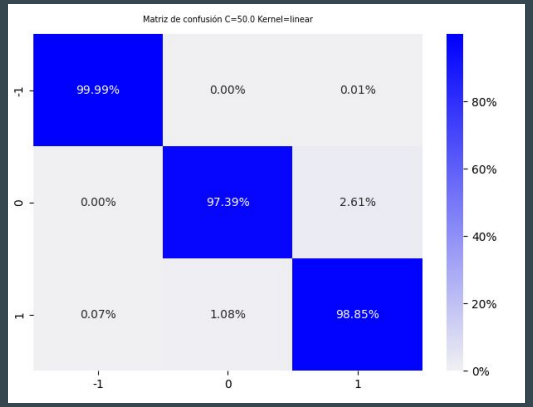
- Utilizamos sklearn.svm.SVC()
- Entrenamos el modelo utilizando distintos valores de C y distintos kernels.
- Una vez entrenado el modelo le presentamos una nueva imagen y predecimos la imagen de salida con la información aprendida
- Cada clase (cielo, pasto, vaca) se representa con un color
- Kernels: linear, polynomial, rbf, sigmoid
- $C \in [0.2, 2]$ con paso 0.2.

# Kernel lineal

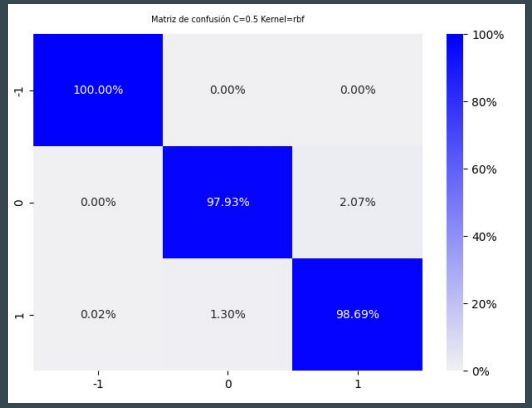


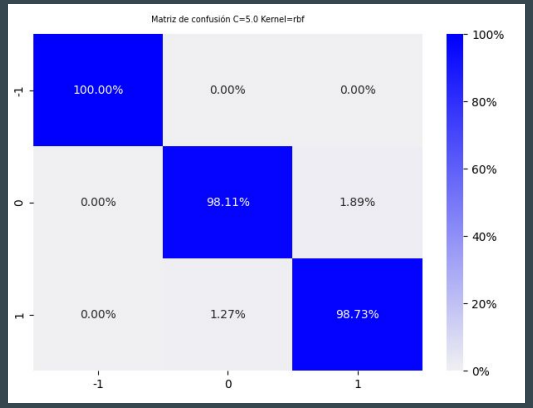



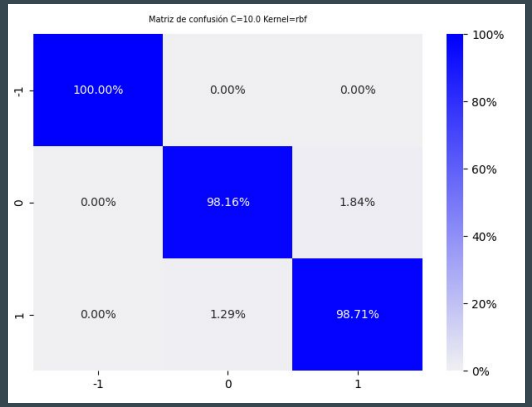



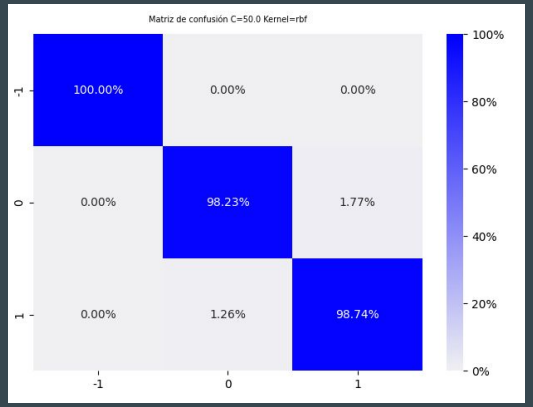



# Kernel Radial Basis Function (RBF)

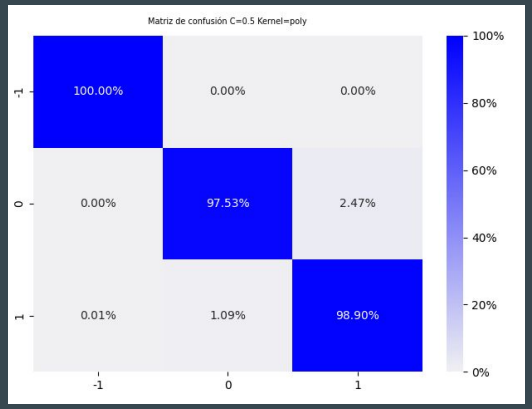


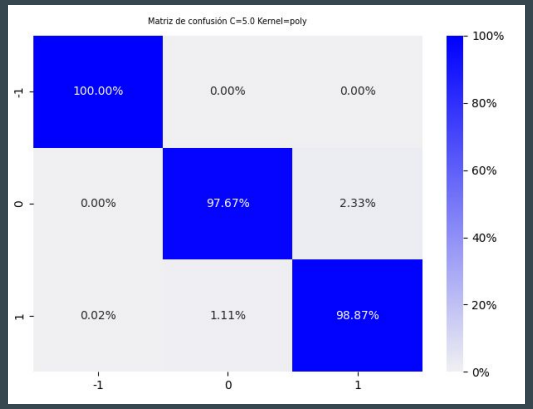



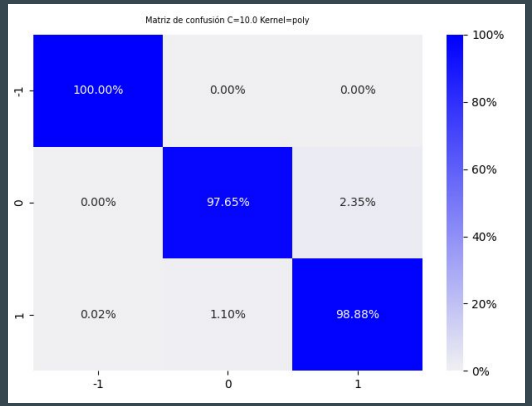



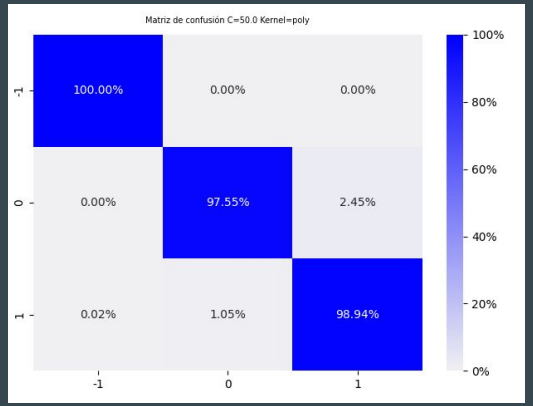



# Kernel Polinómico

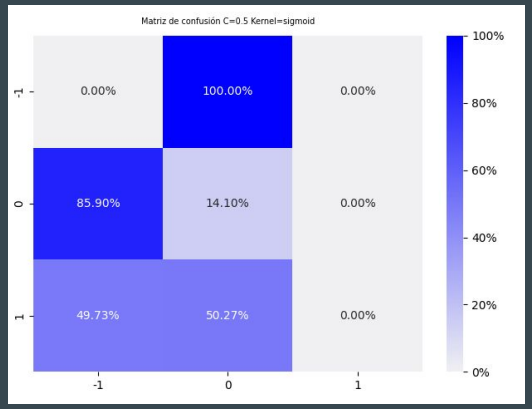


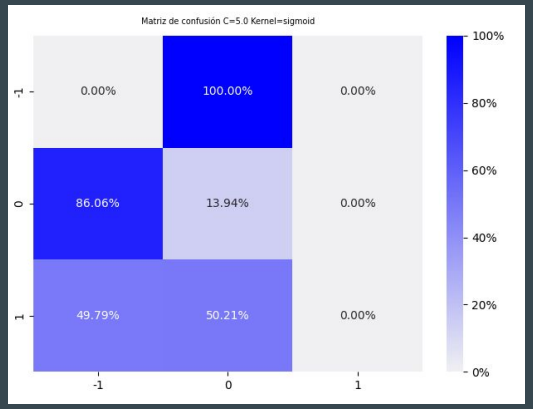



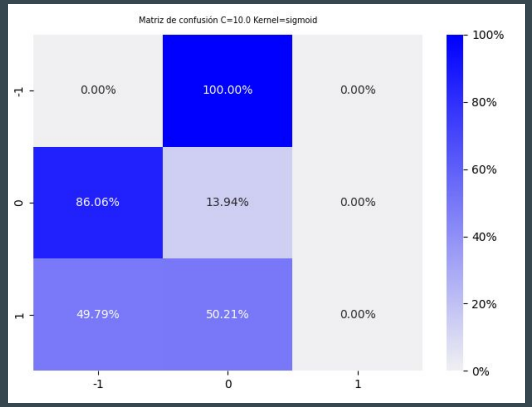



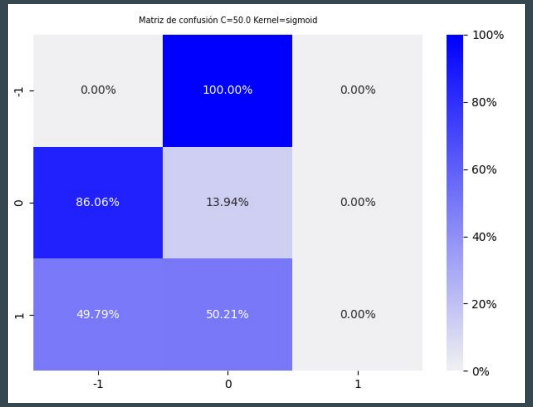



# Kernel Sigmoid



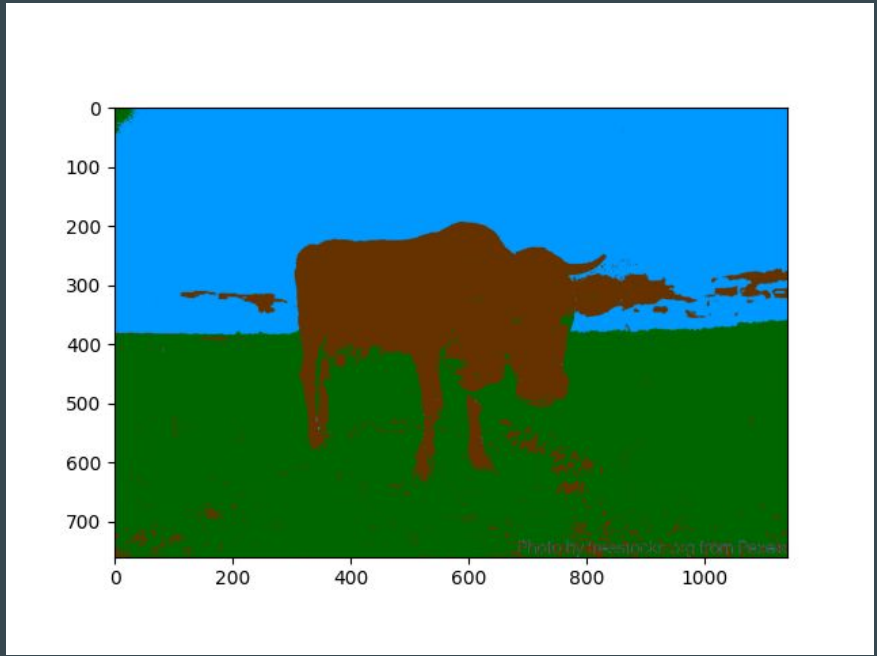



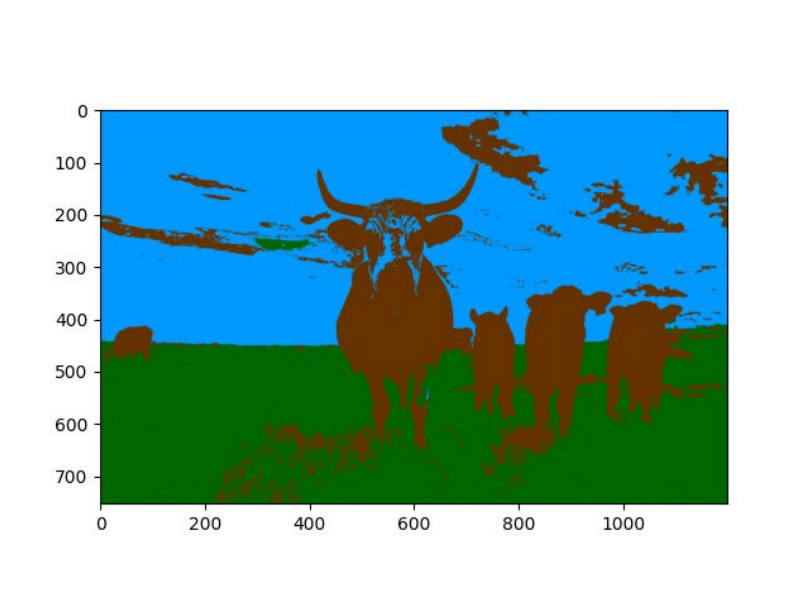
# Clasificación con parámetros óptimos

- Luego de obtener las matrices de confusión, clasificamos las imágenes con los parámetros óptimos
- Kernel: Rbf
- C: 1.6



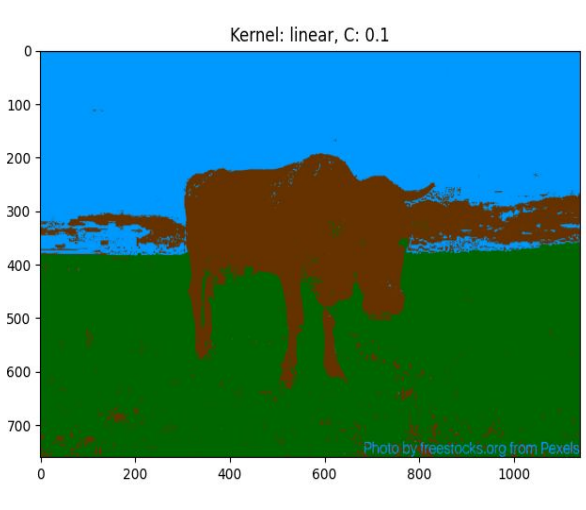
# Clasificación de otra imagen

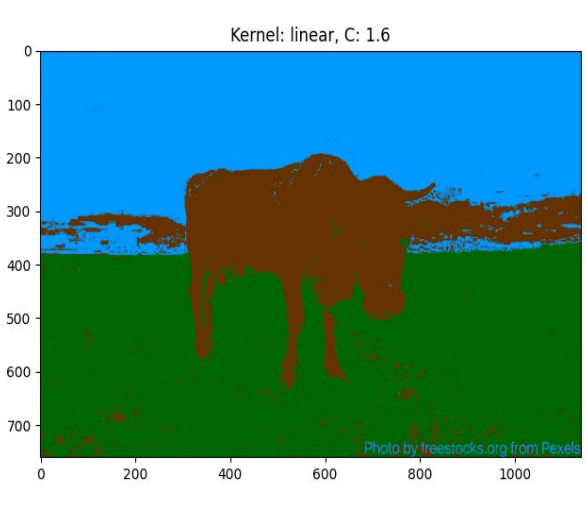
- Clasificamos la otra imagen con los mismos parámetros.

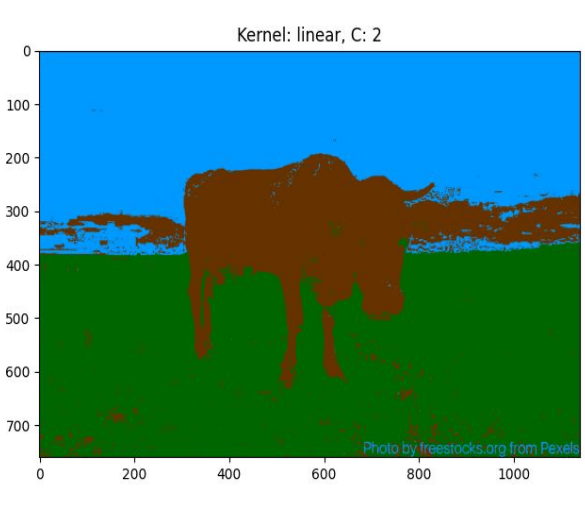


# Clasificación para distintos Kernels

## Kernel Lineal

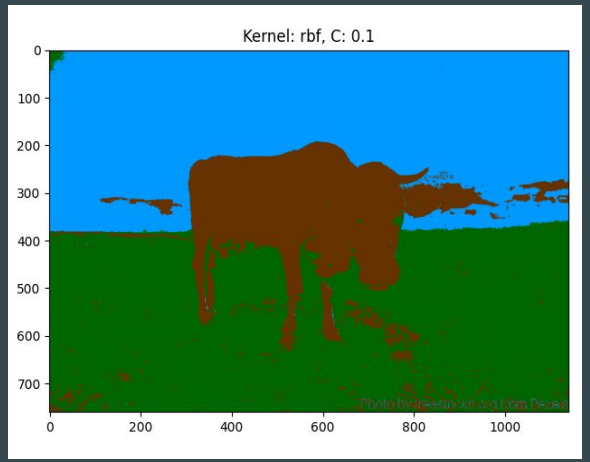


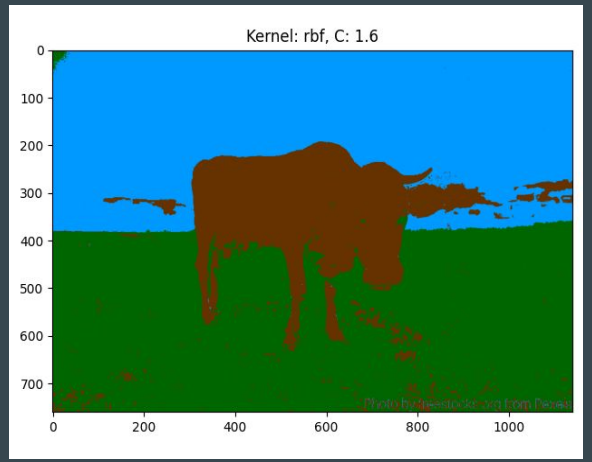



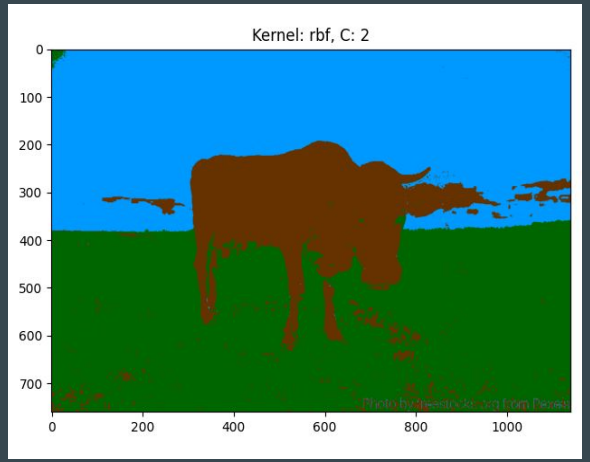



# Clasificación para distintos Kernels

## Kernel RBF

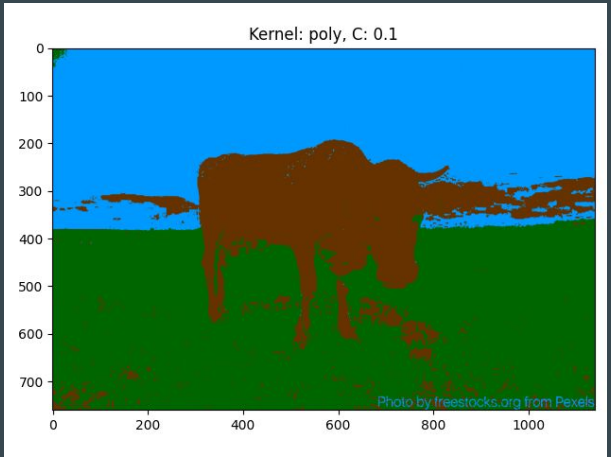


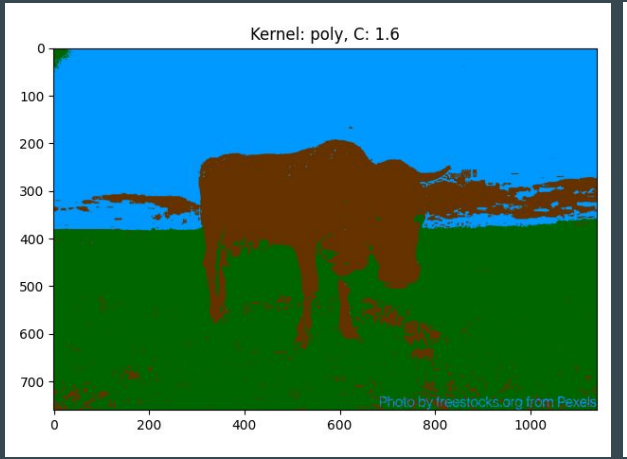



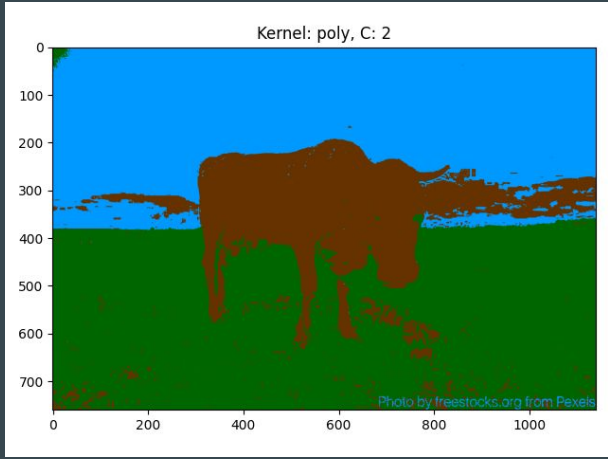



# Clasificación para distintos Kernels

## Kernel Polinomico

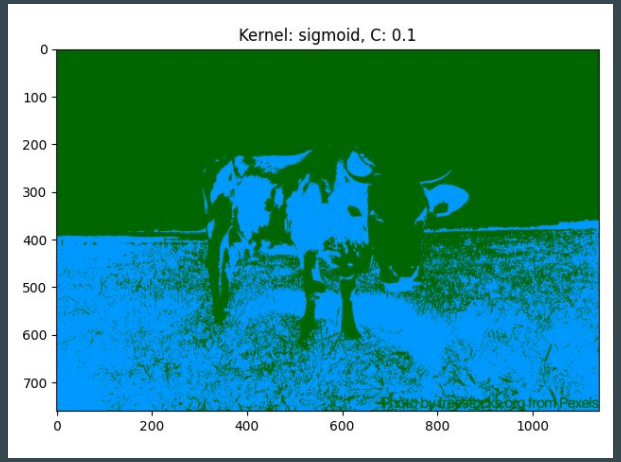


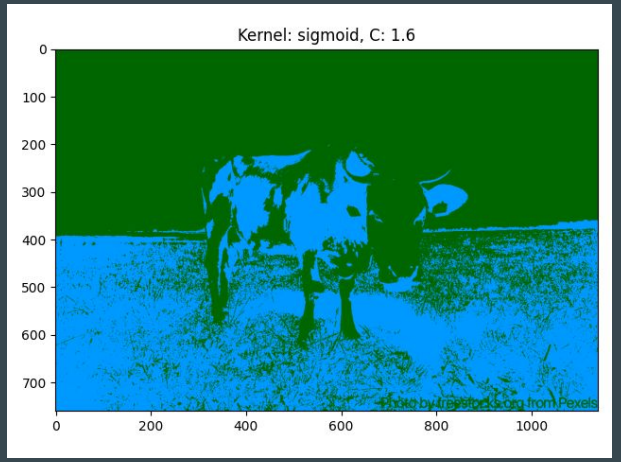



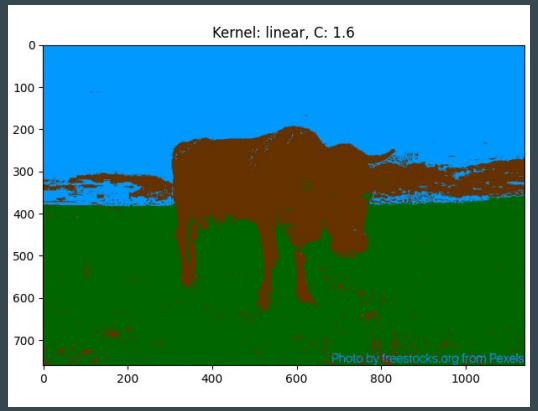
# Clasificación para distintos Kernels

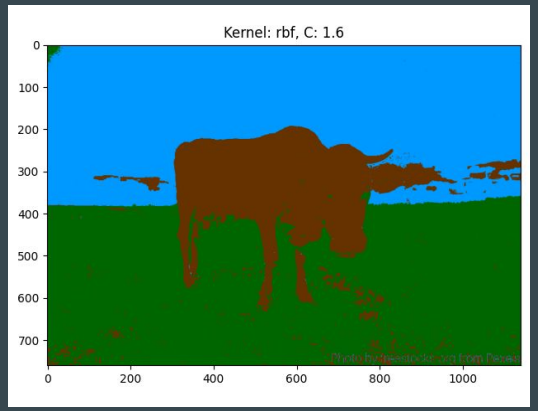
## Kernel Sigmoid

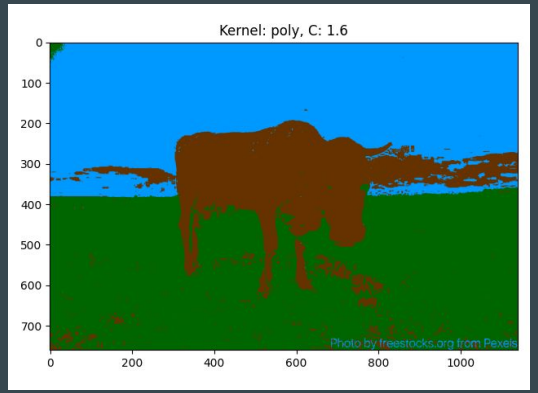


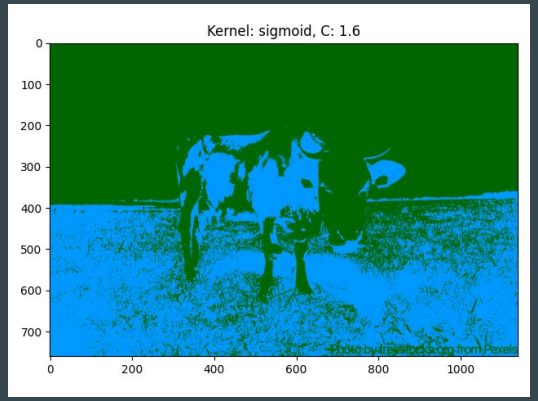



# Comparativa kernels, C = 1.6



# ¡Muchas gracias!

Dey, Patrick
Lombardi, Matías
Vázquez, Ignacio