Ejercicio sobre la complejidad de H y el ruido.

En este ejercicio debemos aprender la dificultad que introduce la aparición de ruido en las etiquetas a la hora de elegir la clase de funciones más adecuada. Haremos uso de tres funciones:

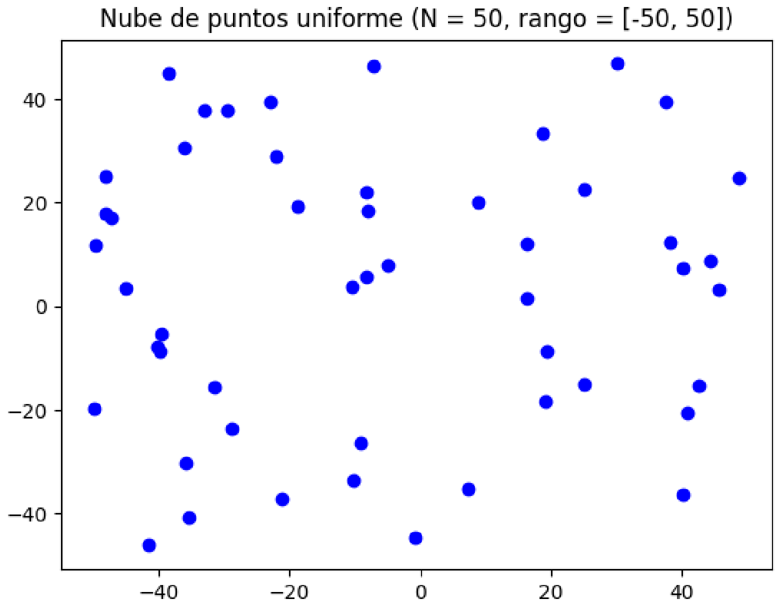
- `simula_unif (N, dim, rango)`, que calcula una lista de N vectores de dimensión dim. Cada vector contiene dim números aleatorios uniformes en el intervalo rango.
- `simula_gaus(N, dim, sigma)`, que calcula una lista de longitud N de vectores de dimen- sión dim, donde cada posición del vector contiene un número aleatorio extraído de una distribucción Gaussiana de media 0 y varianza dada, para cada dimension, por la posición del vector sigma.
- `simula_recta(intervalo)` , que simula de forma aleatoria los parámetros, v = (a, b) de una recta, $y = ax + b$, que corta al cuadrado $[-50, 50] \times [-50, 50]$.

1 Dibujo de las gráficas

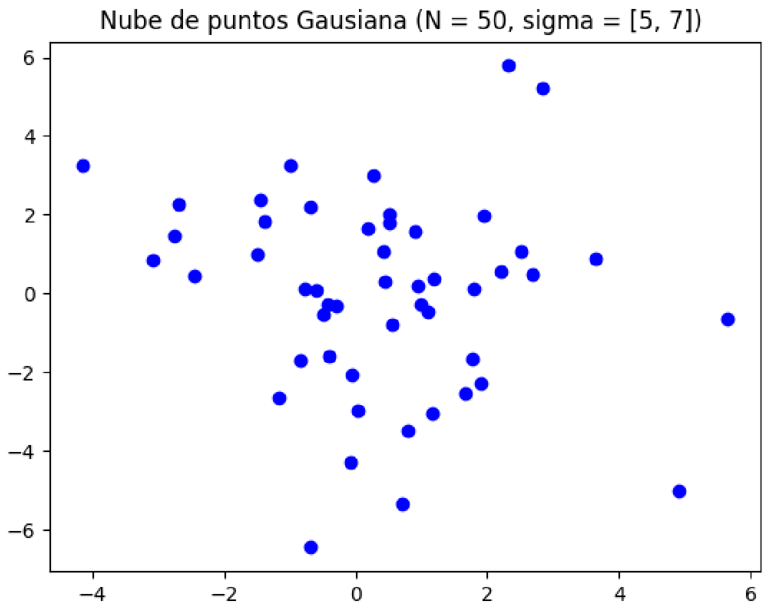
Dibujar gráficas con las nubes de puntos simuladas con las siguientes condiciones:

Para ellos hemos utilizado la función `scatter plot` (introducir código de la función TO-DO)

a) Considere $N = 50, \quad dim = 2, rango = [-50, +50]$ con `simula_unif(N,dim, rango)`.



b) Considere $N = 50, \quad dim = 2, sigma = [5, 7]$ con `simula_gaus(N,dim, sigma)`.



Valoración de la influencia del ruido en la selección de la complejidad de la clase de funciones.

Con ayuda de la función `simula_unif(100, 2, [-50, 50])` generamos una muesta de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función $f(x, y) = y - ax - b$, es decir el signo de la distancia de cada punto a la recta simulada con `simula_recta()`.

Función de muestra de gráficas

Todas estas funciones fueron explicadas en la práctica uno, se utilizan la funciones `scattered` y `contour` de la librería `matplotlib.pyplot`.

Como único comentario, el límite de división de la función de clasificación $f(x, y)$ es muy fácil de dibujar.

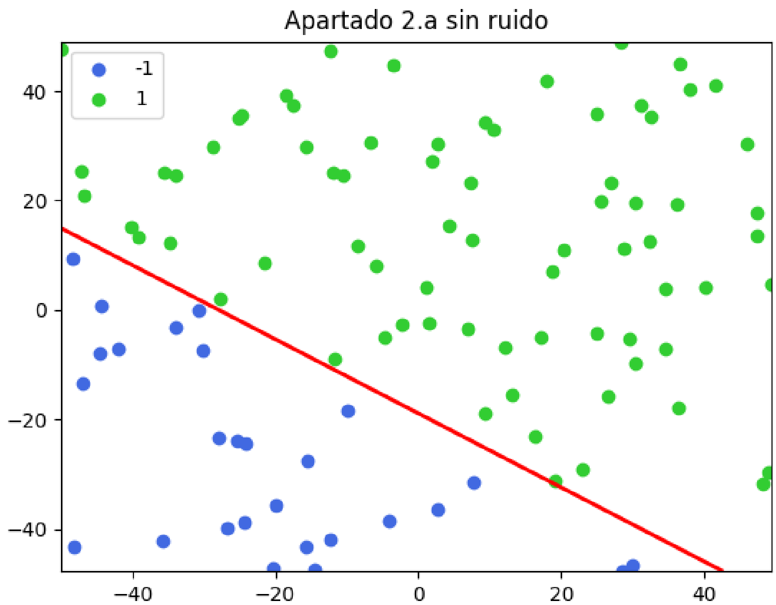
Sabemos que es $f(x, y) = 0$, una recta; luego solo habría que calcular dos puntos de ésta (por ejemplo hacer $x_1 = 0ey_2 = 0$ y resolver respectivas ecuación) y pintar la recta que pasa por esos dos puntos.

Sin embargo se ha optado por hacer uso de la función `contout` para tener mayor generalidad, ya que esta es capaz de pintar los puntos de la ecuación $g(x, y) = 0$ de $g : \mathbb{R}^2 \longrightarrow \mathbb{R}$.

```python
def classified_scatter_plot(x,y, function, plot_title, labels, colors):
        '''Dibuja los datos x con sus respectivas etiquetas y
        Dibuja la función: function
        labels: son las etiquetas posibles que queremos que distinga para colorear,
        (todo esto en el mismo gráfico
        '''
        plt.clf()


        for l in labels:
                index = [i for i,v in enumerate(y) if v == l]
                plt.scatter(x[index, 0], x[index, 1], c = colors[l], label = str(l))



        ## ejes
        xmin, xmax = np.min(x[:, 0]), np.max(x[:, 0])
        ymin, ymax = np.min(x[:, 1]), np.max(x[:, 1])


        ## function plot
        spacex = np.linspace(xmin,xmax,100)
        spacey = np.linspace(ymin,ymax,100)
        z = [[ function(i,j) for i in spacex] for j in spacey ]
        plt.contour(spacex,spacey, z, 0, colors=['red'],linewidths=2 )

        # título
        plt.title(plot_title)

        plt.show()
```

a) Gráfico 2D

El resultado de dibujar esto es



Podemos observar que como era de esperar el dibujo y la clasificación están ien hechos.

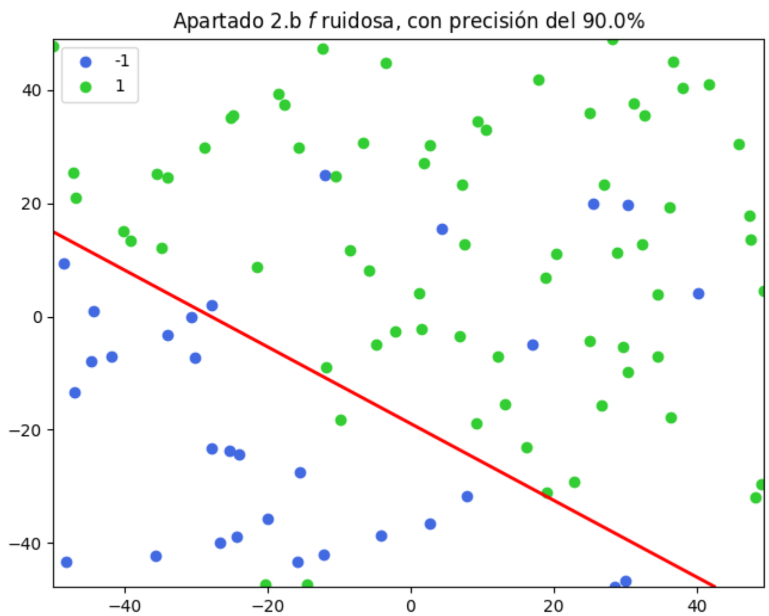
b) Ruido

Introducimos ruído en las etiquetas utilizando la función:

```
def noisyVector(y, percent_noisy_data, labels):
        '''
        y vector sobre el que introducir ruido
        labels: etiquetas sobre las que vamso a introducir ruido
        percent_noisy_data: porcentaje de ruido de cada etiqueta

        '''

        noisy_y = np.copy(y)

        for l in labels:
                index = [i for i,v in enumerate(y) if v == l]
                np.random.shuffle(index)
                len_y = len(index)
                size_noisy_data = int((len_y*percent_noisy_data)/ 100 )
```

```
            for i in index[:size_noisy_data]:
                    noisy_y[i] *= -1
        return noisy_y
```

El resultado tras meter ruido es:



Podríamos sorprendernos de que solo tres datos de los clasificados como negativos sean ahora positivos, esto se debe a que son menos que los positivos. Análicemos en total el porcentaje cambiado con la función `analis_clasificado` para cerciorarnos de que es correcto

Su salida en ejecución es :

```
Apartado 1.2.b
Resultado clasificación:
     Positivos fallados 7.0  de 73, lo que hace un porcentaje de 9.58904109589041
     Negativos fallados 3.0  de 27, lo que hace un porcentaje de 11.11111111111111
     Total fallados 10.0  de 100, lo que hace un porcentaje de 10.0
     La precisión es   de 90.0 %
```

Se nos pedía clasificar mal el 10% de los positivos, que son 73, luego eso supondría modificar 7.3 datos mal, puesto que se redondea, la clasificación actual es de 9.58%. Para el caso de los negativos se procede igual.

Sin embargo el resultado final sí que es 10%, lo cual nos termina por confirmar la corrección del algoritmo ya que el error de clasificación sería $malClasificados =$

$0.1 positivos + 0.1 negativos = 0.1(positivos + negativos)$ y aunque se ha redondea, como uno ha sido a la alta y el otro a la baja esto hace que se compense el total y el porcentaje final de fallados sea el pedido para subcategoría.

c) Nuevas funciones frontera.

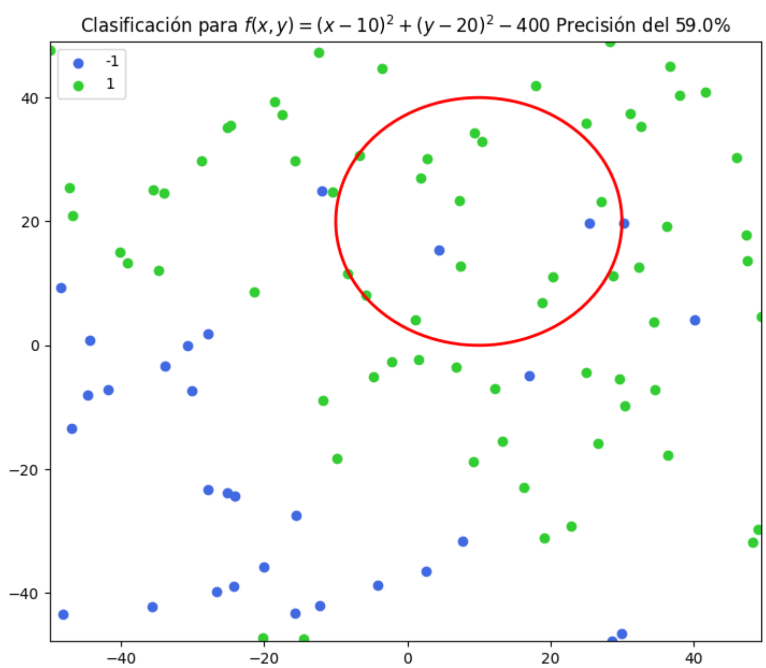
Analizaremos ahora los resultados modificando las funciones frontera:

Para analizar la bondad del ajuste vamos a tener en cuenta la precisión, además para comprobar si beneficia más a un tipo u a otro analizando los positivos y negativos fallados.

Recordamos que la precisioń se define como $\frac{\text{datos bien clasificados}}{\text{datos bien clasificados}}$, nosotro además la indicamos como porcentaje, multiplicada por 100.

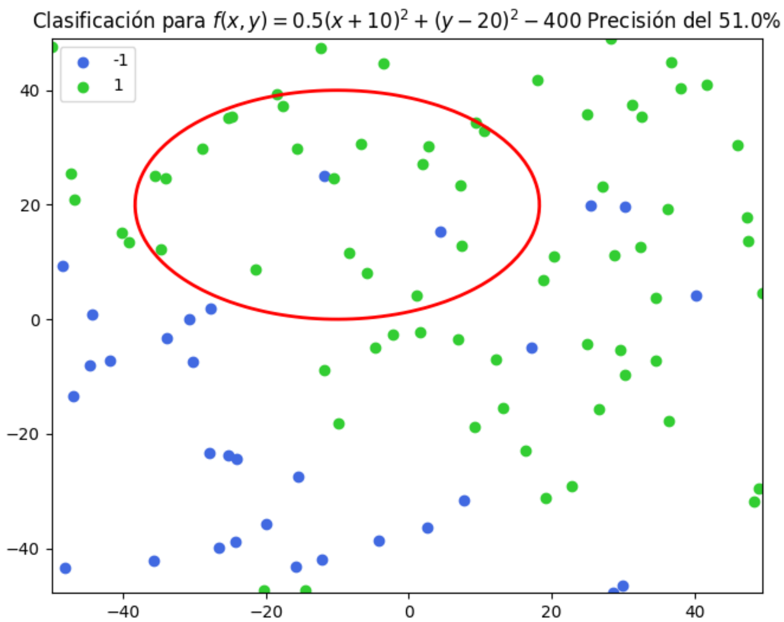
$f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$

- Precisión obtenida: 59%
- Porcentaje positivos fallados: 17.39%
- Porcentaje negativos fallados: 93.548%



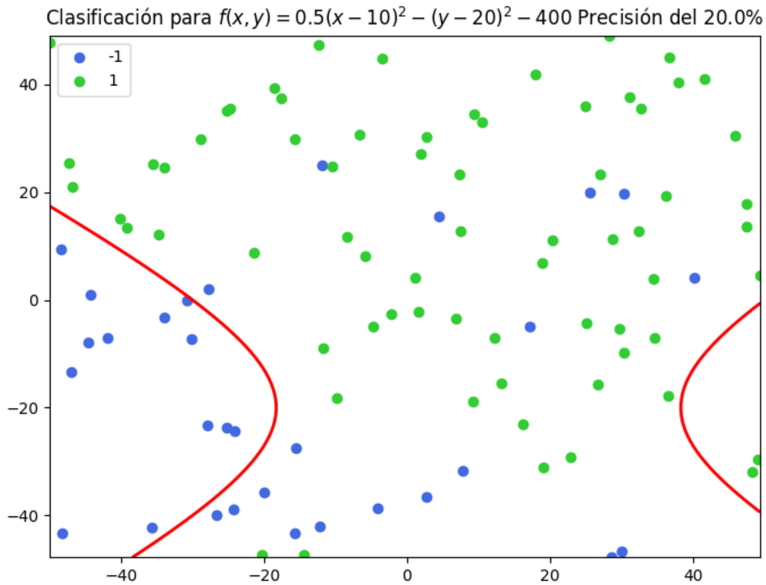
$f(x,y) = 0.5(x+10)^2 + (y-20)^2 - 400$

- Precisión obtenida: 51.0%
- Porcentaje positivos fallados: 28.986%
- Porcentaje negativos fallados: 93.54%



$f(x,y) = 0.5(x-10)^2 - (y-20)^2 - 400$

- Precisión obtenida: 20%
- Porcentaje positivos fallados: 97.101%
- Porcentaje negativos fallados: 41.935%



$f(x, y) = y - 20x^2 - 5x + 3$

- Precisión obtenida: 31%
- Porcentaje positivos fallados: 100%
- Porcentaje negativos fallados: 0%

Clasificación para $f(x, y) = y - 20x^2 - 5x + 3$ Precisión del 31.0%