1. Ejercicio sobre la búsqueda iterativa de óptimos

1. (1 punto) Implementar el algoritmo de gradiente descendente.

In [410]:
```
#Las Librerias que vamos a usar
#enconding: utf-8
import numpy as np
import matplotlib.pyplot as plt
```

In [411]:
```
def gradient_descent(w,learning_rate, function, iterations, epsilon = -1000,registro=False):
    #vamos a usar new_w para almacenar los nuevos componentes
    new_w=np.zeros((2,1),dtype=np.float64)

    #En caso de que queramos un registro de los puntos evaluados y su valor:
    if registro == True:
        loss_history=np.zeros((iterations,1),dtype=np.float64)
        w_history=np.zeros((iterations,2),dtype=np.float64)

    for i in range(iterations):
        #evaluamos la derivada en el punto 'w' y seguimos el sentido de la misma
        #si value<0 la variable w aumentara siguiendo el sentido negativo
        #si value>0 la variable w aumentara siguiendo el sentido positivo
        #si value=0 la variable w habra llegado a su valor optimo
        value, new_w = function(w[0],w[1])
        w=w-(learning_rate*new_w)
        if registro == True:
            loss_history[i]=value
            w_history[i]=w
        #En caso de que sea dificil llegar a obtener 0, estableceremos un valor de aceptacion
        if value<epsilon:
            break
    if registro == True:
        return w,i,loss_history,w_history
    else:
        return w, i
```

2. (2 puntos) Considerar la función E(u,v) = (u2ev −2v2e−u)2. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto (u,v) = (1,1) y usando una tasa de aprendizaje η = 0,01.

a) Calcular analíticamente y mostrar la expresión del gradiente de la función E(u,v)

In [412]:
```
#Definimos la funcion dada y sus derivadas
#Nuestra funcion
def E(u,v):
    return (u**2*(np.exp(v))-2*v**2*(np.exp(-u)))**2, gradE(u,v)

#Derivada parcial de E con respecto a u
def dEu(u,v):
    return 2*((np.exp(v))*u**2-2*v**2*(np.exp(-u)))*(2*v**2*(np.exp(-u))+2*(np.exp(v))*u)
#Derivada parcial de E con respecto a v
def dEv(u,v):
    return 2*(u**2*(np.exp(v))-4*(np.exp(-u))*v)*(u**2*(np.exp(v))-2*(np.exp(-u))*v**2)
#Gradiente de E
def gradE(u,v):
    return np.array([dEu(u,v), dEv(u,v)])
```

In [413]:
```
#Datos del enunciado para evaluar
eta = 0.01
initial_point = np.array([1.0,1.0])
maxIter = 10000000000
```

b)¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u,v) inferior a 10−14 . (Usar flotantes de 64 bits)

In [414]:
```
#valor del enunciado
error2get = 1e-14
#funcion que vamos a evaluar en este apartado de los ejercicios
funcion = E
w, it = gradient_descent(initial_point, eta,funcion,maxIter,error2get)

print("***************************************************")
print("\tHa tardado " + str(it) + " en encontrar la solución" )
print("***************************************************\n")
```

```
***************************************************
        Ha tardado 33 en encontrar la solución
***************************************************
```

c) ¿En qué coordenadas (u,v) se alcanzó por primera vez un valor igual o menor a 10−14 en el apartado anterior.

In [415]:
```
print("*********************************************************")
print("En el punto " + str(w) + " encontro la solución" )
print("*********************************************************\n")
```

```
*********************************************************
En el punto [0.61920767 0.96844827] encontro la solución
*********************************************************
```

3. (2 puntos) Considerar ahora la función f(x,y) = x2 + 2y2 + 2sin(2πx)sin(2πy)

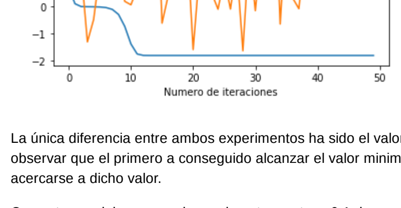
a) Usar gradiente descendente para minimizar esta función. Usar como punto inicial (x0 = 0,1,y0 = 0,1), (tasa de aprendizaje η = 0,01 y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando η = 0,1, comentar las diferencias y su dependencia de η.

In [416]:
```
#Las nueva funcion con su respectivas derivadas parciales
def E_2(u,v):
    return u**2.0+2.0*v**2.0+2.0*np.sin(2.0*np.pi*u)*np.sin(2.0*np.pi*v),gradE_2(u,v)
#Derivada parcial de E con respecto a u
def dEu_2(u,v):
    return (4.0*np.pi*np.sin(2.0*np.pi*v)*np.cos(2.0*np.pi*u)+2.0*u)
#Derivada parcial de E con respecto a v
def dEv_2(u,v):
    return (4.0*(np.pi*np.sin(2.0*np.pi*u)*np.cos(2.0*np.pi*v)+v))
#Gradiente de E
def gradE_2(u,v):
    return np.array([dEu_2(u,v), dEv_2(u,v)])
```

In [417]:
```
#Primero vamos a resolver para el caso de la tasa de aprendizaje de 0.01
#Los datos sobre los que vamos a evaluar:

eta = 0.01
initial_point = np.array([0.1,0.1])
maxIter = 50
funcion = E_2
#No le pasamos ningun valor de aceptacion ya que queremos evaluar todas las iteraciones
w1, it1,Valores_en_puntos1, puntos_evaluados1 = gradient_descent(initial_point, eta,funcion,maxIter,
registro=True)
```

In [418]:
```
#Primero vamos a resolver para el caso de la tasa de aprendizaje de 0.1

eta = 0.1
w2, it2,Valores_en_puntos2, puntos_evaluados2 = gradient_descent(initial_point, eta,funcion,maxIter,
registro=True)
```

In [419]:
```
plt.plot(Valores_en_puntos1,label='Learning rate=0.01')
plt.plot(Valores_en_puntos2,label='learning rate=0.1')
plt.xlabel('Numero de iteraciones')
plt.ylabel('Valor del error')
plt.title('Evolucion del valor de la funcion')
plt.legend()
plt.show()
```



La única diferencia entre ambos experimentos ha sido el valor del Learning rate, sin embargo, con esta alteración podemos observar que el primero a conseguido alcanzar el valor minimo posible, por el contrario el segundo solo ha conseguido acercarse a dicho valor.

Con esto concluimos que el experimento con tasa 0.1 de aprendizaje, ha estado oscilando entorno al minimo debido a su alta tasa de aprendizaje, que hace que su valor se incremente demasiado haciendo que este lo sobrepasase.

Por tanto en el caso del primer experimento, la tasa de aprendizaje era adecuada y por ello en ningun momento hemos oscilado entorno al minimo.

b) Obtener el valor mínimo y los valores de las variables (x,y) en donde se alcanzan cuando el punto de inicio se fija: (0,1,0,1), (1,1),(−0,5,−0,5),(−1,−1). Generar una tabla con los valores obtenidos

In [420]:
```
eta = 0.01
print("******************como aprendizaje 0.01***********************")
w, it2,Valores_en_puntos, Puntos_evaluados = gradient_descent(np.array([0.1,0.1]), eta,funcion,maxIt
er,registro=True)
w2, it2,Valores_en_puntos2, Puntos_evaluados2 = gradient_descent(np.array([1.0,1.0]), eta,funcion,ma
xIter,registro=True)
w3, it3,Valores_en_puntos3, Puntos_evaluados3 = gradient_descent(np.array([-0.5,-0.5]), eta,funcion,
maxIter,registro=True)
w4, it4,Valores_en_puntos4, Puntos_evaluados4 = gradient_descent(np.array([-1.0,-1.0]), eta,funcion,
maxIter,registro=True)

print("Coordenadas Iniciales | Coordenadas del minimo\t |\t Valor del minimo")
print (str(np.array([0.1,0.1]))+"\t\t"+str(Puntos_evaluados[np.argmin(Valores_en_puntos)])+"\t"+str(
Valores_en_puntos[np.argmin(Valores_en_puntos)]))
print (str(np.array([1.0,1.0]))+"\t\t\t"+str(Puntos_evaluados2[np.argmin(Valores_en_puntos)])+"\t\t"
+str(Valores_en_puntos2[np.argmin(Valores_en_puntos)]))
print (str(np.array([-0.5,-0.5]))+"\t\t"+str(Puntos_evaluados3[np.argmin(Valores_en_puntos)])+"\t"+s
tr(Valores_en_puntos3[np.argmin(Valores_en_puntos)]))
print (str(np.array([-1.0,-1.0]))+"\t\t"+str(Puntos_evaluados4[np.argmin(Valores_en_puntos)])+"\t"+s
tr(Valores_en_puntos4[np.argmin(Valores_en_puntos)]))

#utilizo como aprendizaje 0.1 que nos ha dado un peor resultado
eta = 0.1
w, it2,Valores_en_puntos, Puntos_evaluados = gradient_descent(np.array([0.1,0.1]), eta,funcion,maxIt
er,registro=True)
w2, it2,Valores_en_puntos2, Puntos_evaluados2 = gradient_descent(np.array([1.0,1.0]), eta,funcion,ma
xIter,registro=True)
w3, it3,Valores_en_puntos3, Puntos_evaluados3 = gradient_descent(np.array([-0.5,-0.5]), eta,funcion,
maxIter,registro=True)
w4, it4,Valores_en_puntos4, Puntos_evaluados4 = gradient_descent(np.array([-1.0,-1.0]), eta,funcion,
maxIter,registro=True)

print("\n\n******************como aprendizaje 0.1***********************")
print("Coordenadas Iniciales | Coordenadas del minimo\t |\t Valor del minimo")
print (str(np.array([0.1,0.1]))+"\t\t"+str(Puntos_evaluados[np.argmin(Valores_en_puntos)])+"\t\t"+st
r(Valores_en_puntos[np.argmin(Valores_en_puntos)]))
print (str(np.array([1.0,1.0]))+"\t\t\t"+str(Puntos_evaluados2[np.argmin(Valores_en_puntos)])+"\t\t"
+str(Valores_en_puntos2[np.argmin(Valores_en_puntos)]))
print (str(np.array([-0.5,-0.5]))+"\t\t"+str(Puntos_evaluados3[np.argmin(Valores_en_puntos)])+"\t"+s
tr(Valores_en_puntos3[np.argmin(Valores_en_puntos)]))
print (str(np.array([-1.0,-1.0]))+"\t\t"+str(Puntos_evaluados4[np.argmin(Valores_en_puntos)])+"\t"+s
tr(Valores_en_puntos4[np.argmin(Valores_en_puntos)]))
```

```
******************como aprendizaje 0.01***********************
Coordenadas Iniciales | Coordenadas del minimo   |       Valor del minimo
[0.1 0.1]               [ 0.24380497 -0.23792582]       [-1.82007854]
[1. 1.]                 [1.2180703  0.71281195]         [0.59326937]
[-0.5 -0.5]             [-0.73137746 -0.23785536]       [-1.33248106]
[-1. -1.]               [-1.2180703  -0.71281195]       [0.59326937]


******************como aprendizaje 0.1***********************
Coordenadas Iniciales | Coordenadas del minimo   |       Valor del minimo
[0.1 0.1]               [0.23624085 0.22922037]         [-1.64522162]
[1. 1.]                 [0.46440259 0.06919706]         [0.19108031]
[-0.5 -0.5]             [-0.16663291  0.00226581]       [1.04756891]
[-1. -1.]               [-0.46440259 -0.06919706]       [0.19108031]
```

4. (2 punto) ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

La dificultad reside en:

-Encontrar un punto de inicio que permita llegar al óptimo global sin quedar atascado en óptimos locales.

-Encontrar una tasa de aprendizaje que sea lo suficientemente buena como para encontrar el optimo rapidamente pero sin llegar a oscilar entorno a él.

-Encontrar un criterio de parada fiable.

2. Ejercicio sobre Regresión Lineal

Este ejercicio ajusta modelos de regresión a vectores de características extraidos de imágenes de digitos manuscritos. En particular se extraen dos característcas concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

1. (2.5 puntos) Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudoinversa como Gradiente descendente estocástico (SGD). Las etiquetas serán {−1,1}, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando Ein y Eout (para Eout calcular las predicciones usando los datos del fichero de test). (usar Regress_Lin(datos,label) como llamada para la función (opcional)).

In [421]:
```
#Función para la lectura de datos desde fichero
def readData(file_x, file_y):
    # Leemos los ficheros
    datax = np.load(file_x)

    datay = np.load(file_y)
    y = []
    x = []
    # Solo guardamos los datos cuya clase sea la 1 o la 5
    for i in range(0,datay.size):
        if datay[i] == 5 or datay[i] == 1:
            if datay[i] == 5:
                y.append(label5)
            else:
                y.append(label1)
            x.append(np.array([1,datax[i][0], datax[i][1]]))
    x = np.array(x, np.float64)
    y = np.array(y, np.float64)

    return x, y
```

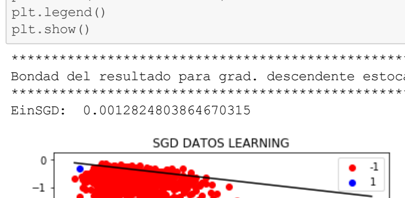
In [422]:
```
def sgd(X,Y,learning_rate,iterations,epsilon,batch):
    w=np.zeros((X.shape[1]),dtype=np.float64)
    w_anterior=np.copy(w)
    idx=np.arange(0,X.shape[0])
    for n in range(iterations):
        np.random.shuffle(idx)
        X[idx]
        Y[idx]
        minibatch=X[0:batch,:]
        w_anterior=np.copy(w)
        for i in range (minibatch.shape[0]):
            sumatory=Sumatoria(X[i],Y[i],w,X.shape[0])
            w=w-(learning_rate*sumatory)

        if np.linalg.norm((w_anterior-w),1)<epsilon:
            break

    return w

def Sumatoria (xi,yi,w,n):
    a=-np.dot(xi,yi)*sigmoid(-yi*np.dot(xi,w.transpose()))

    return a/n

#Funcion que calcula el error a la hora de clasificar
def Err(X,Y,W):
    suma=0

    for xi,yi in zip(X,Y):
        a=1+np.exp(yi*np.dot(w,xi))
        suma+=np.log(a)
    return a/X.shape[0]

###Funcion para calcular la funcion sigmoide
def sigmoid (x):
    return 1/(1+np.exp(-x))
```

In [423]:
```
def PIA(X,Y):
    #Pseudo-inverse algorithm
    #calculo la transpuesta de X
    x_t = X.transpose()

    pseudo_inverse=np.linalg.inv(x_t.dot(X))
    pseudo_inverse=pseudo_inverse.dot(x_t)
    pseudo_inverse=pseudo_inverse.dot(Y)

    return pseudo_inverse
```

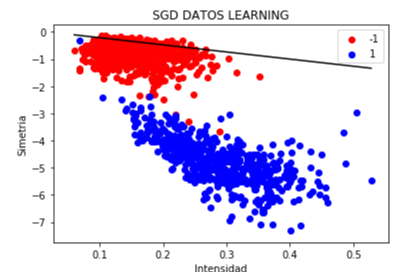
In [424]:
```
label5 = 1
label1 = -1

# Lectura de los datos de entrenamiento
x, y = readData('datos/X_train.npy', 'datos/y_train.npy')
# Lectura de los datos para el test
x_test, y_test = readData('datos/X_test.npy', 'datos/y_test.npy')

eta = 0.01
initial_point = np.array([-1,-1])
maxIter = 100
error2get = 1e-14
batch = 32

w = sgd(x,y,eta,maxIter,error2get,batch)

w_pia=PIA(x,y)

def errorRL(X,Y,w):
    suma=0

    for xi,yi in zip(X,Y):
        a=1+np.exp(yi*np.dot(w,xi))
        suma+=np.log(a)
    return a/X.shape[0]
```

In [425]:
```
labels= [label1,label5]
w= w.reshape(-1)

print("**********************************************************")
print ('Bondad del resultado para grad. descendente estocastico:')
print("**********************************************************")

print ("EinSGD: ", Err(x, y, w))
h=0
colores=['red','blue']
for i in np.unique(y):
    pos= y == i
    x_aux = x[pos,:]
    plt.scatter(x_aux[:,1],x_aux[:,2],label=labels[h], c=colores[h])
    h=h+1

a = -w[1]/w[2]
b = -w[0]/w[2]

#No sé el por qué pero con sgd tengo algún problema que no he conseguido solucionar

plt.plot([np.amin(x[:,1]),np.amax(x[:,1])],[a,b],'k-')
plt.xlabel('Intensidad')
plt.ylabel('Simetria')
plt.title('SGD DATOS LEARNING')
plt.legend()
plt.show()
##############################################

print ("EoutSGD: ", Err(x_test, y_test, w))

h=0

for i in np.unique(y_test):
    pos= y_test == i
    x_aux = x_test[pos,:]
    plt.scatter(x_aux[:,1],x_aux[:,2],label=labels[h], c=colores[h])
    h=h+1

a = -w[1]/w[2]
b = -w[0]/w[2]

#No sé el por qué pero con sgd tengo algún problema que no he conseguido solucionar
plt.plot([np.amin(x_test[:,1]),np.amax(x_test[:,1])],[a,b],'k-')
plt.xlabel('Intensidad')
plt.ylabel('Simetria')
plt.title('SGD DATOS TEST')
plt.legend()
plt.show()

#################################################
print("\n**********************************************************")
print ('Bondad del resultado para la pseudo-inversa:')
print("**********************************************************")
print ("EINpseudo-inversa: "+str(Err(x,y,w_pia)))

h=0
for i in np.unique(y):
    pos= y == i
    x_aux = x[pos,:]
    plt.scatter(x_aux[:,1],x_aux[:,2],label=labels[h], c=colores[h])
    h=h+1

a = -w_pia[1]/w_pia[2]
b = -w_pia[0]/w_pia[2]
plt.plot([np.amin(x[:,1]),np.amax(x[:,1])],[a,b],'k-')

plt.xlabel('Intensidad')
plt.ylabel('Simetria')
plt.title('SGD DATOS LEARNING')
plt.legend()
plt.show()

###################################################
print ("EOUTpseudo-inversa: "+str(Err(x_test,y_test,w_pia)))

h=0

for i in np.unique(y_test):

    pos= y_test == i
    x_aux = x_test[pos,:]
    plt.scatter(x_aux[:,1],x_aux[:,2],label=labels[h], c=colores[h])
    h=h+1

a = -w_pia[1]/w_pia[2]
b = -w_pia[0]/w_pia[2]
plt.plot([np.amin(x_test[:,1]),np.amax(x_test[:,1])],[a,b],'k-')

plt.xlabel('Intensidad')
plt.ylabel('Simetria')
plt.title('SGD DATOS TEST')
plt.legend()
plt.show()
```

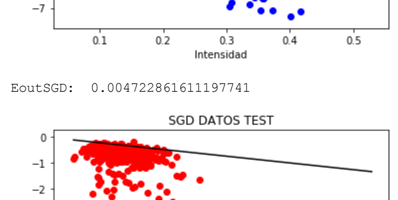
```
**********************************************************
Bondad del resultado para grad. descendente estocastico:
**********************************************************
EinSGD:  0.0012824803864670315
```



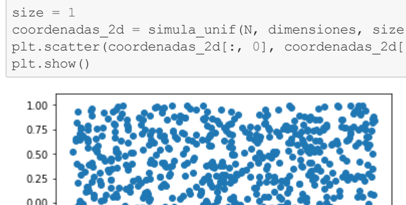

```
EoutSGD:  0.004722861611197741
```




```
**********************************************************
Bondad del resultado para la pseudo-inversa:
**********************************************************
EINpseudo-inversa: 0.0012824803864670315
```




```
EOUTpseudo-inversa: 0.004722861611197741
```

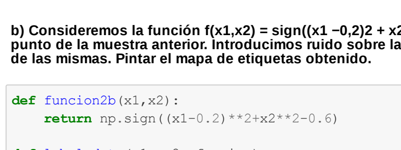


2. En este apartado exploramos como se transforman los errores Ein y Eout cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función simula_unif(N,2,size) que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por [−size,size]×[−size,size]

def simula_unif(N, d, size): return np.random.uniform(-size,size,(N,d))

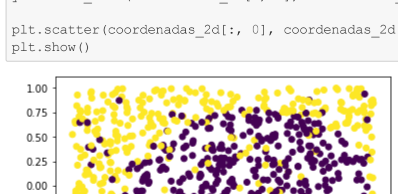
a) Generar una muestra de entrenamiento de N = 1000 puntos en el cuadrado X = [−1,1]×[−1,1]. Pintar el mapa de puntos 2D. (ver función de ayuda)

In [426]:
```
N=1000
dimensiones = 2
size = 1
coordenadas_2d = simula_unif(N, dimensiones, size)
plt.scatter(coordenadas_2d[:, 0], coordenadas_2d[:, 1])
plt.show()
```



b) Consideremos la función f(x1,x2) = sign((x1 −0,2)2 + x2 2 −0,6) que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10% de las mismas. Pintar el mapa de etiquetas obtenido.

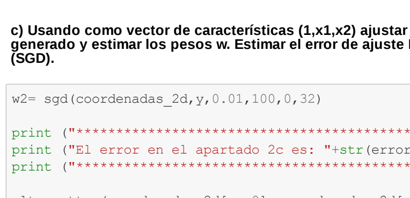
In [427]:
```
def funcion2b(x1,x2):
    return np.sign((x1-0.2)**2+x2**2-0.6)

def label_data(x1, x2, funcion):
    y = funcion(x1,x2)
    idx = np.random.choice(range(y.shape[0]), size=(int(y.shape[0]*0.1)), replace=True)
    y[idx] *= -1

    return y

y = label_data(coordenadas_2d[:, 0], coordenadas_2d[:, 1],funcion2b)

plt.scatter(coordenadas_2d[:, 0], coordenadas_2d[:, 1],c=y)
plt.show()
```



c) Usando como vector de características (1,x1,x2) ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste Ein usando Gradiente Descendente Estocástico (SGD).

In [428]:
```
w2= sgd(coordenadas_2d,y,0.01,100,0,32)

print ("**************************************************")
print ("El error en el apartado 2c es: "+str(errorRL(coordenadas_2d,y,w2)))
print ("**************************************************")

plt.scatter(coordenadas_2d[:, 0], coordenadas_2d[:, 1],c=y)

a = -w[1]/w[2]
b = -w[0]/w[2]
plt.plot([np.amin(coordenadas_2d[:,1]),np.amax(coordenadas_2d[:,1])],[a,b],'k-')
plt.axis( [np.amin(coordenadas_2d[:,1]),np.amax(coordenadas_2d[:,1]), -1.05,1.05] )
plt.show()
```

```
**************************************************
El error en el apartado 2c es: 0.002000360266977711
**************************************************
```



d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y

• Calcular el valor medio de los errores Ein de las 1000 muestras.

• Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de Eout en dicha iteración. Calcular el valor medio de Eout en todas las iteraciones.

In [429]:
```
iterations = 1000
Ein=0
X_train=0
Eout=0
for i in range (iterations):
    X= simula_unif(N=1000, d=2, size=1)
    y= label_data(X[:, 0], X[:, 1],funcion2b)
    X=np.c_[X,np.ones((X.shape[0],1))]

    w=np.zeros((1,3),dtype=np.float64)
    w=sgd(X, y,0.01,10,0,100)
    error_in=Err(X,y,w)
    Ein=Ein+error_in

    x_test=simula_unif(N=1000, d=2, size=1)
    x_test=np.c_[x_test,np.ones((x_test.shape[0],1))]

    y_test =label_data(x_test[:, 0], x_test[:, 1],funcion2b)

    error_out=Err(x_test,y_test,w)
    Eout=Eout+error_out
    if i == iterations/40:
        plt.scatter(x_test[:, 0], x_test[:, 1],c=y_test)

        a = -w[1]/w[2]
        b = -w[0]/w[2]
        plt.plot([np.amin(x_test[:,1]),np.amax(x_test[:,1])],[a,b],'k-')
        plt.axis( [np.amin(x_test[:,1]),np.amax(x_test[:,1]), -1.05,1.05] )
        plt.show()

Ein=Ein/iterations
Eout=Eout/iterations

print ("El error medio para el train es de: "+str(Ein))
print ("El error medio para el test  es de: "+str(Eout))
```