PRACTICA 2 AA

April 19, 2019

0.0.1 IGNACIO MORILLAS PADIAL

0.0.2 TRABAJO 2 AA

ESTE INFORME LO HE REALIZADO CON JUPITER QUE ME RESULTA MAS COMODO A LA HORA DE REALIZARLO

1 1. Ejercicio sobre la complejidad de H y el ruido (6 puntos)

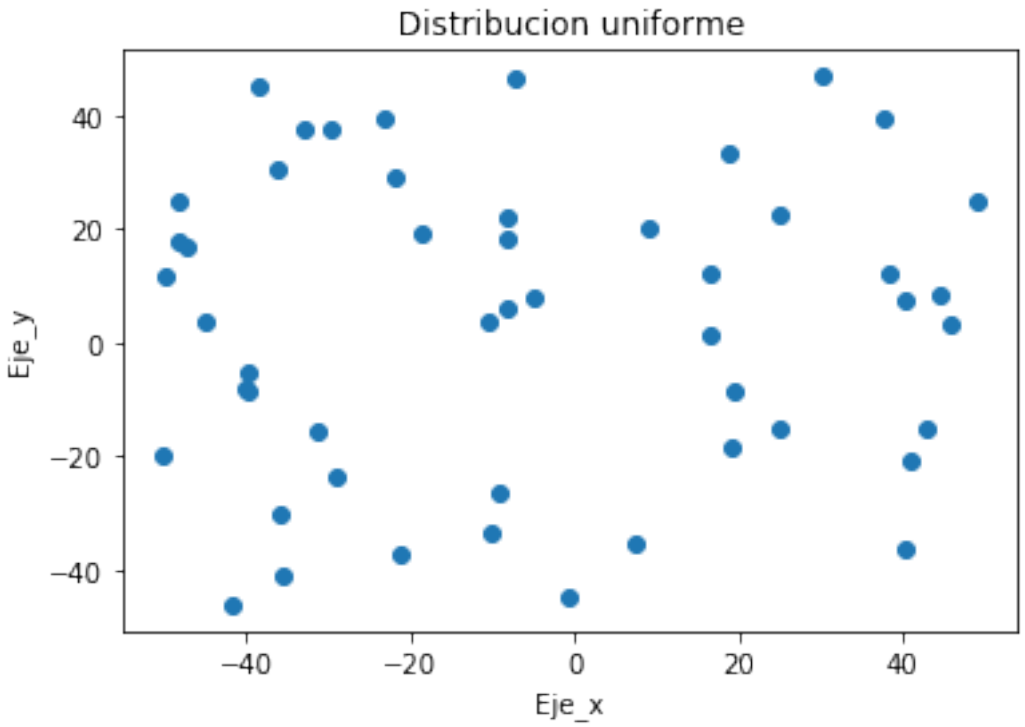
In [690]:
```
import numpy as np
import matplotlib.pyplot as plt


# Fijamos la semilla
np.random.seed(1)


def simula_unif(N, dim, rango):
        return np.random.uniform(rango[0],rango[1],(N,dim))

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 (eje X) se usará una N(0,sqrt(sigma[0]))
        # y para la segunda (eje Y) N(0,sqrt(sigma[1]))
        out[i,:] = np.random.normal(loc=media, scale=np.sqrt(sigma), size=dim)

    return out


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
```

1. (1 punto) Dibujar una gráca con la nube de puntos de salida correspondiente. a) Considere N = 50, dim = 2, rango = [50,+50] con simula_unif(N,dim,rango). b) Considere N = 50, dim = 2 y sigma = [5,7] con simula_gaus(N,dim,sigma).

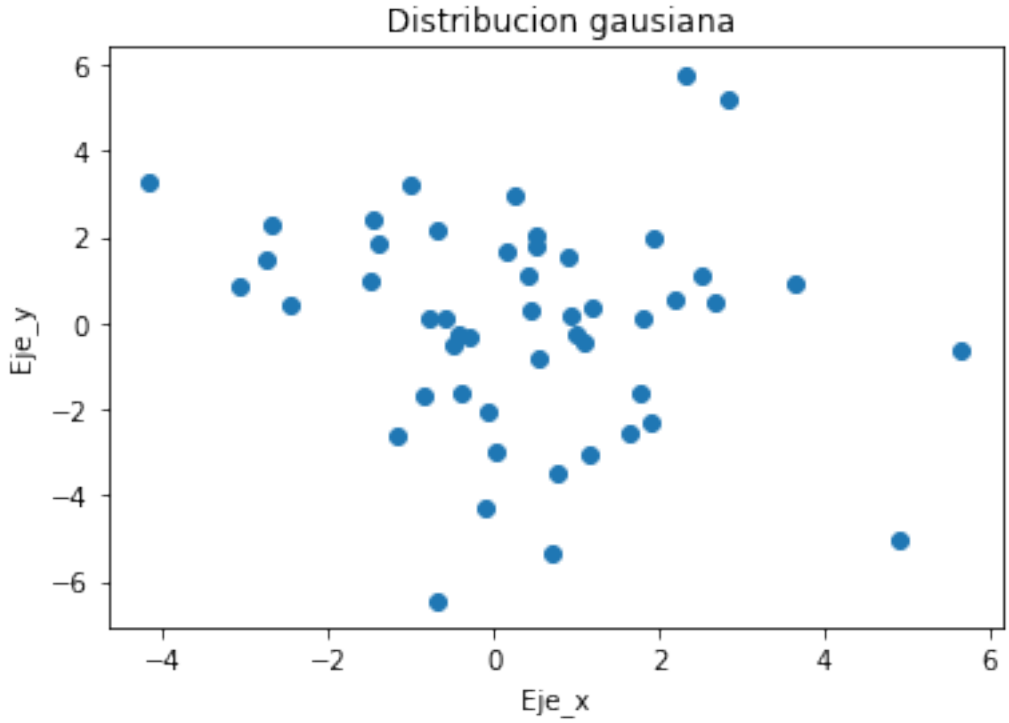
a) Considere N = 50, dim = 2, rango = [50,+50] con simula_unif(N,dim,rango).

In [691]:
```
# EJERCICIO 1.1: Dibujar una gráfica con la nube de puntos de salida correspondiente

x = simula_unif(50, 2, [-50,50])
#CODIGO DEL ESTUDIANTE

plt.title ( " Distribucion uniforme " )
plt.xlabel ( " Eje_x " )
plt.ylabel ( " Eje_y " )
plt.scatter (x [:, 0 ], x [:, 1 ])
plt.show ()
```



b) Considere N = 50, dim = 2 y sigma = [5,7] con simula_gaus(N,dim,sigma).

```
In [692]: x_g = simula_gaus(50, 2, np.array([5,7]))
          #CODIGO DEL ESTUDIANTE

          plt.title ( " Distribucion gausiana" )
          plt.xlabel ( " Eje_x " )
          plt.ylabel ( " Eje_y " )
          plt.scatter (x_g [:, 0 ], x_g [:, 1 ])
          plt.show ()
```



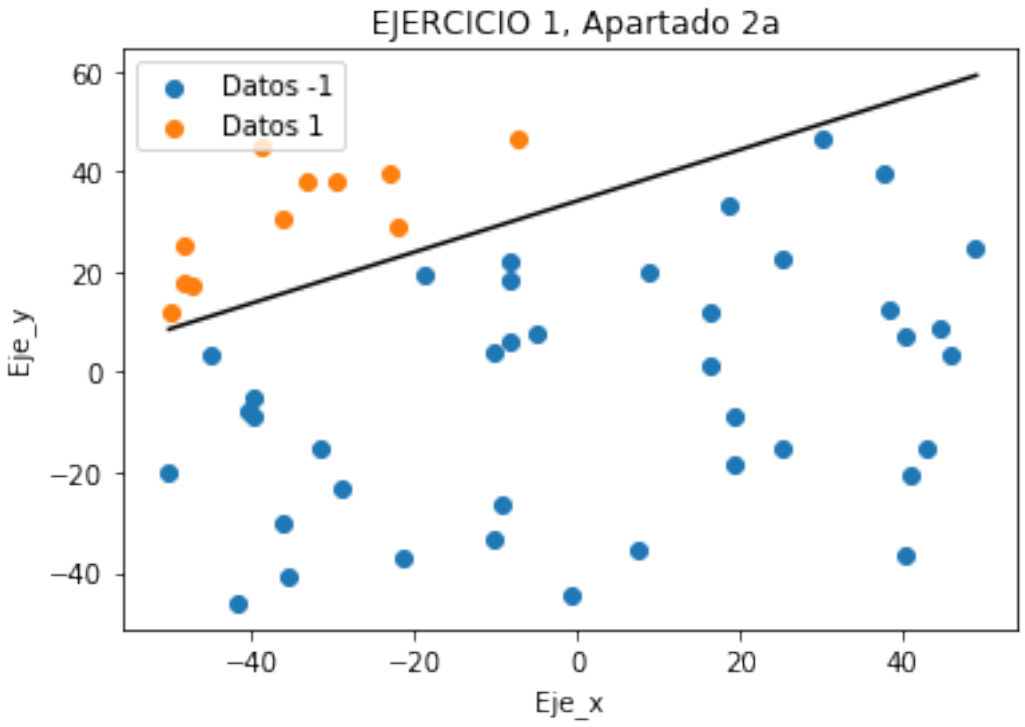
2. (2 puntos) Con ayuda de la función simula_unif() generar una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función f(x,y) = yaxb, es decir el signo de la distancia de cada punto a la recta simulada con simula_recta().

```
In [693]: # EJERCICIO 1.2: Dibujar una gráfica con la nube de puntos de salida correspondiente

          # La funcion np.sign(0) da 0, lo que nos puede dar problemas
          def signo(x):

                  if x >= 0:
                          return 1
                  return -1
```

```
def f(x, y, a, b):

    return signo(y - a*x - b)
```

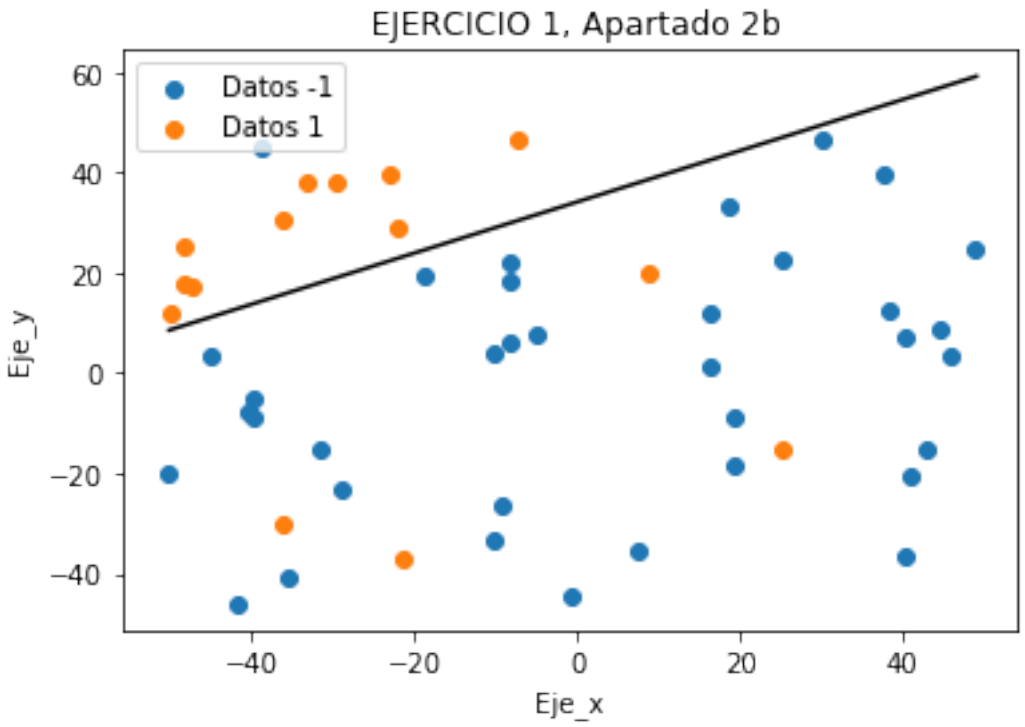
a) Dibujar una gráca donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello. (Observe que todos los puntos están bien clasicados respecto de la recta)

```
In [694]: a,b = simula_recta([-50,50])
          signo=np.vectorize(signo)
          Y = f(x[:, 0 ], x[:, 1 ], a, b)
          G = np.copy(Y)
          colores=['red','blue']
          for i in np.unique(Y):
              pos= Y == i
              x_aux = x[pos,:]
              plt.scatter(x_aux[:,0],x_aux[:,1],label='Datos '+str(i))


          plt.title ( " EJERCICIO 1, Apartado 2a" )
          plt.xlabel ( " Eje_x " )
          plt.ylabel ( " Eje_y " )
          k = range(-50,50)
          plt.plot( k, [a*i+b for i in k],c='black')
          plt.legend()
          plt.show ()
```



b) Modique de forma aleatoria un 10% etiquetas positivas y otro 10% de negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráca anterior. (Ahora hay puntos mal clasicados respecto de la recta)

```
In [695]: idx = np.random.choice(range(Y.shape[0]), size=(int(Y.shape[0]*0.1)), replace=True)
          Y[idx] *= -1

          colores=['red','blue']
          for i in np.unique(Y):
              pos= Y == i
              x_aux = x[pos,:]
              plt.scatter(x_aux[:,0],x_aux[:,1],label='Datos '+str(i))

          plt.title ( " EJERCICIO 1, Apartado 2b" )
          plt.xlabel ( " Eje_x " )
          plt.ylabel ( " Eje_y " )
          k = range(-50,50)
          plt.plot( k, [a*i+b for i in k],c='black')
          plt.legend()
          plt.show ()
```

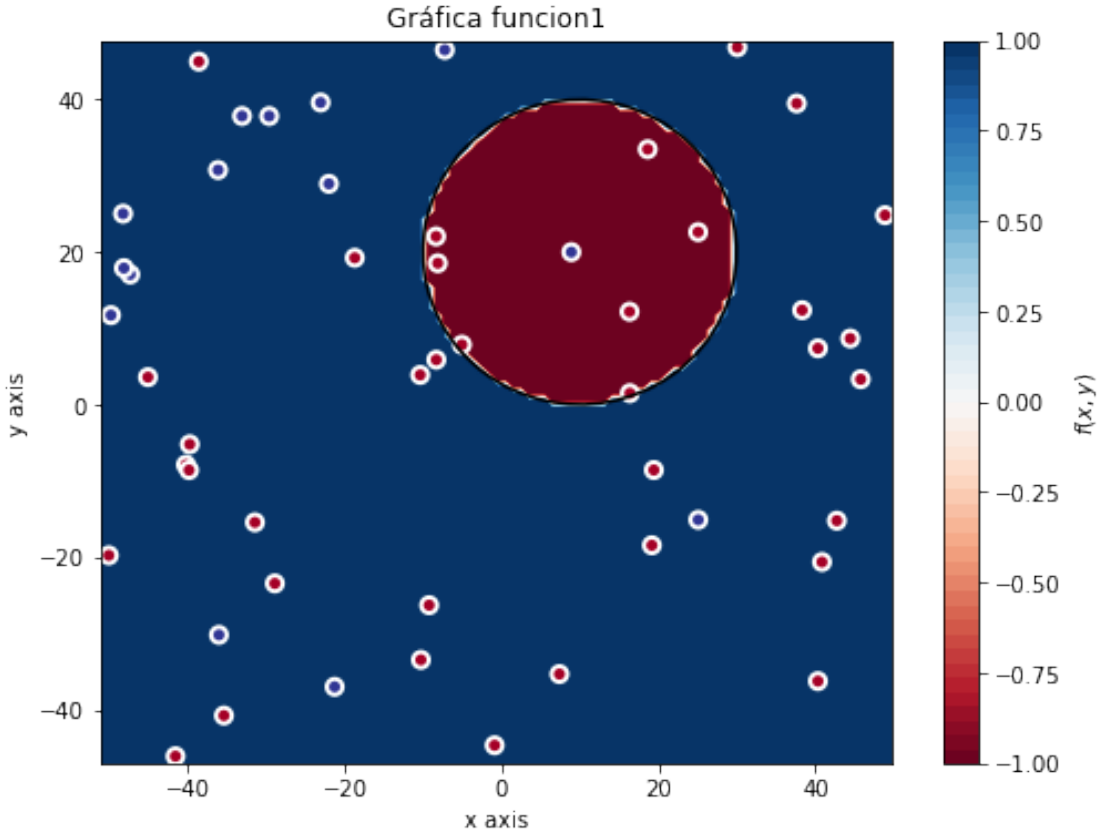


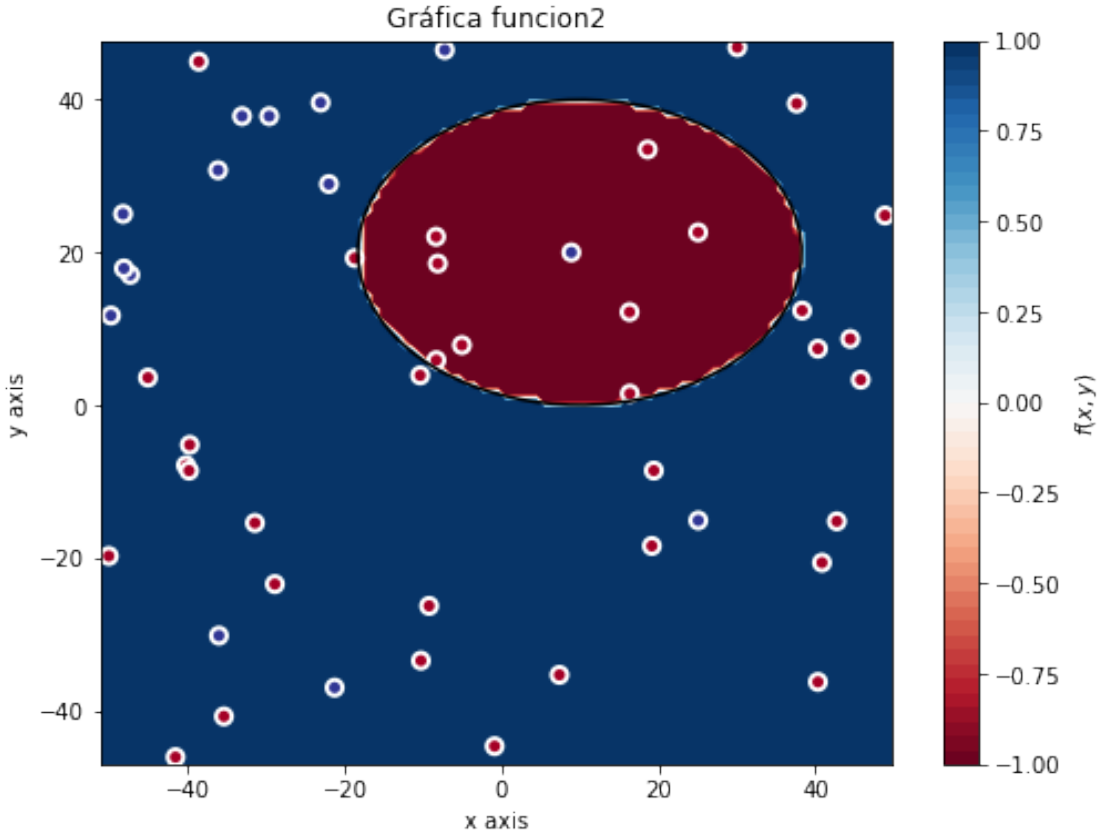
3. (3 puntos) Supongamos ahora que las siguientes funciones denen la frontera de clasicación de los puntos de la muestra en lugar de una recta. f(x,y) = (x10)2 + (y20)2 400
f(x,y) = 0,5(x + 10)2 + (y20)2 400
f(x,y) = 0,5(x10)2 (y + 20)2 400
f(x,y) = y20x2 5x + 3

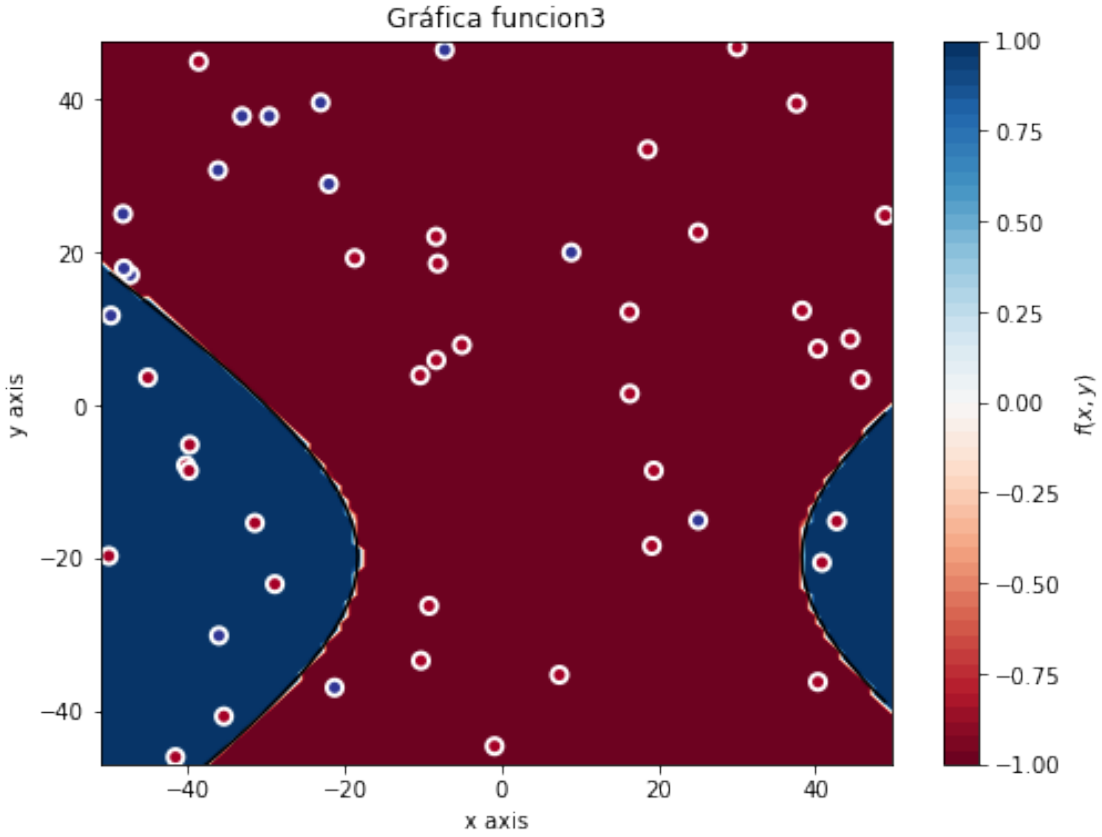
Visualizar el etiquetado generado en 2b junto con cada una de las grácas de cada una de las funciones. Comparar las formas de las regiones positivas y negativas de estas nuevas funciones con las obtenidas en el caso de la recta £Son estas funciones más complejas mejores clasicadores que la función lineal? £En que ganan a la función lineal? Explicar el razonamiento.

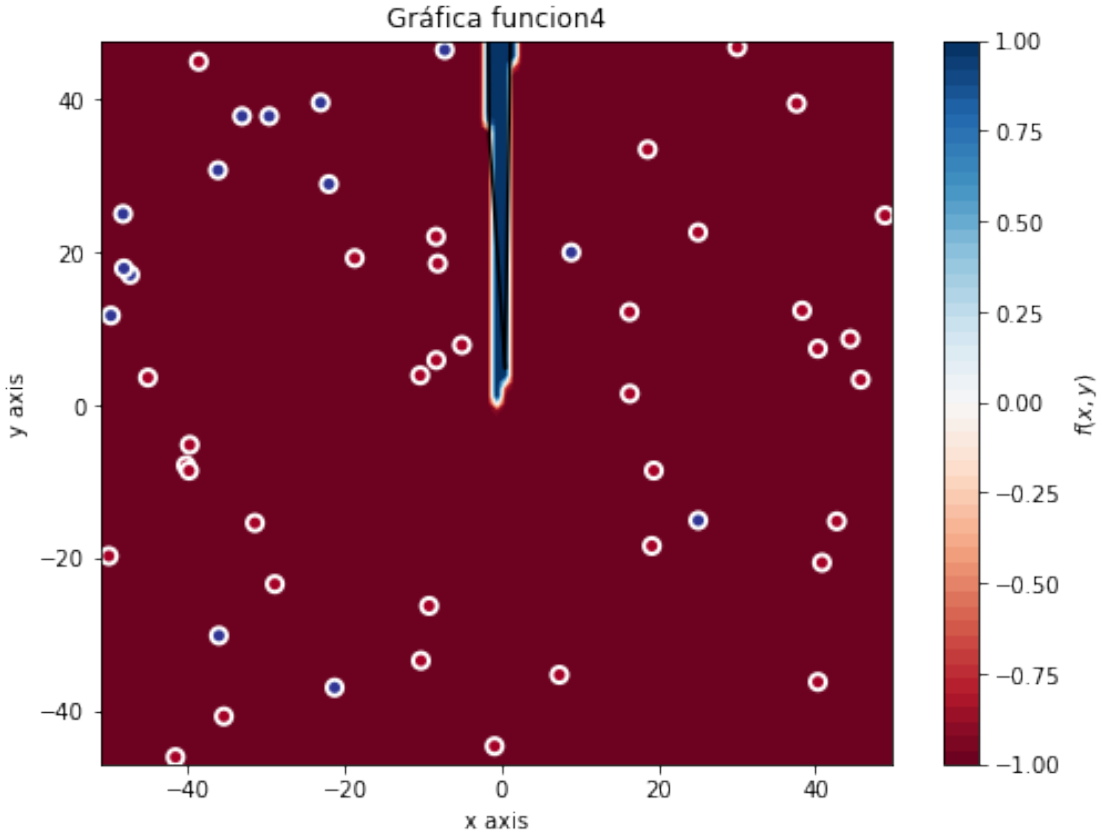
```
In [696]: def plot_datos_cuad(X, y, fz, title='Point cloud plot', xaxis='x axis', yaxis='y axis
              #Preparar datos
              min_xy = X.min(axis=0)
              max_xy = X.max(axis=0)
              border_xy = (max_xy-min_xy)*0.01

              #Generar grid de predicciones
              xx, yy = np.mgrid[min_xy[0]-border_xy[0]:max_xy[0]+border_xy[0]+0.001:border_xy[0
                                min_xy[1]-border_xy[1]:max_xy[1]+border_xy[1]+0.001:border_xy[1
              grid = np.c_[xx.ravel(), yy.ravel(), np.ones_like(xx).ravel()]
              pred_y = fz(grid)
              # pred_y[(pred_y>-1) & (pred_y<1)]
              pred_y = np.clip(pred_y, -1, 1).reshape(xx.shape)

              #Plot
              f, ax = plt.subplots(figsize=(8, 6))
              contour = ax.contourf(xx, yy, pred_y, 50, cmap='RdBu',vmin=-1, vmax=1)
              ax_c = f.colorbar(contour)
              ax_c.set_label('$f(x, y)$')
              ax_c.set_ticks([-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1])
              ax.scatter(X[:, 0], X[:, 1], c=y, s=50, linewidth=2,
                         cmap="RdYlBu", edgecolor='white')

              XX, YY = np.meshgrid(np.linspace(round(min(min_xy)), round(max(max_xy)),X.shape[0
              positions = np.vstack([XX.ravel(), YY.ravel()])
              ax.contour(XX,YY,fz(positions.T).reshape(X.shape[0],X.shape[0]),[0], colors='blac

              ax.set(
                 xlim=(min_xy[0]-border_xy[0], max_xy[0]+border_xy[0]),
                 ylim=(min_xy[1]-border_xy[1], max_xy[1]+border_xy[1]),
                 xlabel=xaxis, ylabel=yaxis)
              plt.title(title)
              plt.show()

In [697]: #CODIGO DEL ESTUDIANTE

          def f1(X):
```

```
        return ((X[:, 0]-10)**2) + ((X[:, 1]-20)**2)-400

def f2(X):
        return (0.5*(X[:, 0]-10)**2) + ((X[:, 1]-20)**2)-400

def f3(X):
        return (0.5*(X[:, 0]-10)**2) - ((X[:, 1]+20)**2)-400

def f4(X):
        return X[:, 1]-20*(X[:, 0]**2)-5*X[:, 0]+3


plot_datos_cuad(x, Y,f1,'Gráfica funcion1')
plot_datos_cuad(x, Y,f2,'Gráfica funcion2')
plot_datos_cuad(x, Y,f3,'Gráfica funcion3')
plot_datos_cuad(x, Y,f4,'Gráfica funcion4')
```




Gráfica funcion2
40
20
0
−20
−40
y axis
−40
−20
0
20
40
x axis
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
f(x, y)


Gráfica funcion3
40
20
0
−20
−40
y axis
−40
−20
0
20
40
x axis
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
f(x, y)



Al ver estas gráficas concluimos que estas funciones son peores clasificadoras para esta muestra que la recta anteriormente planteada. Esto se debe a que las funciones planteadas son cónicas y la distribucion de la muestra no sigue una distribucion adecuada para el uso de este tipo de funciones a la hora de clasificar.

Con una muestra mas cercana a alguna de las representaciones anteriores, estas serían mas certeras que la recta.

2 2. Modelos Lineales

1. (2.5 puntos) Algoritmo Perceptron: Implementarlafunciónajusta_PLA(datos,label,max_iter,vini) que calcula el hiperplano solución a un problema de clasicación binaria usando el algoritmo PLA. La entrada datos es una matriz donde cada item con su etiqueta está representado por una la de la matriz, label el vector de etiquetas (cada etiqueta es un valor +1 o 1), max_iter es el número máximo de iteraciones permitidas y vini el valor inicial del vector. La función devuelve los coecientes del hiperplano.

```
In [698]: def coef2line(w):
              if(len(w)!= 3):
                  raise ValueError('Solo se aceptan rectas para el plano 2d. Formato: [<a0>, <a
```

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

    return a, b


def plot_data(X, y, w,title='Point clod plot',x_l='x axis',y_l='y axis'):
    #Preparar datos
    a, b = coef2line(w)
    min_xy = X.min(axis=0)
    max_xy = X.max(axis=0)
    border_xy = (max_xy-min_xy)*0.01

    #Generar grid de predicciones
    xx, yy = np.mgrid[min_xy[0]-border_xy[0]:max_xy[0]+border_xy[0]+0.001:border_xy[0
                      min_xy[1]-border_xy[1]:max_xy[1]+border_xy[1]+0.001:border_xy[1
    grid = np.c_[xx.ravel(), yy.ravel(), np.ones_like(xx).ravel()]
    pred_y = grid.dot(w)
    pred_y = np.clip(pred_y, -1, 1).reshape(xx.shape)

    #Plot
    f, ax = plt.subplots(figsize=(8, 6))
    contour = ax.contourf(xx, yy, pred_y, 50, cmap='RdBu',
                          vmin=-1, vmax=1)
    ax_c = f.colorbar(contour)
    ax_c.set_label('$w^tx$')
    ax_c.set_ticks([-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1])
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, linewidth=2,
                cmap="RdYlBu", edgecolor='white', label='Datos')
    ax.plot(grid[:, 0], a*grid[:, 0]+b, 'black', linewidth=2.0, label='Solucion')
    ax.set(
       xlim=(min_xy[0]-border_xy[0], max_xy[0]+border_xy[0]),
       ylim=(min_xy[1]-border_xy[1], max_xy[1]+border_xy[1]),
       xlabel=x_l, ylabel=y_l)
    ax.legend()
    plt.title(title)
    plt.show()
```

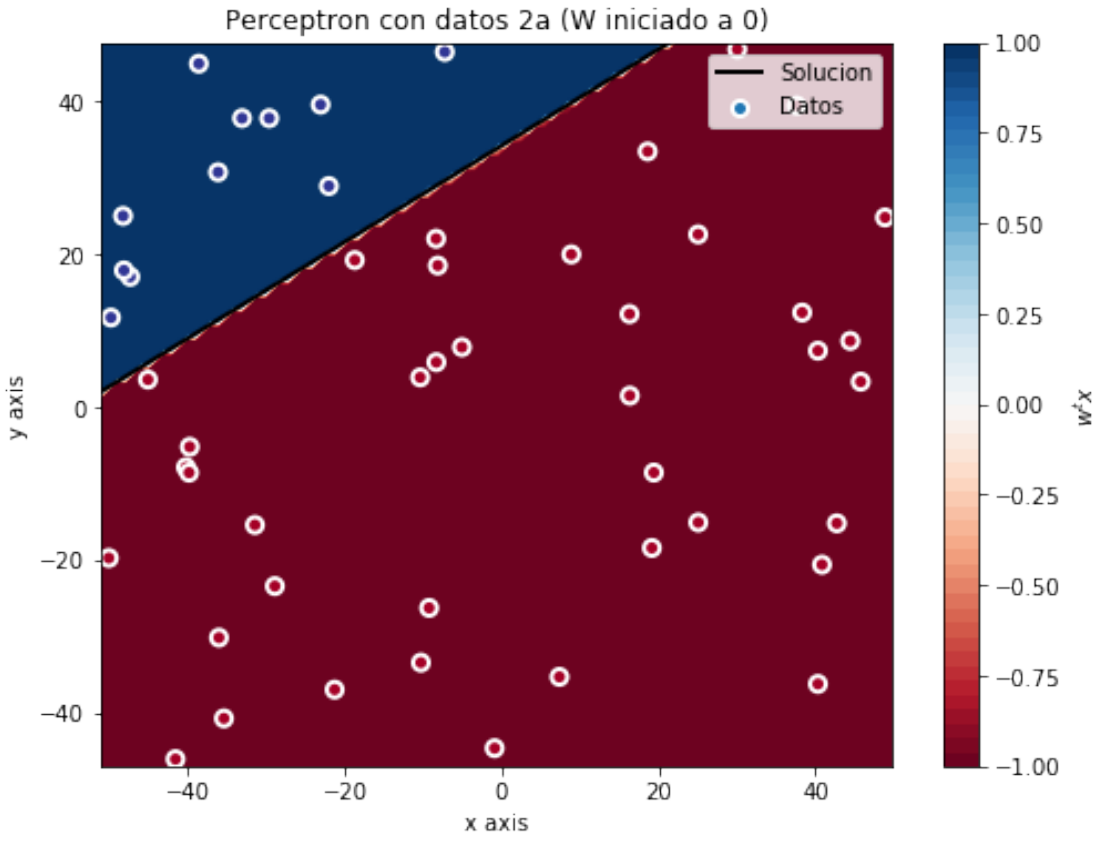
a) Ejecutar el algoritmo PLA con los datos simulados en los apartados 2a de la sección.1. Inicializar el algoritmo con: a) el vector cero y, b) con vectores de números aleatorios en [0,1] (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado relacionando el punto de inicio con el número de iteraciones.

```
In [699]: def ajusta_PLA(datos, label, max_iter, vini):
              total_iters=0
              cambios=True
              ncambios=0
              w=np.copy(vini)
```

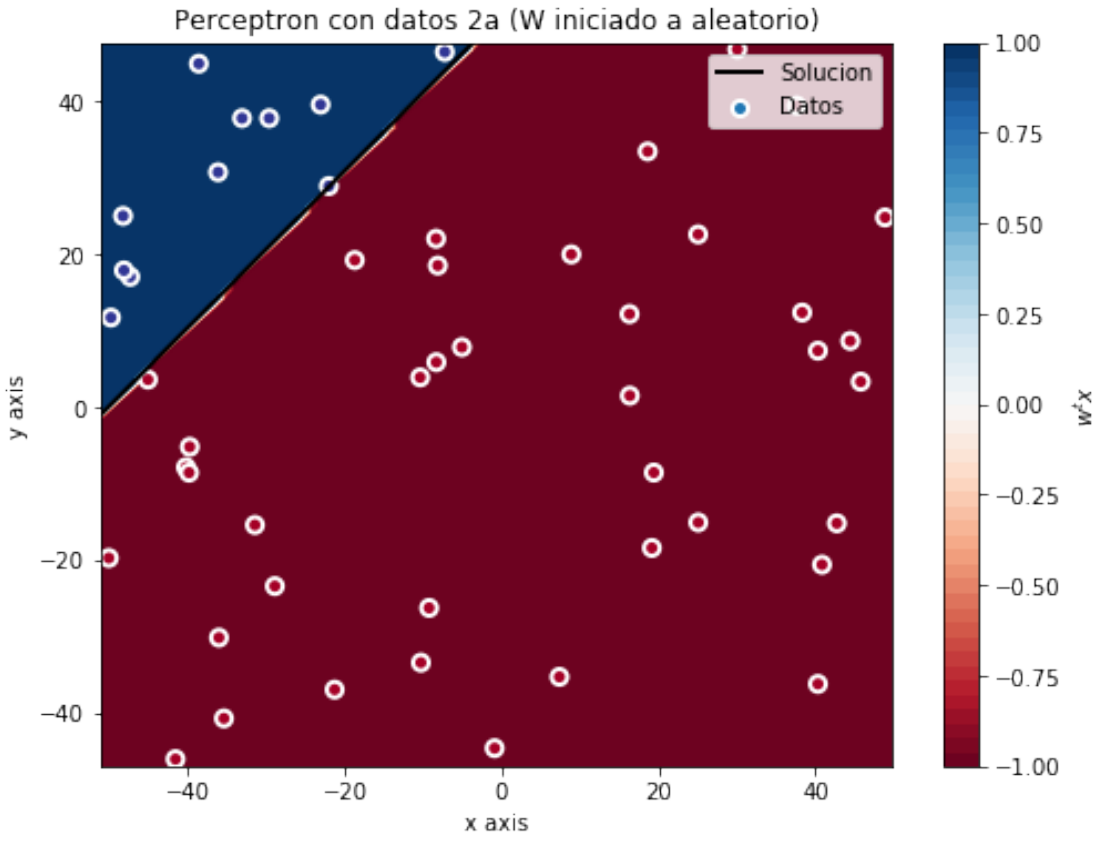
```
                for i in range (0,max_iter):
                        cambios=False
                        total_iters+=1

                        for x,y in zip(datos,label):
                                if np.sign(np.dot(np.transpose(w),x))!=np.sign(y):
                                        w=w+(y*x)
                                        ncambios+=1
                        if (ncambios>0):
                                ncambios=0
                                cambios=True

                        if not cambios:
                                break
                return w,total_iters
```

In [700]:
```
#CODIGO DEL ESTUDIANTE
print ("\n\nDATOS NO MODIFICADOS\n\n")
x1 = np.copy(x)
x1=np.concatenate((x1,np.ones((x1.shape[0],1),dtype=np.float64)),1)

w = np.zeros(3)

w,iters=ajusta_PLA(x1,G,1000,w)
plot_data(x1,G,w,'Perceptron con datos 2a (W iniciado a 0)')
print ("Total de iteraciones para vector iniciado a 0: ",iters)
total=0
for i in range (0,10):
        w=np.random.uniform(0,1,x1.shape[1])
        w,iters=ajusta_PLA(x1,G,1000,w)
        total+=iters
plot_data(x1,G,w,'Perceptron con datos 2a (W iniciado a aleatorio)')
print('Valor medio de iteraciones necesario para converger: {}'.format(np.mean(np.asa
```

```
DATOS NO MODIFICADOS
```




```
Total de iteraciones para vector iniciado a 0:  35
```




```
Valor medio de iteraciones necesario para converger: 70.1
```

Como podemos apreciar, puesto que en este caso, la muestra es lineamente separable y el PLA converge hacia un hiperplano que separa perfectamente la muestra.

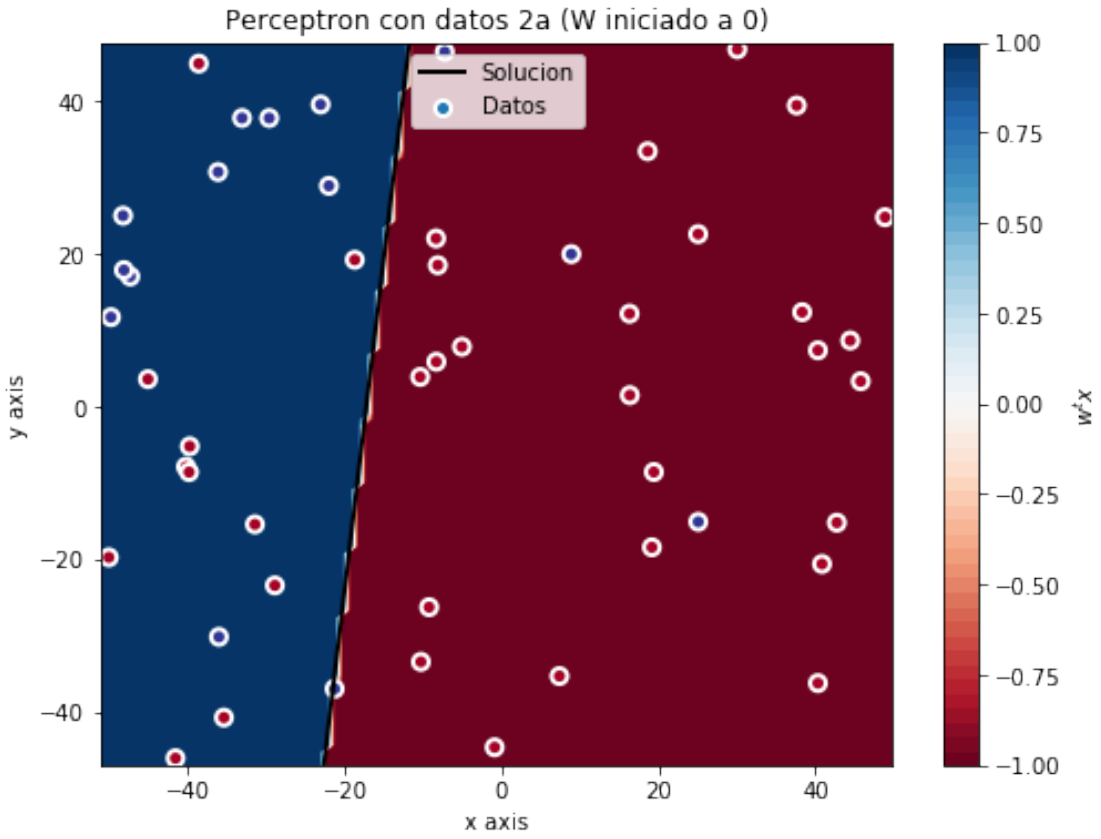
Como vemos, el algoritmo PLA es capaz de obtener una solucion que divide a la muestra perfectamente en 35 iteraciones inicializando el vector a 0. Si iniciamos el vector de forma aleatoria, puede que la solucion inicial caiga muy lejos del objetivo por lo que el algoritmo necesitaria un numero mas elevado de iteraciones para converger, en nuestro caso 70

b) Hacer lo mismo que antes usando ahora los datos del apartado 2b de la sección.1. £Observa algún comportamiento diferente? En caso armativo diga cual y las razones para que ello ocurra.

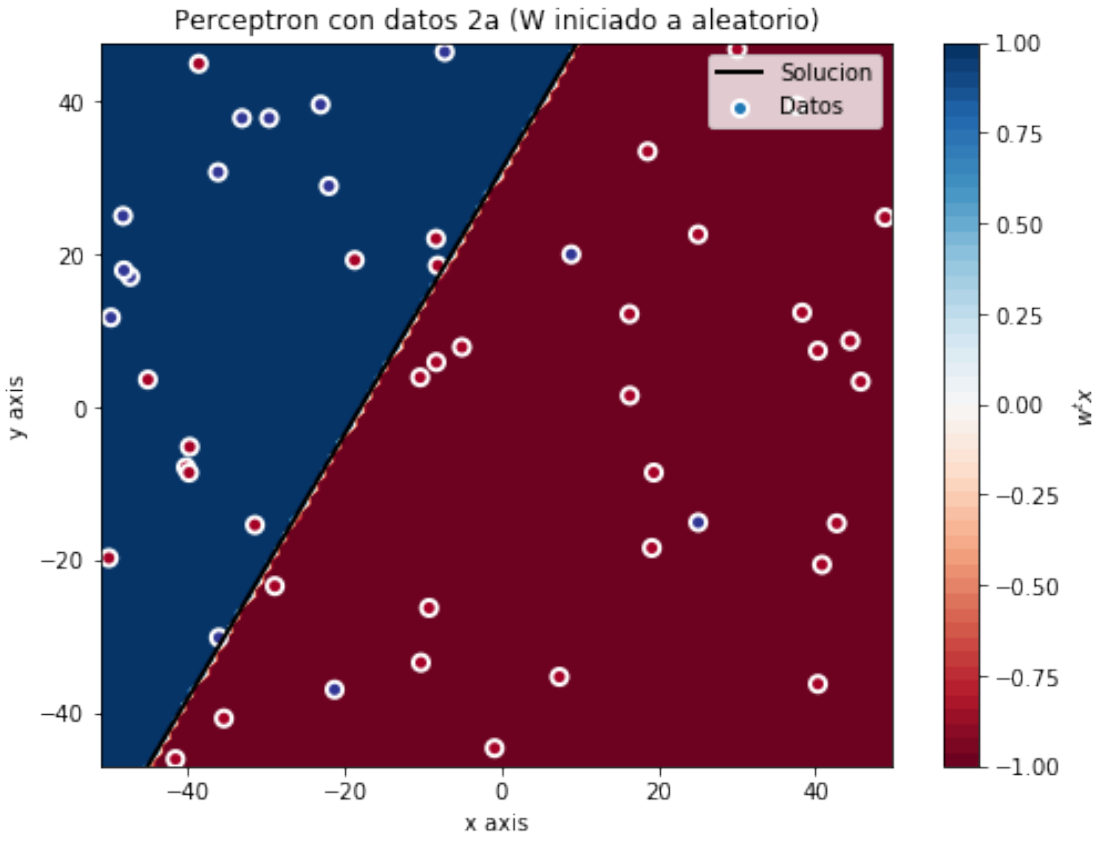
```
In [701]: #CODIGO DEL ESTUDIANTE
          print ("\n\nDATOS MODIFICADOS\n\n")
          x2=np.copy(x)
          x2=np.concatenate((x2,np.ones((x2.shape[0],1),dtype=np.float64)),1)
          w = np.zeros(3)
          w,iters=ajusta_PLA(x2,Y,1000,w)
          plot_data(x2,Y,w,'Perceptron con datos 2a (W iniciado a 0)')
```

```
print ("Total de iteraciones para vector iniciado a 0: ",iters)
total=0
iters=0
for i in range (0,10):
    w=np.random.uniform(0,1,x2.shape[1])
    w,iters=ajusta_PLA(x2,Y,1000,w)
    total+=iters
plot_data(x2,Y,w,'Perceptron con datos 2a (W iniciado a aleatorio)')
print ("Media de iteraciones para vector iniciado a valores aleatoros: ",total/10)
```

DATOS MODIFICADOS



Total de iteraciones para vector iniciado a 0: 1000




```
Media de iteraciones para vector iniciado a valores aleatoros:  1000.0
```

En cuanto a la media de iteraciones vemos que todas las ejecuciones llegan al maximo de iteraciones. esto se debe a que al haber introducido ruido en las etiquetas, hemos hecho que la muestra no sea linealmente separable, por lo que si ehecutamos el PLA con estos datos, el algortimo va a empezar a ciclar indefinidamente ya que como la muestra no es linealmente separable, no existe un hiperplano que separ completamente toda la muestra según las etiquetas y vuelve a repetir el cclo indefinidament, en nuestro caso apra porque tenemos unlimite puesto. Respecto a la relacion entre el punto de inicio e iteraciones, en este caso al hacer en todos los casos el maximo no podemos deducir nada.

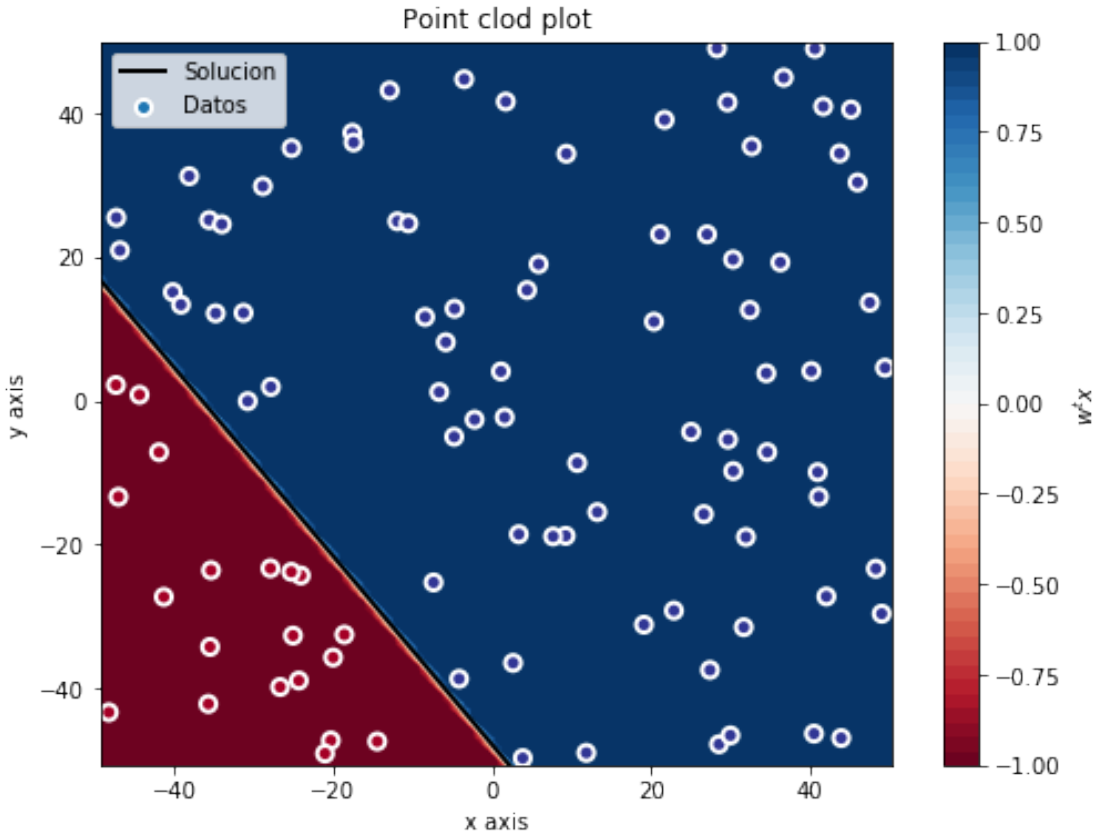
2. (3.5 puntos) Regresión Logística: En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos D para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores 0/1 y por tanto que la etiqueta y es una función determinista de x. Consideremos d = 2 para que los datos sean visualizables, y sea X = [0,2]Œ[0,2] con probabilidad uniforme de elegir cada xX. Elegir una línea en el plano que pase por X como la frontera entre f(x) = 1 (donde y toma valores +1) y f(x) = 0 (donde y toma valores 1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos. Seleccionar N = 100 puntos aleatorios {xn} de X y evaluar las respuestas {yn} de todos ellos respecto de la frontera elegida.

a) Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD) bajo las siguientes condiciones:

```
In [702]: def logisticRegression(X, y, iterations = 10000, lr = 0.01):
                  w = np.zeros(3)
                  w_ini = np.zeros(3)
                  idx = np.arange(X.shape[0])
                  for i in range(iterations):
                          np.random.shuffle(idx)
                          X = X[idx]
                          y = y[idx]
                          batchX = X[0:50:]
                          batchY = y[0:50:]
                          for j in range(batchX.shape[0]):
                                  w = w - lr * function_SGD(batchX[j], batchY[j], w)
                          if(np.linalg.norm(w_ini - w) < 0.01):
                                  return w
                          w_ini = np.copy(w)
                  return w

          def function_SGD(X,y,w):
                  a = np.dot(-y, X)
                  b = sigmoid(np.dot(np.dot(-y, np.transpose(w)), X))
                  return (a * b)/X.shape[0]

          def plot_datos_recta(X, y, a, b, title = 'Point clod plot', xaxis = 'x axis', yaxis =
                  w = line2coef(a, b)
                  min_xy = X.min(axis=0)
                  max_xy = X.max(axis=0)
                  border_xy = (max_xy-min_xy)*0.01
                  xx, yy = np.mgrid[min_xy[0]-border_xy[0]:max_xy[0]+border_xy[0]+0.001:border_
                  grid = np.c_[xx.ravel(), yy.ravel(), np.ones_like(xx).ravel()]
                  pred_y = grid.dot(w)
                  pred_y = np.clip(pred_y, -1, 1).reshape(xx.shape)
                  f, ax = plt.subplots(figsize=(8, 6))
                  contour = ax.contourf(xx, yy, pred_y, 50, cmap='RdBu', vmin=-1, vmax=1)
                  ax_c = f.colorbar(contour)
                  ax_c.set_label('$w^tx$')
                  ax_c.set_ticks([-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1])
                  ax.scatter(X[:, 0], X[:, 1], c = y, s = 50, linewidth = 2, cmap = "RdYlBu",
                  ax.plot(grid[:, 0], a*grid[:, 0]+b, 'black', linewidth = 2.0, label = 'Soluc
                  ax.set(xlim = (min_xy[0]-border_xy[0], max_xy[0]+border_xy[0]), ylim = (min_x
                  ax.legend()
                  plt.title(title)
                  plt.show()

          #Error del SGD
          def error_SGD(w, X, Y):
```

```
    error = 0.
    Y0 = np.copy(Y)
    Y0[Y == -1] = 0
    score = np.zeros(Y.shape)
    for i in range(X.shape[0]):
            value = sigmoid(np.dot(np.dot(Y0[i], np.transpose(w)), X[i]))
            score[i] = value
            if (value <= 0.5 and Y0[i] == 1) or (value > 0.5 and Y0[i] == 0):
                    error += 1.
    return error/X.shape[0], score
```

In [703]:
```
a, b = simula_recta([-50,50])
X = simula_unif(100, 2, (-50, 50))
X = np.c_[X, np.ones((X.shape[0],))]
y = np.sign(X[:, 1]-a*X[:, 0]-b)

#Se lanza la Regresion Logistica
w = logisticRegression(X, y)
#Se pintan los datos
plot_datos_recta(X, y, a, b)
#Se obtiene el error
error, y_score = error_SGD(w, X, y)
print("\nError de Clasificacion: ", error)
```

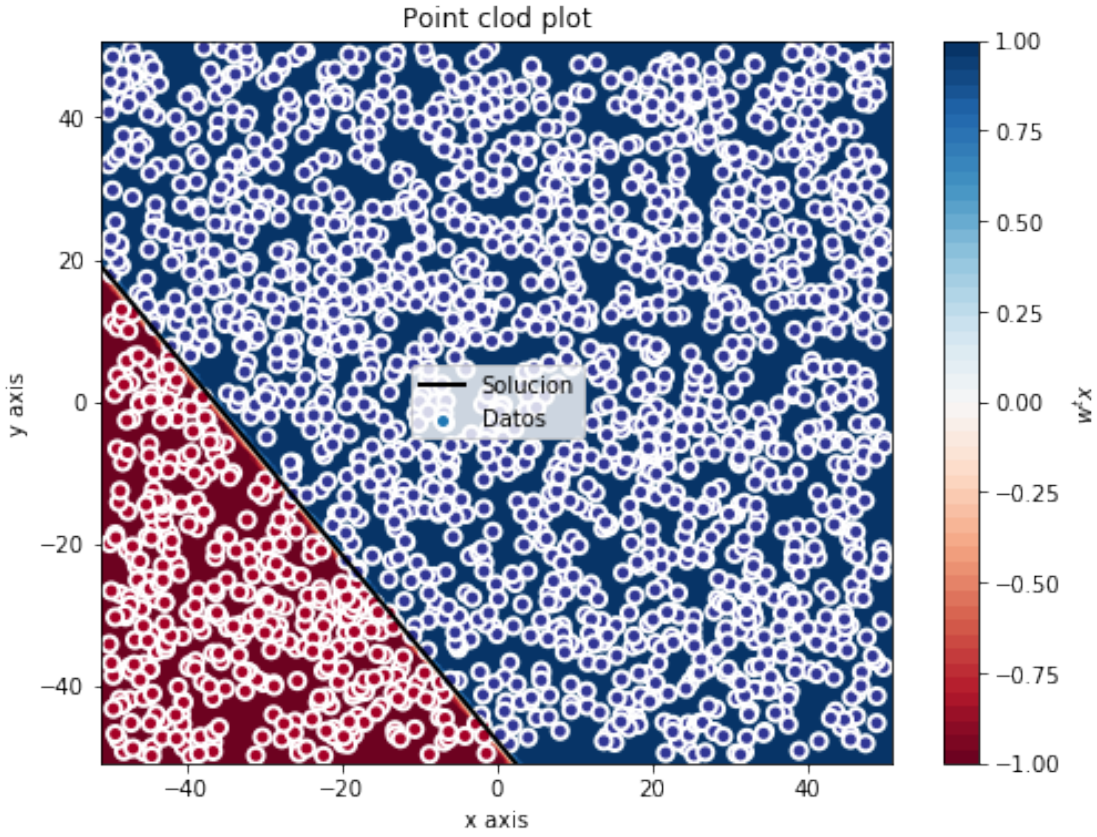


```
Error de Clasificacion:  0.02
```

En la primera grafica vemos como el GDE se acerva al resultado esperado aunque con esta tasa de aprendizaje dificilemnte conseguiremos un resultado exacto o muy bueno, dependera de la muestra.

b) Usar la muestra de datos etiquetada para encontrar nuestra solución g y estimar Eout usando para ello un número sucientemente grande de nuevas muestras (>999).

```
In [704]: #Simulamos los parametros (test) (misma funcion de etiquetado)
          X0 = simula_unif(2000, 2, (-50, 50))
          X0 = np.c_[X0, np.ones((X0.shape[0],))]
          y0 = np.sign(X0[:, 1]-a*X0[:, 0]-b)
          #Se pintan los datos
          plot_datos_recta(X0, y0, a, b)
          #Se obtiene el error
          error, y_score = error_SGD(w, X0, y0)
          print("Error de Clasificacion: ", error)
```



```
Error de Clasificacion:  0.0325
```

Vemos claramente que el error fuera de la muestra es mayor al error dentro de la muestra. Esto nos indica que nuestro modelo funciona igual de bien dentro de la muestra como fuera de ella, auqneu no hay mucha diferencia ya que entonces hubiesems realizado un sobreajuste.

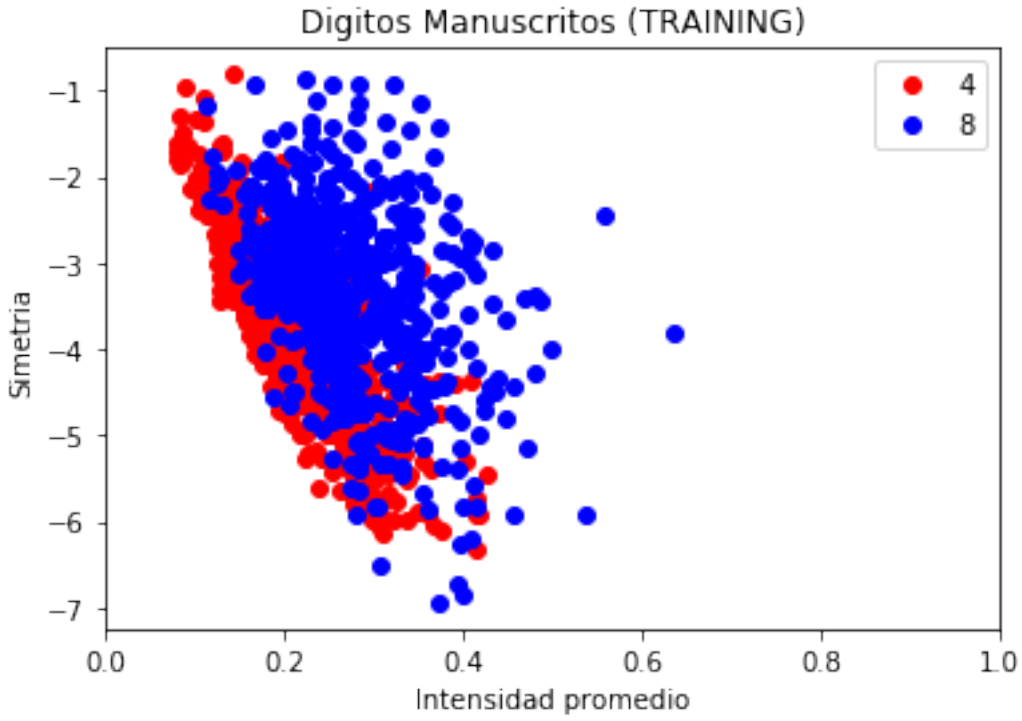
3 3.BONUS

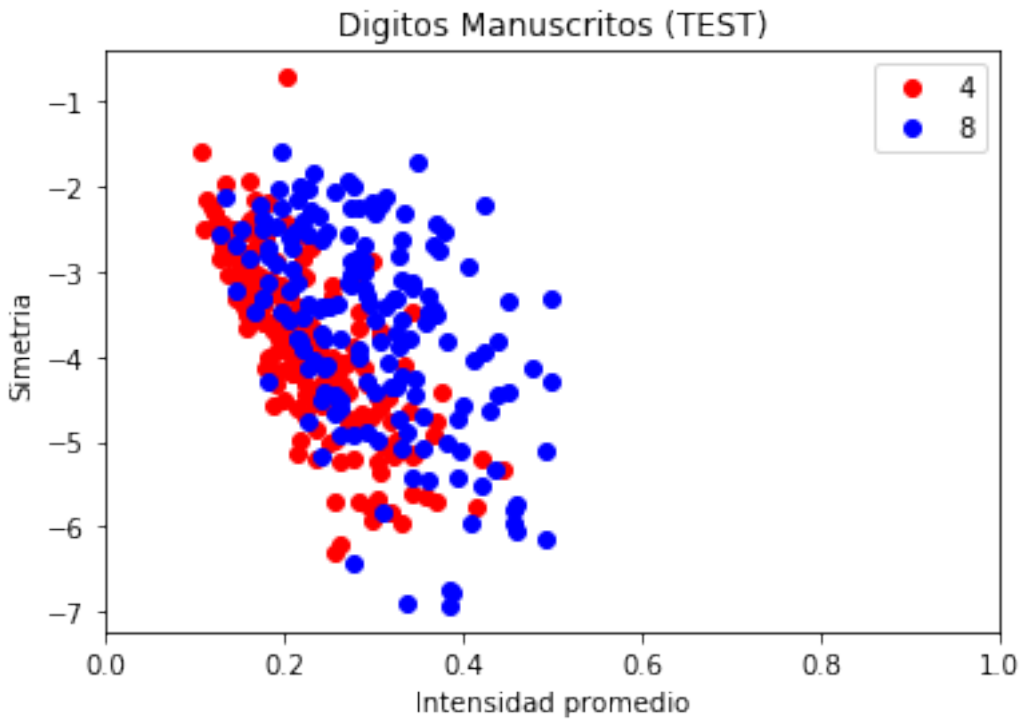
(1.5 puntos) Clasicación de Dígitos. Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 4 y 8. Usar los cheros de entrenamiento (training) y test que se proporcionan. Extraer las características de intensidad promedio y simetría en la manera que se indicó en el ejercicio 3 del trabajo 1.

1. Plantear un problema de clasicación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g.

```
In [705]: # Funcion para leer los datos
          def readData(file_x, file_y, digits, labels):
                  # Leemos los ficheros
                  datax = np.load(file_x)
                  datay = np.load(file_y)
                  y = []
                  x = []
                  # Solo guardamos los datos cuya clase sea la digits[0] o la digits[1]
                  for i in range(0,datay.size):
                          if datay[i] == digits[0] or datay[i] == digits[1]:
                                  if datay[i] == digits[0]:
                                          y.append(labels[0])
                                  else:
                                          y.append(labels[1])
                                  x.append(np.array([datax[i][0], datax[i][1],1]))

                  x = np.array(x, np.float64)
                  y = np.array(y, np.float64)

                  return x, y

          # Lectura de los datos de entrenamiento
          x, y = readData('datos/X_train.npy', 'datos/y_train.npy', [4,8], [-1,1])
          # Lectura de los datos para el test
          x_test, y_test = readData('datos/X_test.npy', 'datos/y_test.npy', [4,8], [-1,1])


          #mostramos los datos
```

```
fig, ax = plt.subplots()
ax.plot(np.squeeze(x[np.where(y == -1),0]), np.squeeze(x[np.where(y == -1),1]), 'o',
ax.plot(np.squeeze(x[np.where(y == 1),0]), np.squeeze(x[np.where(y == 1),1]), 'o', co
ax.set(xlabel='Intensidad promedio', ylabel='Simetria', title='Digitos Manuscritos (
ax.set_xlim((0, 1))
plt.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(np.squeeze(x_test[np.where(y_test == -1),0]), np.squeeze(x_test[np.where(y_te
ax.plot(np.squeeze(x_test[np.where(y_test == 1),0]), np.squeeze(x_test[np.where(y_tes
ax.set(xlabel='Intensidad promedio', ylabel='Simetria', title='Digitos Manuscritos (
ax.set_xlim((0, 1))
plt.legend()
plt.show()
```





2. Usar un modelo de Regresión Lineal y aplicar PLA-Pocket como mejora. Responder a las siguientes cuestiones.

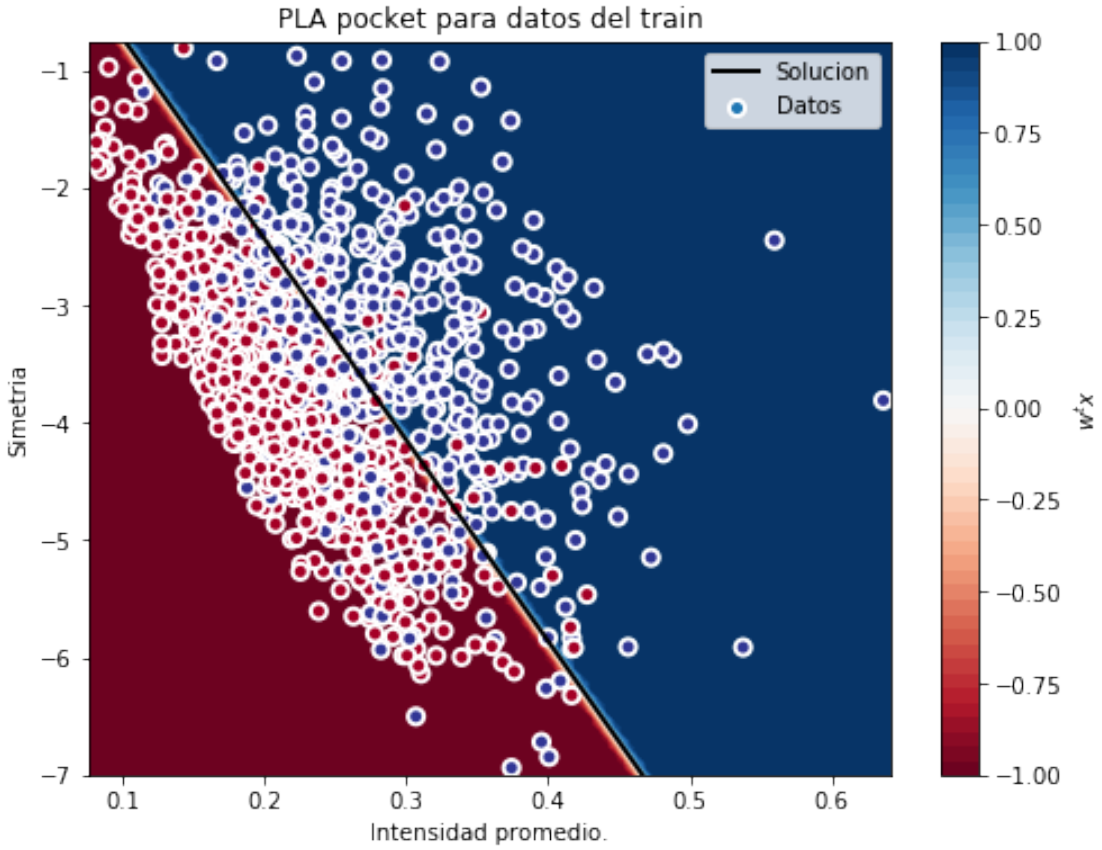
```
In [706]: """Error lineal para calcular el mejor Ein para el pla pocket"""
          def Error_lineal(X,Y,W):
                  sumatory=0

                  for i in range(X.shape[0]):
                          a=W.dot(X[i])
                          if a*Y[i]<0:
                                  sumatory=sumatory+1

                  sumatory=sumatory/X.shape[0]
                  return sumatory

          #LINEAR REGRESSION FOR CLASSIFICATION

          #CODIGO DEL ESTUDIANTE
          def PIA(X,Y):
                  #Pseudo-inverse algorithm
                  #calculo la transpuesta de X
                  x_t = X.transpose()
```

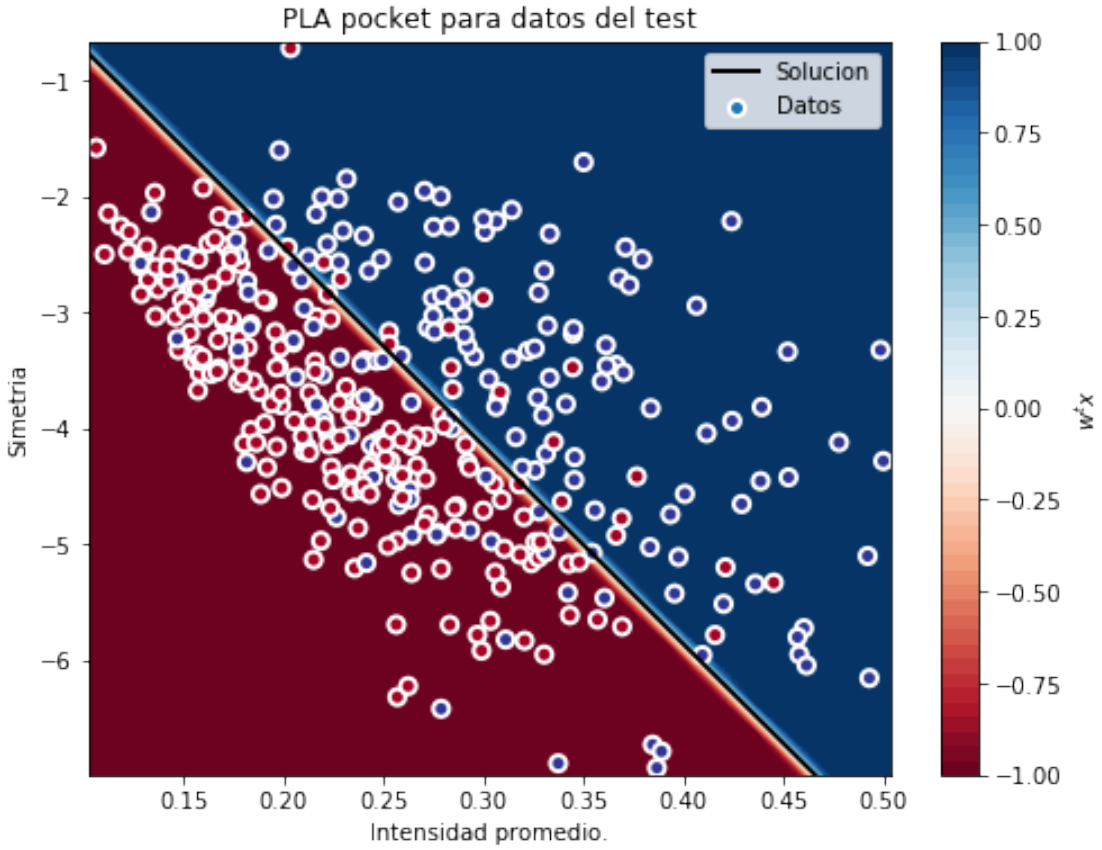
```
        pseudo_inverse=np.linalg.inv(x_t.dot(X))
        pseudo_inverse=pseudo_inverse.dot(x_t)
        pseudo_inverse=pseudo_inverse.dot(Y)

        return pseudo_inverse

#POCKET ALGORITHM

#CODIGO DEL ESTUDIANTE
def PLA_pocket(datos,label,max_iter,wini):
        total_iters=0
        cambios=True
        ncambios=0
        w=np.copy(wini)
        best_w=np.copy(w)
        best_ein=1
        for i in range (0,max_iter):
                cambios=False
                total_iters+=1

                for x,y in zip(datos,label):
                        if np.sign(np.dot(np.transpose(w),x))!=np.sign(y):
                                w=w+(y*x)
                                ncambios+=1
                if (ncambios>0):
                        ncambios=0
                        cambios=True
                err=Error_lineal(datos,label,w)
                if err<best_ein:
                        best_ein=err
                        best_w=np.copy(w)

                if not cambios:
                        break

        return best_w,total_iters

def plot_data(X, y, w,title='Point clod plot',x_l='x axis',y_l='y axis'):
    #Preparar datos

    a, b = coef2line(w)
    min_xy = X.min(axis=0)
    max_xy = X.max(axis=0)
    border_xy = (max_xy-min_xy)*0.01

    #Generar grid de predicciones
    xx, yy = np.mgrid[min_xy[0]-border_xy[0]:max_xy[0]+border_xy[0]+0.001:border_xy[
                      min_xy[1]-border_xy[1]:max_xy[1]+border_xy[1]+0.001:border_xy[
```

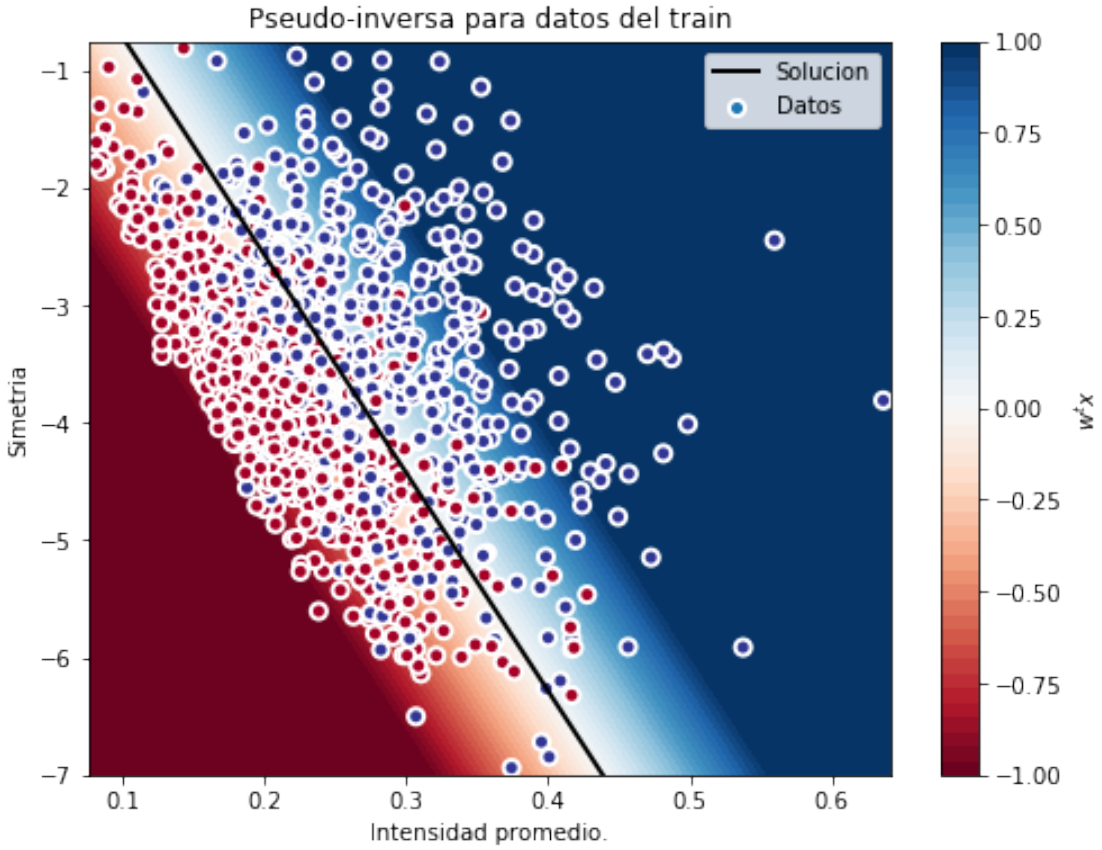
```
grid = np.c_[xx.ravel(), yy.ravel(), np.ones_like(xx).ravel()]
pred_y = grid.dot(w)
pred_y = np.clip(pred_y, -1, 1).reshape(xx.shape)

#Plot
f, ax = plt.subplots(figsize=(8, 6))
contour = ax.contourf(xx, yy, pred_y, 50, cmap='RdBu',
                      vmin=-1, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label('$w^tx$')
ax_c.set_ticks([-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1])
ax.scatter(X[:, 0], X[:, 1], c=y, s=50, linewidth=2,
            cmap="RdYlBu", edgecolor='white', label='Datos')
ax.plot(grid[:, 0], a*grid[:, 0]+b, 'black', linewidth=2.0, label='Solucion')
ax.set(
   xlim=(min_xy[0]-border_xy[0], max_xy[0]+border_xy[0]),
   ylim=(min_xy[1]-border_xy[1], max_xy[1]+border_xy[1]),
   xlabel=x_l, ylabel=y_l)
ax.legend()
plt.title(title)
plt.show()
```

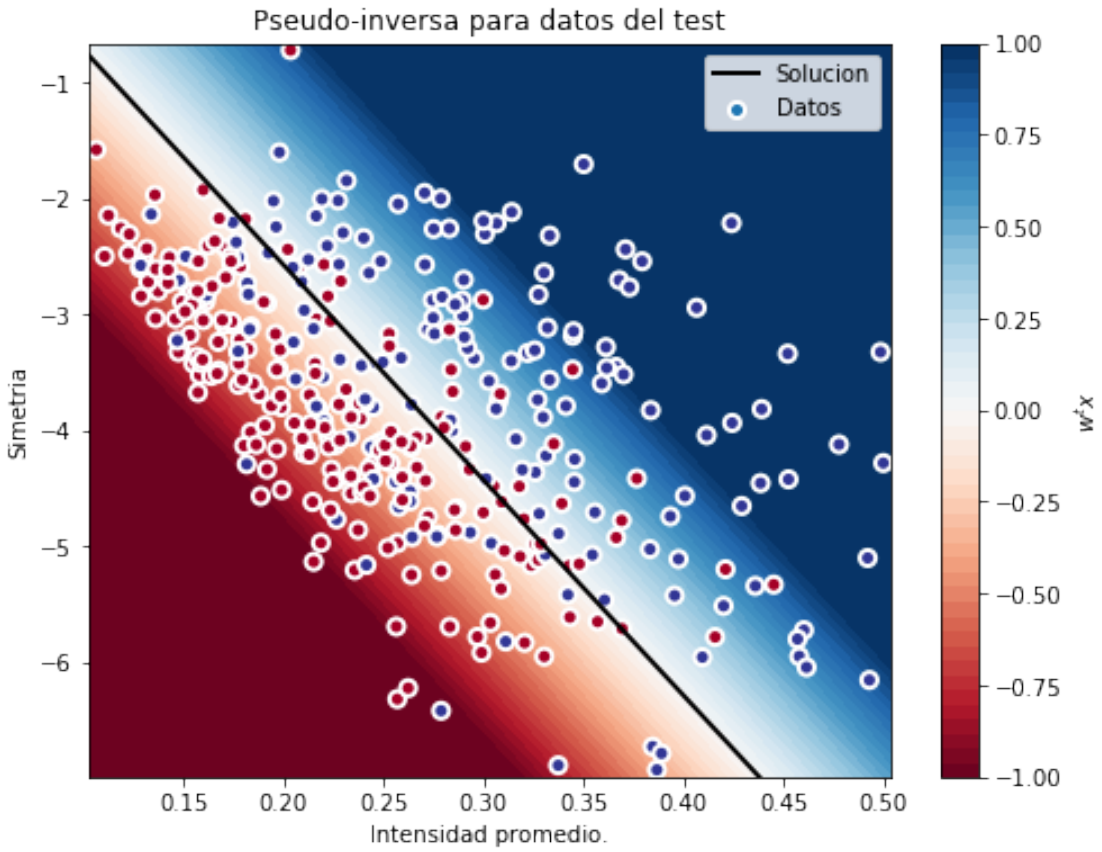
a) Generar grácos separados (en color) de los datos de entrenamiento y test junto con la función estimada.

```
In [707]: w_pia=PIA(x,y)
          w_pla,iters=PLA_pocket(x,y,1000,np.zeros(X_train.shape[1]))

In [708]: plot_data(x,y,w_pla,title="PLA pocket para datos del train",x_l="Intensidad promedio
          plot_data(x_test,y_test,w_pla,title="PLA pocket para datos del test",x_l="Intensidad
          plot_data(x,y,w_pia,title="Pseudo-inversa para datos del train",x_l="Intensidad prom
          plot_data(x_test,y_test,w_pia,title="Pseudo-inversa para datos del test",x_l="Intens
```


PLA pocket para datos del train
Solucion
Datos
Simetria
Intensidad promedio.
−1
−2
−3
−4
−5
−6
−7
0.1
0.2
0.3
0.4
0.5
0.6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$w^t x$


PLA pocket para datos del test
Solucion
Datos
Simetria
Intensidad promedio.
−1
−2
−3
−4
−5
−6
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$w^t x$


Pseudo-inversa para datos del train
Solucion
Datos
Simetria
Intensidad promedio.
−1
−2
−3
−4
−5
−6
−7
0.1
0.2
0.3
0.4
0.5
0.6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$w^t x$



b) Calcular Ein y Etest (error sobre los datos de test).

```
In [709]: print ("\n\n\nALGORITMO PLA-POCKET")
          print ("\n\n\nEin: ",Error_lineal(x,y,w_pla))
          print ("Etest: ",Error_lineal(x_test,y_test,w_pla))
          print ("\n\n\nALGORITMO PSEUDO-INVERSA")
          print ("\n\n\nEin: ",Error_lineal(x,y,w_pia))
          print ("Etest: ",Error_lineal(x_test,y_test,w_pia))
```

```
ALGORITMO PLA-POCKET


Ein:  0.228643216080402
Etest:  0.2459016393442623
```

```
ALGORITMO PSEUDO-INVERSA



Ein:  0.22780569514237856
Etest:  0.25136612021857924
```

Como podemos ver el resultado no es demasiado bueno. Debemos pensar que los datos no eran fácil de clasidifcar y menos con un modelo lineal. Por tanto, el metodo de la pseudo inversa nos ofrece una solucion que podemos considerar buena respecto a los datos que teníamos y realiza una division de forma general de forma rápida.

En el caso del PLA-Pocket el vecto que obteniamos del método de la pseudo inversa. Que lo que hace es almacenar por cada iteracion la mejor soluciónencontrada de modo que finalmente si para por el numero de iteraciones, puede devolver la mejor solucion y no la que en ese momento tenga como pasa con la pseudo inversa. Por lo que vemos hay una pequeña mejora en el resultado con respecto a la pseudo inversa, pero igalmente sigue sin ser muy buena, esto sedebe a que es realemtne dificil dividir correctamente con un modelo lineal.

c) Obtener cotas sobre el verdadero valor de Eout. Pueden calcularse dos cotas una basada en Ein y otra basada en Etest. Usar una tolerancia = 0,05. £Que cota es mejor?

```
In [710]: def VC(e,ndigitos,dim,tc):
              return e+np.sqrt((8/ndigitos)*np.log((4*((2*ndigitos)**dim +1))/tc))

          eoutTrain = VC(Error_lineal(x,y,w_pla),x[:, 0].size,3,0.05)
          eoutTest = VC(Error_lineal(x_test,y_test,w_pla),x_test[:, 0].size,3,0.05)

          print ("\n\n\nEin: ",eoutTrain)
          print ("Etest: ",eoutTest)
```

```
Ein:  0.6595797215511057
Etest:  0.972738760463036
```

Como podemos ver las cotas de error son muy altas. La diferencia que hay entre ambas es bastante notoria, esto se debe a que una tiene un conjunto de datos mayor y por lo tanto puede aproximar de forma más exacta el error. En el caso del test no tenemos muchos ejemplos, por ello la cota se nos hace casi inservible. En definitiva lo unico que podemos extraer, es que el factor influye para que fuera de la muestra tendremos un buen resultado, esel tamaño dela muestra, ya que la dimension VC será la misma.