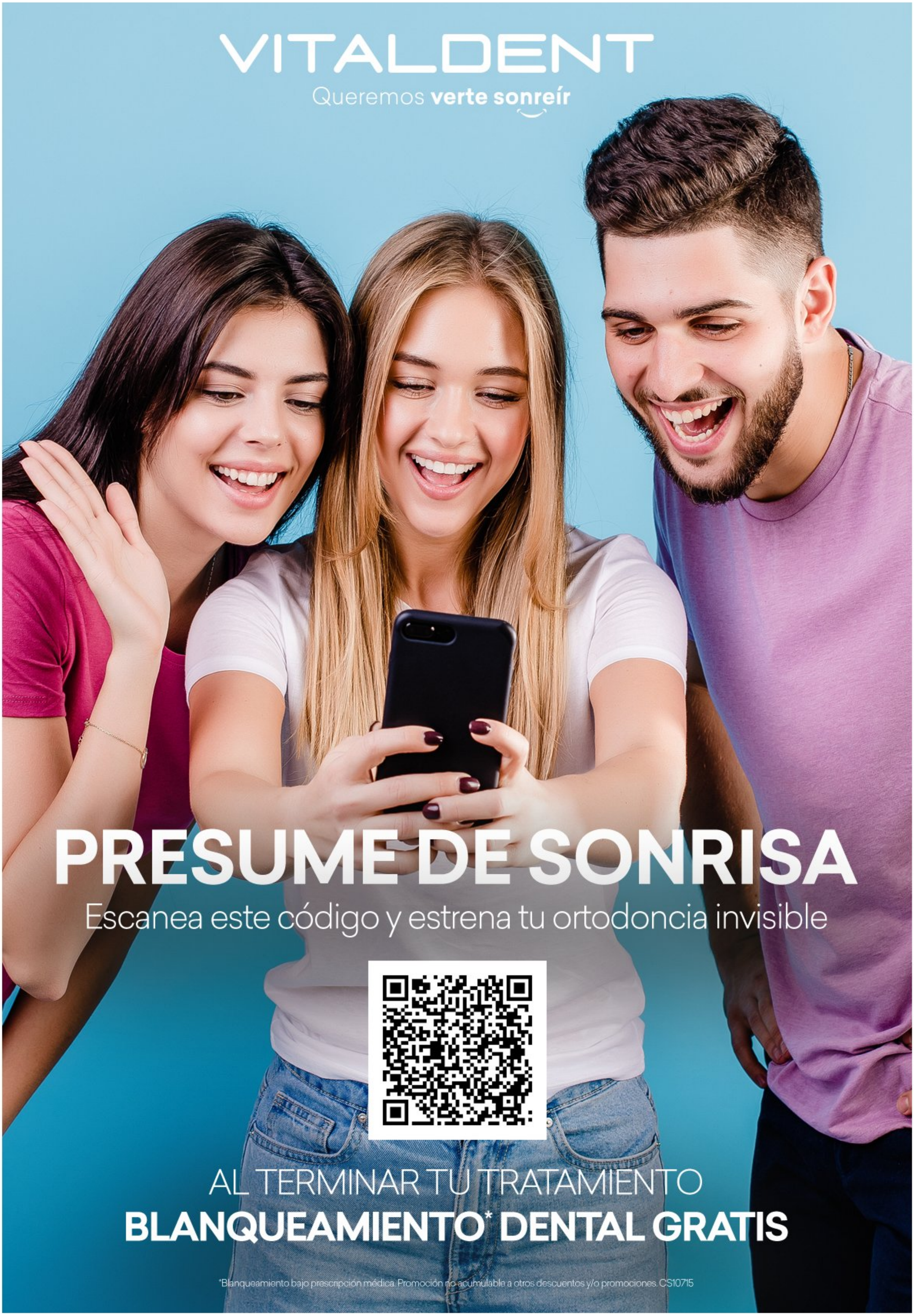
codej2.pdf

postdata9

Aprendizaje Automatico

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada



VITALDENT
Queremos verte sonreír
PRESUME DE SONRISA
Escanea este código y estrena tu ortodoncia invisible
AL TERMINAR TU TRATAMIENTO
BLANQUEAMIENTO* DENTAL GRATIS
*Blanqueamiento bajo prescripción médica. Promoción no acumulable a otros descuentos y/o promociones. CS10715

PRÁCTICA 2:

Ejercicio 2

```python
import numpy as np
import matplotlib.pyplot as plt
import statistics as stats

print("\n")
print("########################################\n")
print('--- EJERCICIO SOBRE MODELOS LINEALES ---')
input("########################################\n")

np.random.seed(1)
#simula_unif
# --- Parámetros:
#        - N: número de vetores / tamaño de la lista de retorno
#        - dim: tamaño de cada vector
#        - rango: intervalo de los valores de los números de cada vector
#
# --- Retorno:
#        - Una lista de N vectores con tamaño dim cada vector, cuyos números
#        están en el intervalo rango
def simula_unif(N, dim, rango):
      return np.random.uniform(rango[0],rango[1],(N,dim))


#simula_gaus
# --- Parámetros:
#        - N: número de vectores / tamaño de la lista de retorno
#        - dim: tamaño de cada vector
#        - sigma: intervalo que determinará la función gaussiana, son dos valores
#              el primero es la media y el segundo la desviación típica
#
# --- Retorno:
#        Una lista de N vectores de tamaño dim con valores que estarán dentro de la
#        función gaussiana
def simula_gaus(N, dim, sigma):
    media = 0
    out = np.zeros((N,dim),np.float64)
    for i in range(N):
        # Para cada columna dim se emplea un sigma determinado. Es decir, para
        # la primera columna se usará una N(0,sqrt(5)) y para la segunda N(0,sqrt(7))
        out[i,:] = np.random.normal(loc=media, scale=np.sqrt(sigma), size=dim)

    return out


#simula_recta
# --- Parámetros:
#        - intervalo: rango del cual se extraerán los valores aleatoriamente los
#             puntos (x1,y1) y (x2,y2) que definen una recta
#
# --- Retorno:
#        Los valores de a y b, donde a es la pendiente de la recta, y
#        b, el término independiente
def simula_recta(intervalo):
    points = np.random.uniform(intervalo[0], intervalo[1], size=(2, 2))
    x1 = points[0,0]
    x2 = points[1,0]
    y1 = points[0,1]
    y2 = points[1,1]
    # y = a*x + b
    a = (y2-y1)/(x2-x1) # Calculo de la pendiente.
    b = y1 - a*x1       # Calculo del termino independiente.

    return a, b
```

```python
#error
# --- Parámetros:
#       - x: puntos, vector de dos columnas
#       - y: etiquetas de los puntos
#       - w: vector de pesos
#
# --- Retorno:
#       El error del algoritmo Perceptron
def error(ptos, etiq, w):
      error = 0

    #realizamos un for que realiza tantas iteraciones como puntos haya
      for j in range(ptos.shape[0]):

        #calculo la etiqueta
             etiq_cal = np.sign( np.dot(w.T, ptos[j]))

        #si es distinta que la que debería salir, aumento el error
             if (etiq_cal != etiq[j]):
                   error += 1

      return error


#ajuste_PLA
# --- Parámetros:
#       - puntos: un vector de puntos con dos columnas, la primera son las
#                 coordenadas x, la segunda las coordenadas y
#       - etiquetas: vector de 1, -1, que se corresponde con las etiquetas de los ptos
#       - max_iter: número de iteraciones máximo que va a ejecutar el algoritmo
#       - w_act: vector inicial de dimensión 3
#
# --- Retorno:
#       Devuelve un vector de pesos de 3 elementos, que definen la recta divisoria
#       de los puntos.
def ajuste_PLA(puntos, etiquetas, max_iter, w_act):

    #vector de pesos antiguo, le asigno valor para que pase la primera condición del
    #while pero no importa qué valor sea, ya que justo después del while adquiere el
    #valor de w_act
    w_ant = w_act + 1

    #a los puntos, le añado una columna de 1
    puntos = np.c_[np.ones(puntos.shape[0]), puntos]

    #it es el contador de iteraciones
    it = 0

    #Vamos iterando que llegue al máximo de iteraciones o el valor del vector de
    #pesos no cambie
    while it < max_iter and np.array_equal(w_act,w_ant) == False:

            w_ant = w_act   #w_ant lo actualizo con w_act
            it = it + 1     #incremento el número de iteraciones
            j = 0           #j mantiene el número de iteraciones del siguiente while

            #vamos a iterar hasta que llegue al tamaño de las etiquetas
            while j < etiquetas.size:
                #calculamos su etiqueta, y si es distinta de la que tenemos,
                #actualizamos el vector de pesos actual
                if(np.sign(np.dot(w_act.T, puntos[j])) != etiquetas[j]):
                    w_act = w_act + etiquetas[j]*puntos[j]

                    #si el error es 0, devuelvo el número de iteraciones
                    # y el vector de pesos
                    if(error(puntos, etiquetas, w_act) == 0):
```

```
                        return w_act, it

                #incremento el índice j
                j = j + 1

    return w_act, it


#pinta_recta
# --- Parámetros:
#       - w: vector de pesos obtenidos tras el PLA
#       - i: un índice que determina si queremos pintar la línea con leyenda o sin ella
#           i = 0: la pintamos con leyenda
#           i = 1: la pintamos sin leyenda
#
# --- Retorno:
#       Dibuja la recta divisioria
def pinta_recta(w, i):

    #obtenemos el valor de la pendiente
#    a = (-w[2]/w[1])/(w[2]/w[0])
#
#    #obtenemos el valor de la constante
#    b = (-w[2]/w[1])
#
#    obtenemos el valor de la pendiente
    a = (-w[0]/w[2])/(w[0]/w[1])

    #obtenemos el valor de la constante
    b = (-w[0]/w[2])

    axes = plt.gca()                  #obtengo los ejes
    val_x = np.array(axes.get_xlim()) #los valores del eje x
    val_y = b + a * val_x             #los valores del eje y

    if(i == 0):
        #pintamos la gráfica
        plt.plot(val_x, val_y, '-', label='Recta divisoria', c='green')
        plt.legend()

    else:
        #pintamos la gráfica
        plt.plot(val_x, val_y, '-', c='green')

    plt.axis([-60,60,-60,60])
```

```
##########################
#                        #
#      Ejercicio 1       #
#                        #
##########################

print("*********************************************\n")
print('*** EJERCICIO 1 --- ALGORITMO PERCEPTRON ***')
input('*********************************************\n')
print('Implementar el algoritmo Perceptron: ajusta_PLA(datos, label, max_iter, v_ini).
\n')


--------------------------------
|                              |
|         Apartado a.a)        |
|                              |
--------------------------------

input(" -- apartado a.a)")
print("_______________")
print('Ejecutar el algoritmo PLA con los datos del 1.2a de la sección anterior con un
vector inicial de 0.')

#parámetros para obtener los datos uniformes
N = 50 ; dim = 2 ; intervalo = [-50,50]

#obtenemos los datos con los datos con los que vamos a trabajar
datos = simula_unif(N, dim, intervalo)

#para poder coger las etiquetas de los puntos, necesito la pendiente y el término
independiente
x = datos[:,0] ; y = datos[:,1] ; a, b = simula_recta(intervalo)

#obtengo las etiquetas, el signo de los puntos y lo almaceno en etiquetas
etiquetas = np.sign(y - a*x - b)

#necesito un vector inicial de 3 elementos con valor 0
v_i= np.zeros(3, np.float64)

#llamo a la función para obtener el vector de pesos y el número de iteraciones
w, i = ajuste_PLA(datos, etiquetas, 50, v_i)

#pinto la gráfica
plt.title('Gráfica 1.a.a, con vector inicial a 0.')

#primero selecciono aquellos datos cuya etiqueta sea negativa y los pinto morado
plt.scatter(datos[etiquetas < 0, 0], datos[etiquetas < 0, 1], c='purple',
label='Etiqueta -1')

#después selecciono aquellos datos cuya etiqueta sea positiva y los pinto de naranja
plt.scatter(datos[etiquetas > 0, 0], datos[etiquetas > 0, 1], c='orange',
label='Etiqueta 1')

plt.xlabel('Eje x')
plt.ylabel('Eje y')

#pinto la recta pasándole el vector de pesos obtenido con el pla
pinta_recta(w,0)
plt.legend()
plt.show()

print('El algoritmo necesita', i, 'iteraciones.\n')
```

```
--------------------------------
|                              |
|         Apartado a.b)        |
|                              |
--------------------------------

input(" -- apartado a.b)")
print("______________")
print('Ejecutar el algoritmo PLA 10 veces con los datos del 1.2a de la sección anterior
con un vector inicial aleatorio entre 0 y 1.')

#en lista_w vamos a almacenar los 10 resultados de ejecutar el pla
lista_w = []
lista_i = []

#realizamos un bucle que itere 10 veces
for i in range(0,10):

    #inicializo el vector inicial de 3 elementos
    v_i = np.zeros(3, np.float64)

    #y le doy valores aleatorios
    for j in range(0,3):
        v_i[j] = np.random.rand(1,1)

    #obtengo los pesos del pla para dicho vector inicial aleatorio
    w, it = ajuste_PLA(datos, etiquetas, 600, v_i)

    #y lo almaceno en lista_w
    lista_w.append(w)
    lista_i.append(it)

#pinto la gráfica
plt.title('Gráfica 1.a.b, con vectores iniciales aleatorios entre [0,1)')

#primero selecciono aquellos datos cuya etiqueta sea negativa y los pinto morado
plt.scatter(datos[etiquetas < 0, 0], datos[etiquetas < 0, 1], c='purple',
label='Etiqueta -1')

#después selecciono aquellos datos cuya etiqueta sea positiva y los pinto de naranja
plt.scatter(datos[etiquetas > 0, 0], datos[etiquetas > 0, 1], c='orange',
label='Etiqueta 1')

plt.xlabel('Eje x')
plt.ylabel('Eje y')

#pinto la recta con el primer resultado de la lista con etiqueta
pinta_recta(lista_w[0],0)

#pinto las siguientes listas sin etiqueta
for a in lista_w:
    pinta_recta(a,1)
plt.axis([-60,60,-60,60])
plt.show()

media_i = stats.mean(lista_i)
print('El número medio de iteraciones es', media_i, '\n')
```

```
---------------------------------
|                               |
|          Apartado b)          |
|                               |
---------------------------------

input(" -- apartado b)")
print("—————————————")
print('Realizar lo mismo que el apartado anterior, pero con los datos del 1.2b.')

et_neg = [] ; et_neg = datos[etiquetas == -1.0]
et_pos = [] ; et_pos = datos[etiquetas == 1]

#averiguamos cuántos elementos forman el 10% de la lista de etiquetas negativas y
positivas
u_pc_n = int(len(et_neg) * 0.1) #negativas
u_pc_p = int(len(et_pos) * 0.1) #positivas

et_neg_ruido = et_neg
et_pos_ruido = et_pos

#las desordenamos
np.random.shuffle(et_neg_ruido)
np.random.shuffle(et_pos_ruido)

#a los u_pc_n/p elementos primeros les cambiamos al etiqueta, esto es, los sacamos
#de su correspondiente lista y los almacenamos en la de su otra clase

#almacenamos en variables auxiliares los valores que queremos sacar
val_n = et_neg_ruido[0:u_pc_n]    #negativos
val_p = et_pos_ruido[0:u_pc_p]    #positivos

#los eliminamos de sus correspondientes listas
et_neg_ruido = et_neg_ruido[u_pc_n:]
et_pos_ruido = et_pos_ruido[u_pc_p:]

#y los almacenamos en la lista de la otra clase
#en la lista de etiquetas positivas, voy a meter los valores negativos
#y le inserto una columna de 1, -1 que establece la etiqueta
et_neg_ruido = np.concatenate((et_neg_ruido, val_p), axis=0)
et_neg_ruido = np.c_[et_neg_ruido, np.ones(et_neg_ruido.shape[0])*-1.0]

et_pos_ruido = np.concatenate((et_pos_ruido, val_n), axis=0)
et_pos_ruido = np.c_[et_pos_ruido, np.ones(et_pos_ruido.shape[0])]

datos_ruido = np.concatenate((et_neg_ruido, et_pos_ruido), axis=0)
np.random.shuffle(datos_ruido)

#necesito un vector inicial de 3 elementos con valor 0
v_i= np.zeros(3, np.float64)

#llamo a la función para obtener el vector de pesos y el número de iteraciones
w_r, i = ajuste_PLA(datos_ruido[:, :2], datos_ruido[:, 2], 50, v_i)

#pinto la gráfica
plt.title('Gráfica 1.b, con vector inicial a 0.')

#primero selecciono aquellos datos cuya etiqueta sea negativa y los pinto morado
plt.scatter(datos_ruido[datos_ruido[:,2] < 0, 0], datos_ruido[datos_ruido[:,2] < 0, 1],
c='purple', label='Etiqueta -1')

#después selecciono aquellos datos cuya etiqueta sea positiva y los pinto de naranja
plt.scatter(datos_ruido[datos_ruido[:,2] > 0, 0], datos_ruido[datos_ruido[:,2] > 0, 1],
c='orange', label='Etiqueta 1')

plt.xlabel('Eje x')
```

```python
plt.ylabel('Eje y')

#pinto la recta pasándole el vector de pesos obtenido con el pla
pinta_recta(w_r,0)
plt.legend()
input('Gráfica con vector inicial de 0')
plt.show()

print('El algoritmo necesita', i, 'iteraciones.\n')


#en lista_w vamos a almacenar los 10 resultados de ejecutar el pla
lista_w_r = []
lista_i_r = []

#realizamos un bucle que itere 10 veces
for i in range(0,10):

    #inicializo el vector inicial de 3 elementos
    v_i = np.zeros(3, np.float64)

    #y le doy valores aleatorios
    for j in range(0,3):
        v_i[j] = np.random.rand(1,1)

    #obtengo los pesos del pla para dicho vector inicial aleatorio
    w_r, it_r = ajuste_PLA(datos_ruido[:, :2], datos_ruido[:, 2], 50, v_i)

    #y lo almaceno en lista_w
    lista_w_r.append(w_r)
    lista_i_r.append(it_r)

#pinto la gráfica
plt.title('Gráfica 1.b, con vectores iniciales aleatorios entre [0,1)')

#primero selecciono aquellos datos cuya etiqueta sea negativa y los pinto morado
plt.scatter(datos_ruido[datos_ruido[:,2] < 0, 0], datos_ruido[datos_ruido[:,2] < 0, 1],
c='purple', label='Etiqueta -1')

#después selecciono aquellos datos cuya etiqueta sea positiva y los pinto de naranja
plt.scatter(datos_ruido[datos_ruido[:,2] > 0, 0], datos_ruido[datos_ruido[:,2] > 0, 1],
c='orange', label='Etiqueta 1')

plt.xlabel('Eje x')
plt.ylabel('Eje y')

#pinto la recta con el primer resultado de la lista con etiqueta
pinta_recta(lista_w_r[0],0)

#pinto las siguientes listas sin etiqueta
for a in lista_w_r:
    pinta_recta(a,1)
plt.axis([-60,60,-60,60])

input('Gráfica con vectores iniciales aleatorios.')
plt.show()

media_i_r = stats.mean(lista_i_r)
print('El número medio de iteraciones es', media_i_r, '\n\n')
```

```
##########################
#                        #
#       Ejercicio 2      #
#                        #
##########################

print("*******************************************\n")
print('*** EJERCICIO 2 --- REGRESIÓN LOGÍSTICA ***')
input('*******************************************\n')

--------------------------------
|                              |
|          Apartado a)         |
|                              |
--------------------------------

input(" -- apartado a)")
print("_______________")
print('Implementar el algoritmo de Regresión Logística con Gradiente Descendente Estocástico con una nube de puntos de tamaño 100, cuyos valores estarán entre 0 y 2, y cuya etiqueta será 0 ó 1.')
print('El vector de pesos debe estar inicializado a 0, la condición de parada será cuando ||w (t-1) - w (t) || < 0.01, y la tasa de aprendizaje 0.01')

np.random.seed(2)

#pinta_recta
# --- Parámetros:
#       - w: vector de pesos obtenidos tras la regresión
#       - i: un índice que determina si queremos pintar la línea con leyenda o sin ella
#           i = 0: la pintamos con leyenda
#           i = 1: la pintamos sin leyenda
#
# --- Retorno:
#       Dibuja la recta divisioria
def pinta_recta_reg(w, i):

    #obtenemos el valor de la pendiente
    a = (-w[2]/w[1])/(w[2]/w[0])

    #obtenemos el valor de la constante
    b = (-w[2]/w[1])

    axes = plt.gca()                  #obtengo los ejes
    val_x = np.array(axes.get_xlim()) #los valores del eje x
    val_y = b + a * val_x             #los valores del eje y

    if(i == 0):
        #pintamos la gráfica
        plt.plot(val_x, val_y, '-', label='Recta divisoria', c='green')
        plt.legend()

    else:
        plt.plot(val_x, val_y, '-', c='green')       #pintamos la gráfica
```

```python
#sigmoide
# --- Parámetros:
#       - x: la coordenada x de un punto, es un valor
#
# --- Retorno:
#       Devuelve el valor de la función sigmoide del valor x
def sigmoide(x):
    return 1/(1+np.exp(-x))


#h(x,w)
# --- Parámetros:
#       - x: es un punto, con 2 coordenadas
#       - w: el vector de pesos
#
# --- Retorno:
#     El resultado de la función sigmoide aplicada al producto vectorial de pesos por x
def h(x,w):
      return sigmoide(np.dot(x,w))


#estocastico
# --- Parámetros:
#       - x: un punto con 2 coordenadas
#       - y: etiqueta de dicho punto
#       - w: vector de pesos del algoritmo de regresión
#
# --- Retorno:
#       El valor del gradiente descendente estocástico en dicho punto con su etiqueta
#       y con el vector de pesos correspondiente
def estocastico(x,y,w):
    return (1/x.shape[0])*x.T.dot(h(x,w)-y)


#error_reg
# --- Parámetros:
#       - x: un punto con 2 coordenadas
#       - y: etiqueta de dicho punto
#       - w: vector de pesos del algoritmo de regresión
#
# --- Retorno:
#       Calcula el error logístico
def error_reg(x, y, w):
    mat = np.dot(x, w)
    return np.sum(y * mat - np.log(1 + np.exp(mat)))


#regresion
# --- Parámetros:
#       - x: vector de puntos con dos columnas, una para la coordenada x y otra para la
y
#       - y: vector de etiquetas de dichos puntos, toman un valor de 0 ó 1
#       - max_iter: número máximo de iteraciones
#       - tasa: la tasa de aprendizaje
#
# --- Retorno:
#    El vector de pesos resultante y una lista de error que ha ido tomando en cada paso
def regresion(x, y, max_iter, tasa):

    #vector de pesos iniciales inicializados a 0
    w = np.zeros(3, np.float64) ; w_ant = np.zeros(3, np.float64)

    #obtengo 100 números del 0 al 100 --> (0 1 2 3 4 ... 98 99)
    #estos serán los índices que cogeré para cada época
    orden = np.array(range(0, x.shape[0]))

    i = 0   #iteraciones a 0
    lista_error = []    #inicialización de la lista de errores
```

```
    #mientras no supere el número máximo de iteraciones
    while i < max_iter:

        #desordeno el orden en el que cogeré los puntos
        np.random.shuffle(orden)

        #almaceno en epc_x aquellos elementos cuya fila sea el número que se encuentra
        # en orden, y me quedo con los 50 primeros. Hago lo mismo para epc_y
        epc_x = x[orden] ; epc_x = x[:50]
        epc_y = y[orden] ; epc_y = y[:50]

        #recorro epc_x y calculo el vector de pesos
        for j in range(0, epc_x.shape[0]):
            w = w - tasa*estocastico(epc_x[j], epc_y[j], w)

        #almaceno el error resultante en la lista
        lista_error.append(error_reg(epc_x, epc_y, w))

        #condición de parada del algoritmo
        if(np.linalg.norm(w_ant - w) < 0.01):
            return w, lista_error

        w_ant = w
        i = i + 1

    return w, lista_error

#el intervalo de nuestros puntos estarán entre 0 y 2, y tendremos 100 puntos
intervalo_reg = [0,2] ; N_reg = 100

#obtengo la pendiente y el término independiente en el intervalo [0,2]
a, b = simula_recta(intervalo_reg)

#obtengo los 100 puntos con la función simula_unif y le añado una columna de 1
datos_reg = simula_unif(N_reg, dim, intervalo_reg) ; datos_reg = np.c_[datos_reg,
np.ones(datos_reg.shape[0])]

#en etq_reg almaceno las etiquetas de dichos puntos
etq_reg = np.sign(datos_reg[:,1] - a*datos_reg[:,0] - b)

#como la etiqueta debe ser 0 ó 1, aquellas que sean -1 las cambio por 0
etq_reg[etq_reg == -1.0] = 0

#obtenemos los datos
w_reg, list_err = regresion(datos_reg, etq_reg, 10000, 0.01)


#pinto la gráfica
input('Comprobación que el algoritmo de Regresión funciona correctamente')
plt.title('Gráfica del ejercicio 2a de Regresión Logística')

#primero selecciono aquellos datos cuya etiqueta sea negativa y los pinto morado
plt.scatter(datos_reg[etq_reg == 0, 0], datos_reg[etq_reg == 0, 1], c='purple',
label='Etiqueta 0')

#después selecciono aquellos datos cuya etiqueta sea positiva y los pinto de naranja
plt.scatter(datos_reg[etq_reg == 1, 0], datos_reg[etq_reg == 1, 1], c='orange',
label='Etiqueta 1')

plt.xlabel('Eje x')
plt.ylabel('Eje y')

#pinto la recta con el primer resultado de la lista con etiqueta
pinta_recta_reg(w_reg,0)
plt.show()
```

```
--------------------------------
|                              |
|          Apartado b)         |
|                              |
--------------------------------

input(" -- apartado b)")
print("———————————————")
print('Estimar el error medio para 1000 muestras, con datos de 100 elementos.\n')

errores = []

#en un for se realiza el cálculo del error y lo almaceno en una lista
for i in range(0, 1000):
    ptos = simula_unif(N_reg, dim, intervalo_reg) ; ptos = np.c_[ptos,
                       np.ones(ptos.shape[0])]
    etqs = np.sign(datos_reg[:,1] - a*datos_reg[:,0] - b)
    etqs[etqs == -1.0] = 0

    errores.append(error_reg(ptos, etqs, w_reg))

#calculo la media del error
media_reg = stats.mean(errores)

print('El error medio es', media_reg)

#pinto el error en una gráfica
plt.title('Error medio de Regresión para datos de 100 elementos en 1000 muestras')
plt.xlabel('Iteraciones')
plt.ylabel('Error')
plt.plot(errores)
plt.show()
```