



INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE:

MACHINE LEARNING

APUNTES

ALUMNO:

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio



INSTITUTO POLITÉCNICO NACIONAL



FECHA DE ENTREGA: 15/01/2025

Tercer Parcial

Algoritmo K-means

Algoritmo de Clustering (De tipo supervisado). Dado un conjunto de puntos, se encarga de organizarlos en K clusters. El método se caracteriza por calcular centroides y evaluar la distancia que hay entre los puntos y cada K -centroide (un centroide por cada cluster deseado). A la muestra se le asocia el K -ésimo centroide más cercano.

Algoritmo básico:

- 1.- Determinar K (se recomienda hacer inspección visual y determinarlo con base a experiencia).
- 2.- Inicializar centroides (X_i, Y_i) de forma aleatoria donde $i \in [1, 2, 3, \dots, K]$.
- 3.- Calcular la distancia de la muestra X_j al centroide C_i . Asignar el punto X_j a la clase K como aquel que menor distancia tiene.
- 4.- Recalcular centroide con las muestras asignadas a la clase.

Desventajas:

- Sensible a la inicialización de centroides o los puntos medios. Por lo tanto, si un centroide es inicializado para que sea un punto lejano, podría terminar sin puntos asociados y, al mismo tiempo, más de un grupo podría terminar incluido con un solo centroide.
- De manera similar, más de un centroide podría inicializarse en el mismo grupo, lo que daría como resultado una agrupación de Ficiente.

Máquinas de soporte vectorial

1.- Tratan de encontrar un hiperplano vía un kernel (En su forma básica una línea) a través de márgenes.

2.- Son clasificadores

La intención SVM (Support Vector Machine) es maximizar la distancia que hay entre los márgenes y los vectores de soporte. Estos últimos son aquellos vectores que se generan del hiperplano $Wx + b$ a los puntos más cercanos respecto al margen.

Este algoritmo presenta ventajas respecto a la regresión lineal o KNN porque solamente considera los puntos cercanos al margen y no todos los puntos como los algoritmos mencionados.

El margen $M = \frac{2}{\|W\|}$ y por tanto los SVM se transforman en

un problema de minimización de W con respecto a los vectores de soporte. Lo anterior hace uso de multiplicadores de Lagrange. Además, el problema se basa en que :

$$y_i (W \cdot x_i + b) \geq 1 \rightarrow \text{Class } 1$$

entonces el punto x_i ... está bien clasificado y por tanto los parámetros no requieren cambios. Si

$$y_i (W \cdot x_i + b) < 1$$

entonces el punto no está bien clasificado y se requiere mover el plano. Conjugando lo anterior, con los pesos optimizados ya aplicando los multiplicadores de Lagrange se tiene:

$$\text{Si } y_i (W \cdot x_i + b) \leq 1$$

$$W = W - \eta \cdot (2 \lambda W - y_i x_i)$$

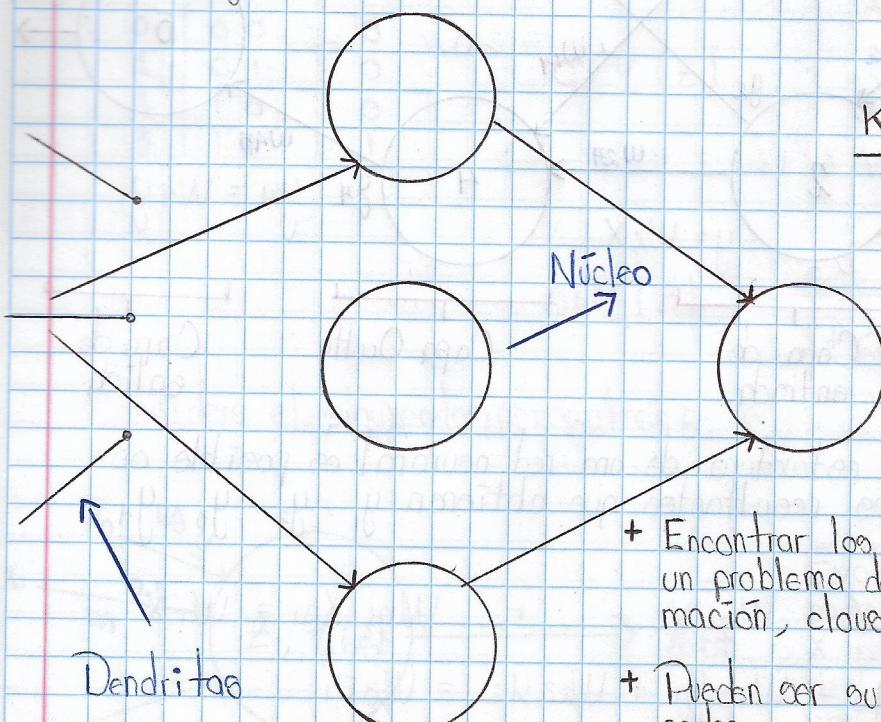
$$b = b + \eta \cdot \lambda \cdot y_i$$

Donde: η = Tasa de aprendizaje W = Pesos del hiperplano

λ = Constante de corrección.

Puedes Neuronales

Pueden imitar el comportamiento de una neurona real al procesar información y transmitirla.



- + Encontrar los pesos W para resolver un problema de clasificación, estimación, clustering, entre otros.

- + Pueden ser supervisadas o no supervisadas.

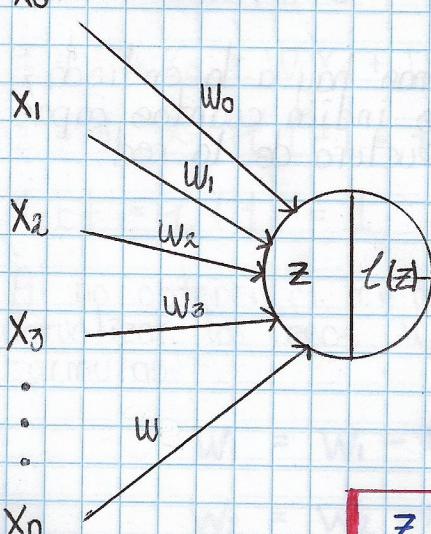
Valor de activación

$$z = W \cdot X \quad \text{Donde:}$$

$$W = [w_0, w_1, w_2, w_3, \dots, w_n]$$

$$X = [x_0, x_1, x_2, x_3, \dots, x_n]$$

y obt \downarrow Bias



Generar una cantidad de hiperplanos tal que puedan delimitar diferentes zonas

Función Universal

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

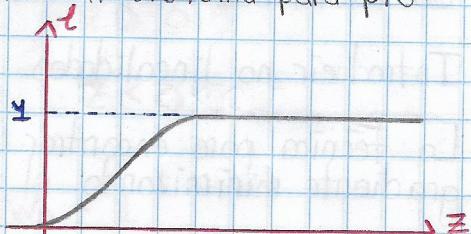
Requieren de varios datos de entrenamiento.

* Para solucionar problemas no lineales con redes neuronales se hace uso de las funciones de activación pasando el argumento z .

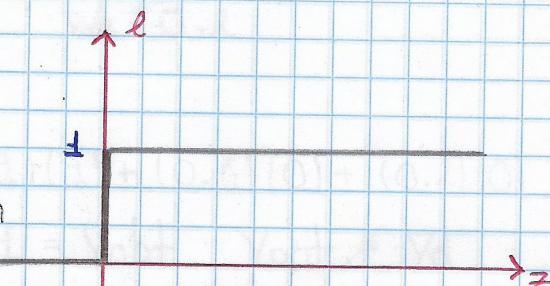
* Sean ponderar características relevantes al sistema para producir una salida adecuada.

Ejemplos:

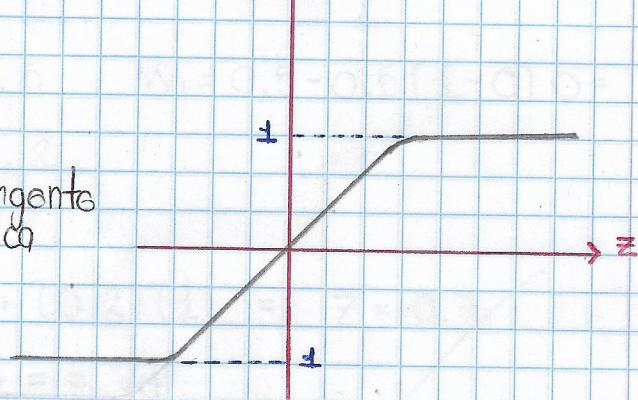
$$* \ell(z) = \frac{1}{1+e^{-z}} \quad \text{Función Sigmoidal}$$



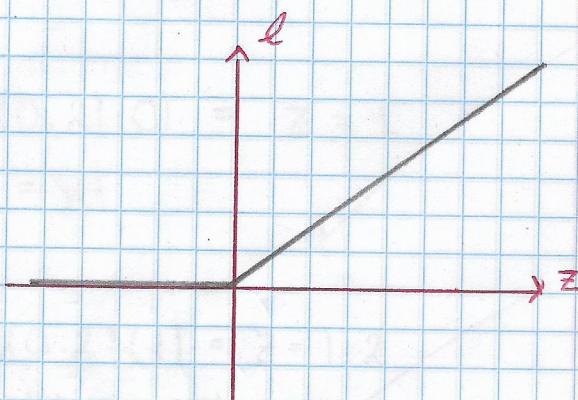
$$\circ \ell(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \quad \text{Función Escalón Unitario}$$



$$* \ell(z) = \tanh(z) \quad \text{Función tangente hiperbólica}$$



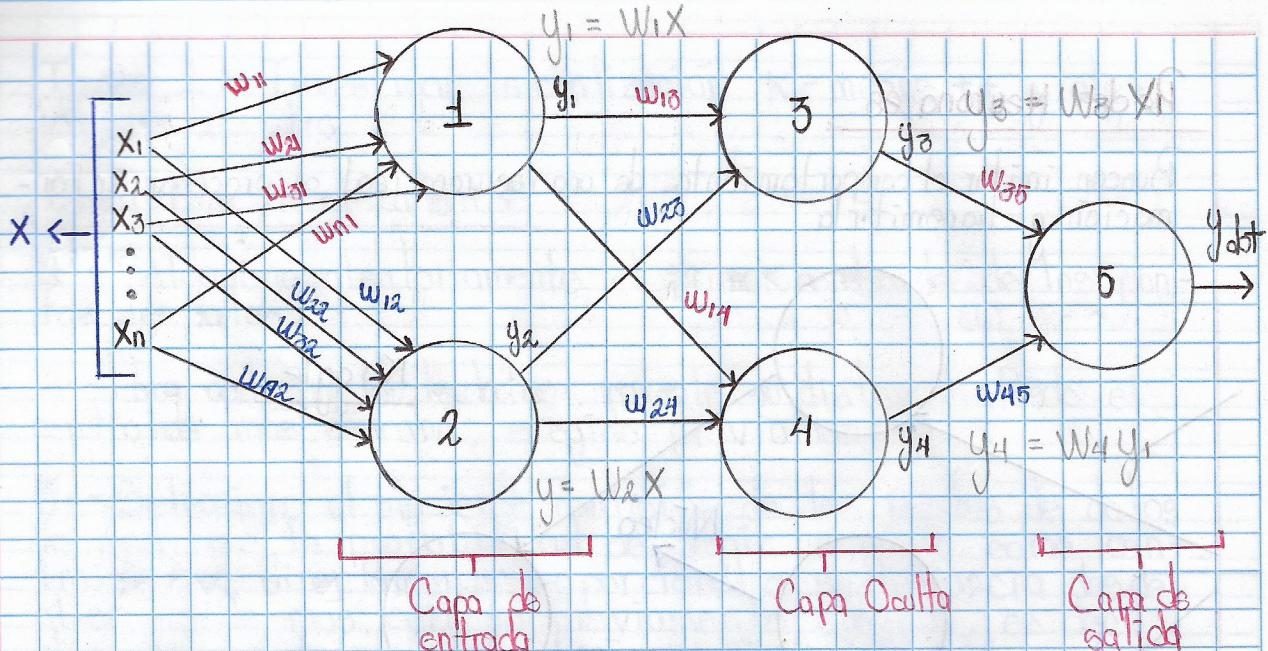
$$\ell(z) = \max(0, z) \quad \text{ReLU}$$



$$y_{obt} = \ell(z)$$

- * El objetivo de las funciones de activación es cambiar el dominio de las características.
- * Se recomienda escoger dominios semejantes
- * Introducir no linearidades

La técnica para encontrar los pesos más común es descenso del gradiente minimizando el ECU como función de costo.



Considerando esta estructura de una red neuronal es posible obtener las ecuaciones resultantes que obtienen y_1, y_2, y_3, y_4 y y_{out} .

$$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + \dots + w_{1n}x_n = w_{11}x$$

$$y_2 = w_{14}y_1 \quad y_3 = w_{13}y_1 + w_{23}y_2 = w_{23}y_2$$

$$y_4 = w_{14}y_1 + w_{24}y_2 = w_{14}y_1$$

$$y_{\text{out}} = w_{35}y_3 + w_{45}y_4$$

Topología de red: Indica cuantas neuronas hay a la entrada, a la salida y en cada capa oculta. Además indica cuantas capas ocultas hay; por tanto indica la estructura de la red.

Sea la siguiente tabla de verdad

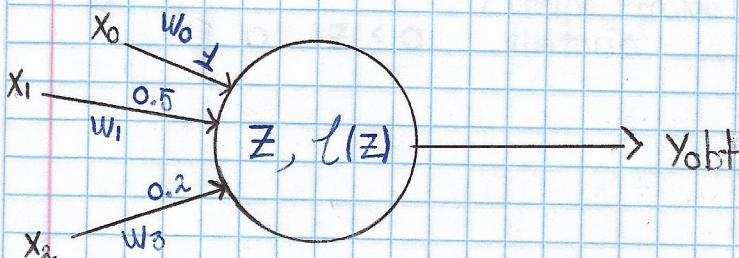
i	x_1	x_2	y_d
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

Suponga:

$$a) \ell(z) = \begin{cases} 1 & z \geq 0.5 \\ 0 & z < 0.5 \end{cases}$$

$$b) \psi(z) = \frac{1}{1+e^{-z}}$$

Considere el siguiente perceptrón:



Evalue las entradas para cada muestra del vector X y determine si los pesos son adecuados para resolver el problema.

Utilice un Learning Rate de 0.5

Para la muestra 0 (Big0).

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 = (\pm)(\pm) + (0.5)(0) + (0.2)(0)$$

$$z = \pm \approx l(z) = l(\pm) = \pm$$

$$y_{obt} = l(\pm) = \pm \quad \text{¿ } y_{obt} == y_d? \text{ No son iguales}$$

Al no obtener el mismo valor en nuestra y_{obt} es necesario actualizar los pesos w_0 , w_1 y w_2 , utilizando las siguientes fórmulas.

$$w_1 = w_1 - \eta \cdot (y_{obt} - y_{di}) \cdot x_1$$

$$w_2 = w_2 - \eta \cdot (y_{obt} - y_{di}) \cdot x_2$$

$$w_0 = w_0 - \eta \cdot (y_{obt} - y_{di})$$

Algoritmo**For** épocas :**for** i in #muestras

$$z = w_0 x_0 + w_1 x_1(i) + w_2 x_2(i)$$

$$y_{\text{obt}} = \phi(z)$$

$$\text{entrenamiento} = 0$$

if ($y_{\text{obt}} \neq y_d(i)$)

$$w_1 = w_1 - lr \cdot (y_{\text{obt}} - y_d(i)) \cdot x_1(i)$$

.

.

.

$$\text{entrenamiento} = 1$$

-if entrenamiento = 0**break**



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

UNIDAD DE APRENDIZAJE:

MACHINE LEARNING

TAREAS

ALUMNO:

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio



INSTITUTO POLITÉCNICO NACIONAL



FECHA DE ENTREGA: 15/01/2025

Tarea 1: Investigar inicialización K-means++ y algún otro.

Algoritmo K-means++

1.- Selecciona aleatoriamente el primer centroide de los puntos de datos.

2.- Para cada punto de datos, calcula su distancia desde el centroide más cercano, elegido previamente.

3.- Selecciona el siguiente centroide de los puntos de datos de modo que la probabilidad de elegir un punto como centroide sea directamente proporcional a su distancia desde el centroide más cercano previamente elegido, es decir, el punto que tiene la distancia máxima desde el centroide más cercano tiene más probabilidades de ser seleccionado a continuación como centroide.

4.- Repetir pasos 2 y 3 hasta que se hayan muestreado K centroides.

(Tarea 2. Utilizando el perceptrón visto en clase implementar el entrenamiento con las mismas características pero con la tabla de verdad de una OR.

i	$x_1 x_2$	y_d
0	0 0	0
1	0 1	1
2	1 0	1
3	1 1	1

$$l(z) = \begin{cases} 1 & z \geq 0.5 \\ 0 & z < 0.5 \end{cases}$$

$$\lambda_r = 0.5$$

$$\begin{aligned} w_0 &= 1 \\ w_1 &= 0.5 \quad x_0 = 1 \\ w_2 &= 0.2 \end{aligned}$$

Para la muestra 0

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 = (1)(1) + (0.5)(0) + (0.2)(0)$$

$$z = 1 \rightarrow l(z) = l(1) = 1 = y_{obt} \quad y_{obt} \neq y_d$$

Actualización de pesos

$$w_0 = 1 - 0.5(1 - 0) = 0.5 \quad w_1 = 0.5 - 0.5(1 - 0) = 0.5$$

$$w_2 = 0.2 - 0.5(1 - 0) = 0.2$$

Para la muestra 1

$$z = (0.5)(1) + (0.5)(0) + (0.2)(1) = z = 0.7$$

$$l(z) = l(0.7) = 1 = y_{obt} = y_d$$

Para la muestra 2

$$z = (0.5)(1) + (0.5)(1) + (0.2)(0) = z = 1$$

$$l(z) = l(1) = 1 = y_{obt} = y_d$$

Para la muestra 3

$$z = (0.5)(1) + (0.5)(1) + (0.2)(1) = z = 1.2$$

$$l(z) = l(1.2) = 1 = y_{obt} = y_d$$



INSTITUTO POLITÉCTICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

**UNIDAD DE APRENDIZAJE:
MACHINE LEARNING
CÓDIGOS**

ALUMNO:

Gonzalez Llamosas Noe Ramses

PROFESOR:

Ortiz Castillo Marco Antonio



INSTITUTO POLITÉCNICO NACIONAL



FECHA DE ENTREGA: 15/01/2025

```

ALGORITMO KMEANS
import matplotlib.pyplot as plt
import random
import numpy as np

def distancia_ecludiana(X,x_test):
    x_test = np.array(x_test)
    d = (X - x_test)**2
    distancia = [None] * len(d)
    for i in range(len(d)):
        distancia[i] = np.sqrt(d[i][0] + d[i][1])
    return distancia

def Kmeans(data, k, epochas):
    data = np.array(data)
    indices_usados = [-1]*k
    centroides = []
    num_datos, num_caracteristicas = data.shape

    ## Asignar aleatoriamente los centroides
    for i in range(k):
        while True:
            random_centroides = random.randint(0, num_datos -1) #sacar un numero aleatorio con distribucion especifica de tipo int
            duplicado = False
            for j in range(i):
                if indices_usados[j] == random_centroides:
                    duplicado = True
                    break
            if duplicado == False:
                indices_usados[i] = random_centroides
                centroides = centroides + [data[random_centroides]]
                break

    #####
    ##### Calcular la distancia de los puntos a los centroides
    for iteraciones in range(epochas):
        clouster_asignados = [0]*num_datos
        for i in range(num_datos):
            distancias = [0]*k
            for j in range(k):
                distancias[j] = distancia_ecludiana([centroides[j]], data[i])[0] #Distancia del punto a los 4 centros
            min_distancia_index=np.argmin(distancias)

```

```

        clouster_asignados[i] = min_distancia_index

#####
# Hacer el promedio de los datos que pertenecen a una clase
# (clouster) para obtener nuevos centroides
Nuevos_Centroides=[]
for closuter_index in range(k): #recorre todas las clases clouster 1,
2 ,3 hasta clase k
    puntos_en_clouster=[]
    for los_datos in range(num_datos):
        if clouster_asignados[los_datos]==closuter_index:
            puntos_en_clouster += [data[los_datos]] #apila los datos de la
            clase en una lista

#####
# Obtencion de los promedios de los datos de la clase
if len(puntos_en_clouster)>0:
    clouster_sumas= [0]* num_caracteristicas
    for dato_en_clouster in puntos_en_clouster: #recorre todos los
    puntos en el clouster
        for caracteristicas in range(num_caracteristicas): #recorre
        todas las caracteristicas
            clouster_sumas[caracteristicas] +=
dato_en_clouster[caracteristicas] #suma todos los valores de la primera
y segunda columna
    promedio_clouster = [clouster_sumas[j]/len(puntos_en_clouster)
for j in range (num_caracteristicas)]
    Nuevos_Centroides += [promedio_clouster]
else:
    Nuevos_Centroides += [centroides[closuter_index]]

## Early stop
#print(f"Centroides actuales: {len(centroides)}, Nuevos centroides:
{len(Nuevos_Centroides)}")
variacion = True
epsilon=0.0000000001
for i in range(k):
    for j in range(num_caracteristicas):
        if(abs(Nuevos_Centroides[i][j]-centroides[i][j]) > epsilon:
            variacion= False
            break
        if variacion==False:
            break
    if variacion == True:
        break

```

```

        centroides = Nuevos_Centroides

## Agrupamiento
clousters= [[] for _ in range(k)]
for i in range(num_datos):
    clousters[clouster_asignados[i]] = clousters[clouster_asignados[i]]
+ [data[i]]

return centroides,clousters

def graficar_puntos(data, clousters, centroides):
    colors= ('red', 'blue', 'green', 'purple')
    for clouster_index in range(len(clousters)):
        x_points= [clousters[clouster_index][i][0] for i in
range(len(clousters[clouster_index]))]
        y_points= [clousters[clouster_index][i][1] for i in
range(len(clousters[clouster_index]))]
        plt.scatter(x_points, y_points, color=colors[clouster_index],
label=f'Clouster {clouster_index + 1}')
        x_centroides = [centroides[i][0] for i in range(len(centroides))]
        y_centroides = [centroides[i][1] for i in range(len(centroides))]
        plt.scatter(x_centroides, y_centroides,
color='black',marker='x',s=100 ,label='Centroides')
        plt.xlabel("x-axis")
        plt.ylabel("y-axis")
        plt.title("Kmeans")
        plt.legend()
        plt.grid(True)
        plt.show()

#for i in range(len(data)):
#    plt.scatter(data[i][0], data[i][1], color='red')
#plt.xlabel("x")
#plt.ylabel("y")
#plt.title("Kmeans")
#plt.grid()
#plt.show()

data = [[1,2],[1.5,2.3],[1.2,1.9],
        [4,5],[4.1,5.1],[4.4,5.3],
        [9,10],[9.1,10.1],[8.8,10.4],
        [15,16.1],[15.2,16.5],[14.9,15.9]
]

```

```

k= 4
epochas=1000
centroides, clousters = Kmeans(data, k, epochas)
graficar_puntos(data,clousters,centroides)

```

ALGORITMO MÁQUINAS DE SOPORTE VECTORIAL

```

import numpy as np
import matplotlib.pyplot as plt

def entrenamiento_MSV(X, Y):
    numero_muestras, numero_caracteristicas = X.shape

    # Inicialización parámetros externos
    epochas = 1000
    lr = 0.01
    lamda = 1/epochas

    # Inicialización de parámetros internos
    w = np.zeros(numero_caracteristicas) + 0.1
    b = 0.1

    vectores_soporte = []
    vectores_soporte_indices = []

    for epoca in range(epochas):
        for i,x in enumerate(X):
            condicion_margen = Y[i] * (np.dot(x, w) + b) >= 1
            if condicion_margen:
                # Idealmente w no debería modificarse, no obstante se
                # recomienda modificarlo con pequeñas variaciones que garanticen que en
                # todo momento estamos alejados del margen de los vectores de soporte
                w = w - lr * (2 * lamda * w)
            else:
                w = w - lr * (2 * lamda * w - np.dot(Y[i], x))
                b = b - lr * lamda * Y[i]
            vectores_soporte += [x]
            vectores_soporte_indices += [i]

    # Cálculo del margen del hiperplano
    M = 1 / np.linalg.norm(w)

    return w, b, vectores_soporte, vectores_soporte_indices

# Fase de operación

```

```

def prediccion_MSV(X_test, w, b):
    return np.sign(np.dot(X_test, w) + b)

# Datos fijos de ejemplo
X_fixed = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6],
                     [1, 1], [2, 2], [3, 3], [4, 4], [5, 5],
                     [-1, -2], [-2, -3], [-3, -4], [-4, -5], [-5, -6],
                     [-1, -1], [-2, -2], [-3, -3], [-4, -4], [-5, -5]])

Y_fixed = np.array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1,
                    1, 1, 1, 1])

w, b, vectores_soporte, vectores_soporte_indices =
entrenamiento_MSV(X_fixed, Y_fixed)

# Graficamos los datos, el margen, los vectores de soporte y el
hiperplano
plt.scatter(X_fixed[:, 0], X_fixed[:, 1], c=Y_fixed,
cmap=plt.cm.Paired)
plt.scatter(X_fixed[vectores_soporte_indices, 0],
X_fixed[vectores_soporte_indices, 1], s=100, facecolors='none',
edgecolors='g', linewidths=2, marker='o')
plt.xlabel('Características 1')
plt.ylabel('Características 2')

# Graficar el margen
xx, yy = np.meshgrid(np.arange(X_fixed[:, 0].min() - 1, X_fixed[:, 0].max() + 1, 0.01),
                      np.arange(X_fixed[:, 1].min() - 1, X_fixed[:, 1].max() + 1, 0.01))

Z = np.dot(np.c_[xx.ravel(), yy.ravel()], w) - b
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, levels=[-1, 0, 1], alpha=0.8, linestyles=['--', '-.-'])

# Graficar el hiperplano
plt.plot(xx, (-w[0] * xx - b) / w[1], 'k--')
plt.title('Máquinas de Soporte Vectorial')
plt.show()

X_test = [2, 6]
prediccion = prediccion_MSV(X_test, w, b)
print(f"La clase del valor es {prediccion}")

```