Práctica 2

Aprendizaje Automático

Alejandro Borrego Megías

Curso 2020-2021

Índice

1. Complejidad de $\mathcal{H}$ y ruido **2**

1.1 Dibujar gráficas con las nubes de puntos simuladas con las siguientes condiciones 2

1.2 Vamos a valorar 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 muestra de puntos 2D a los que vamos a 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() 4

1. Complejidad de $\mathcal{H}$ y ruido

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

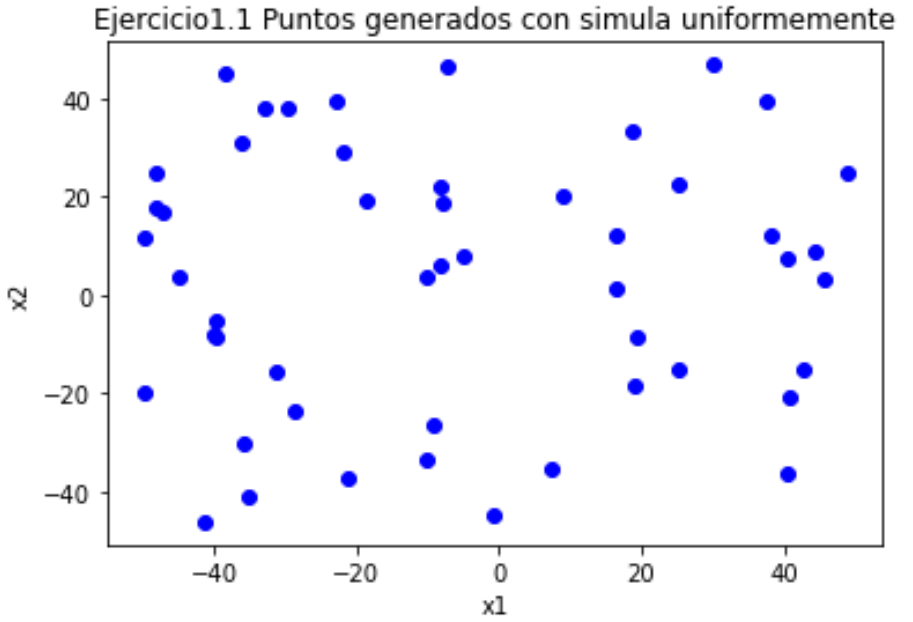
- **Considere N=50, dim=2, rango=[-50,50] con simula_unif(N,dim,rango).**
- **Considere N=50, dim=2, sigma=[5,7] con simula_gaus(N,dim,sigma).**

En este primer ejercicio vamos a probar las funciones que se nos proporcionan en el template para generar nubes de puntos:

En primer lugar, genero con ayuda de la función simula_unif(50,2,[-50,50]) una nube de 50 puntos en 2 dimensiones que toman valores en el intervalo [-50,50] en cada componente y qu que siguen una distribución uniforme de probabilidad, el código sería el siguiente:

```
x = simula_unif(50, 2, [-50,50])
# Dibujo el Scatter Plot de los puntos generados
plt.scatter(x[:,0],x[:,1], c='blue')
plt.title('Ejercicio1.1 Puntos generados con simula uniformemente')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

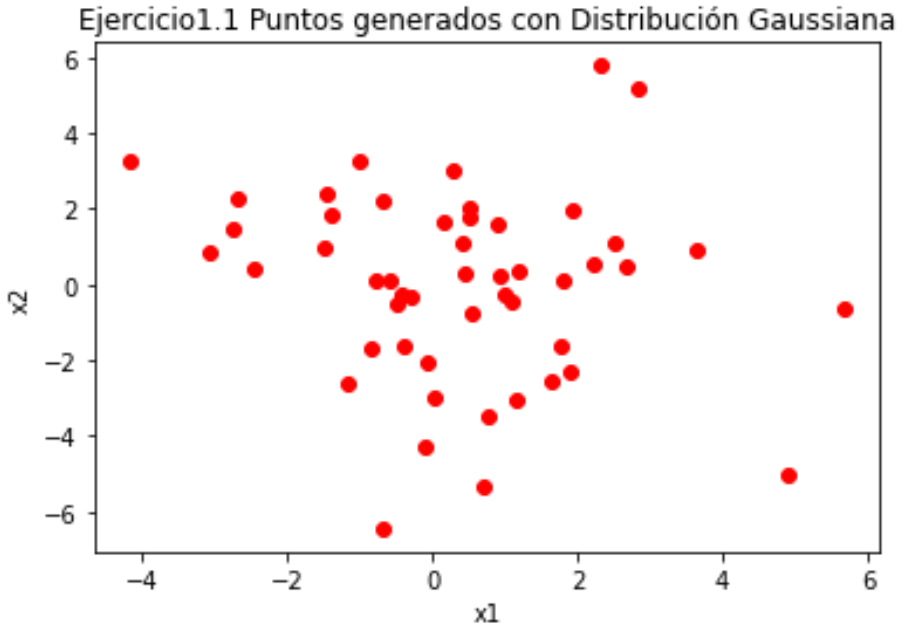
La matriz x contiene los valores de la primera y segunda coordenada de cada punto y con la función scatter de matplotlib.pyplot representamos la nube de puntos, que quedaría así:



En segundo lugar, genero con ayuda de ka función simula_gaus(50,2,[5,7]) una nube de 50 puntos en 2 dimensiones que siguen una distribución Normal de parámetros N(0,5) en el eje x y N(0,7) en el eje y, el código sería el siguiente:

```
x = simula_gaus(50, 2, np.array([5,7]))
# Dibujo el Scatter Plot de los puntos generados
plt.scatter(x[:,0],x[:,1], c='red')
plt.title('Ejercicio1.1 Puntos generados con Distribución Gaussiana')
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()
```

Y el gráfico generado es:



1.2 Vamos a valorar 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 muestra de puntos 2D a los que vamos a 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()

- **Dibujar un grafico 2D donde los puntos muestren (usen colores) el resultado de su etiqueta. Dibuje también la recta usada para etiquetar. (observe que todos los puntos están bien clasificados respecto de la recta)**

En este apartado vamos a generar de nuevo una nube de puntos en 2 dimensiones que siguen una distribución Normal, pero en este caso, usaremos un hiperplano (una recta pues estamos en 2D) para dar una etiqueta a cada punto del espacio, si un punto se sitúa por encima de la recta, al evaluar dicho punto en $f(x, y)$ obtendrá un valor positivo, y la función signo le pondrá como etiqueta $+1$, en cambio si el punto se sitúa por debajo de la recta, en este caso se le asigna el valor -1.

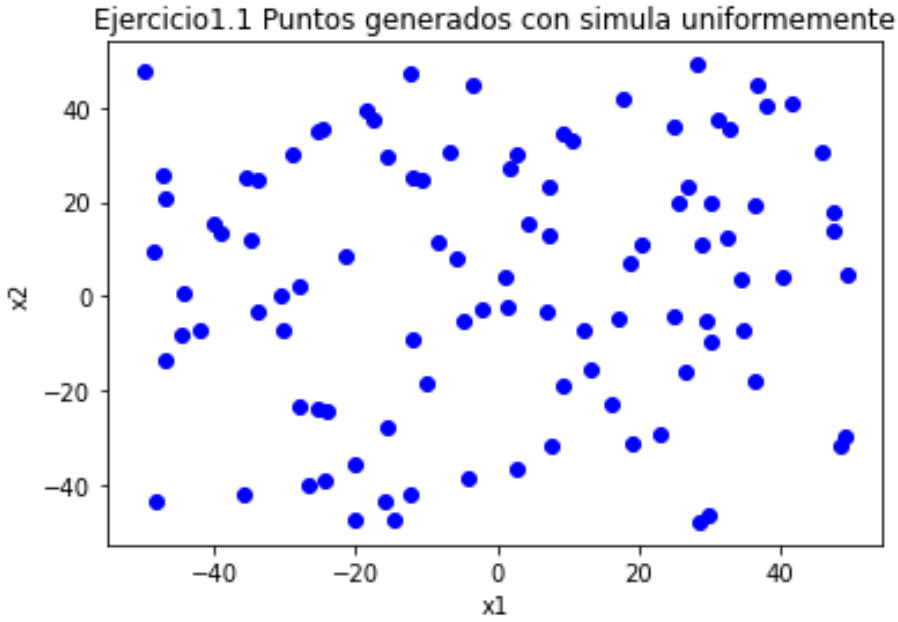
Para obtener los coeficientes a, b de la recta $f(x, y) = y - ax - b$ usaremos la función simula_recta([-50,50]) que se nos proporciona en el template, y pasamos

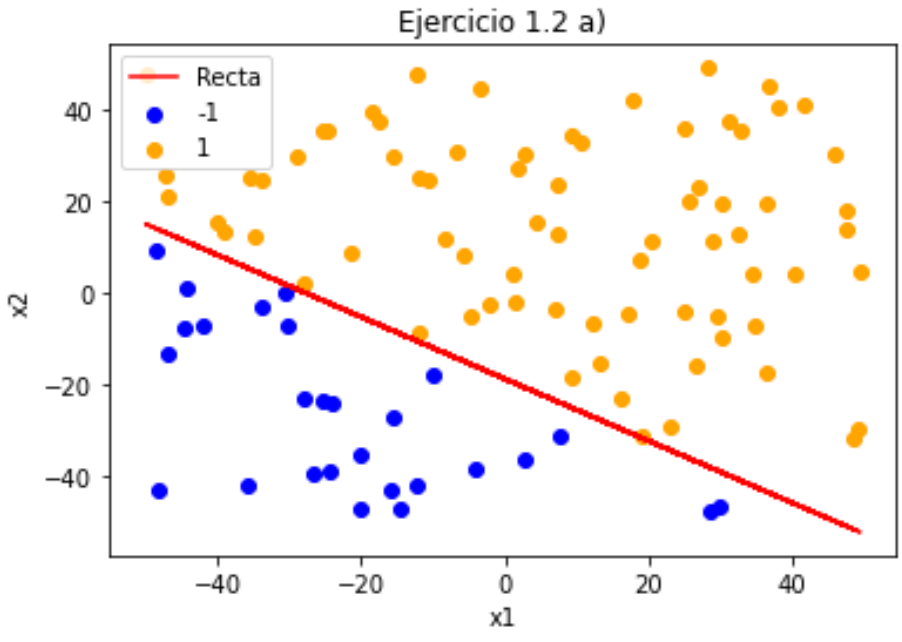
como parámetro el intervalo de defincición, en este caso [-50,50]. Finalmente, representamos en un gráfico la recta junto con los puntos clasificados. El código sería el siguiente:

```
#Calculamos los coeficientes de la recta
a,b=simula_recta([-50,50])

#Generamos las etiquetas
y=[]

for i in x :
    y.append(f(i[0],i[1],a,b))

y=np.array(y)
y0 = np.where(y == -1) #capturo los índices de los elementos con -1
y1 = np.where(y == 1) #capturo los índices de los elementos con 1
#x_2 contiene dos arrays, uno en cada componente, el primero tiene los valores de x con eti
x_2 = np.array([x[y0[0]],x[y1[0]]])
plt.scatter(x_2[0][:, 0], x_2[0][:, 1],  c = 'blue', label = '-1') #Dibujamos los puntos co
plt.scatter(x_2[1][:, 0], x_2[1][:, 1],  c = 'orange', label = '1')#Dibujamos los de etique

#Calculamos las imagenes de los puntos (sin aplicar la función signo) y así dibujar la recta
imagenes=[]

for i in x :
    imagenes.append(a*i[0]+b)

plt.plot( x[:,0], imagenes, c = 'red', label='Recta') #Para representarlo, despejo x2 de la
plt.legend()
plt.title("Ejercicio 1.2 a)")
plt.xlabel('x1')
plt.ylabel('x2')
plt.figure()
plt.show()
```

En primer lugar, como hemos dicho obtenemos los coeficientes de la recta, y después obtenemos las etiquetas para cada punto de la matriz x (matriz de los 100 puntos obtenidos por simula_unif()), en segundo lugar pintamos por separado los puntos clasificados con +1 y -1 usando el color naranja para los primeros y azul para los segundos. Finalmente dibujo la gráfica de la recta y represento todos los datos juntos en un gráfico. Finalmente comentar que la forma en que represento los datos es la misma que en la práctica pasada y por tanto ya se explicó en la práctica anterior cómo se realizaba.

A continuación muestro los gráficos correspondientes a este apartado:


Ejercicio1.1 Puntos generados con simula uniformemente
x2
x1
40
20
0
−20
−40
−40
−20
0
20
40


Ejercicio 1.2 a)
Recta
-1
1
x2
x1
40
20
0
−20
−40
−40
−20
0
20
40

Como comentario final, por la forma en que hemos procedido, el error de clasificación cometido por esta recta será de 0, ya que hemos usado esta recta para clasificar los datos.

- **Modifique de forma aleatoria un 10 % de las etiquetas positivas y otro 10 % de las etiquetas negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior. (Ahora habrá puntos mal clasificados respecto de la recta)**

En este apartado convertimos el vector y_0 de índices que nos indicaba las posiciones de las etiquetas con valor -1 en el vector y (vector de etiquetas) en un DataFrame de Pandas, ya que esta librería nos permite con facilidad obtener el 10 % de los índices y cambiar el valor de su etiqueta. Después hacemos lo mismo con el vector y_1.

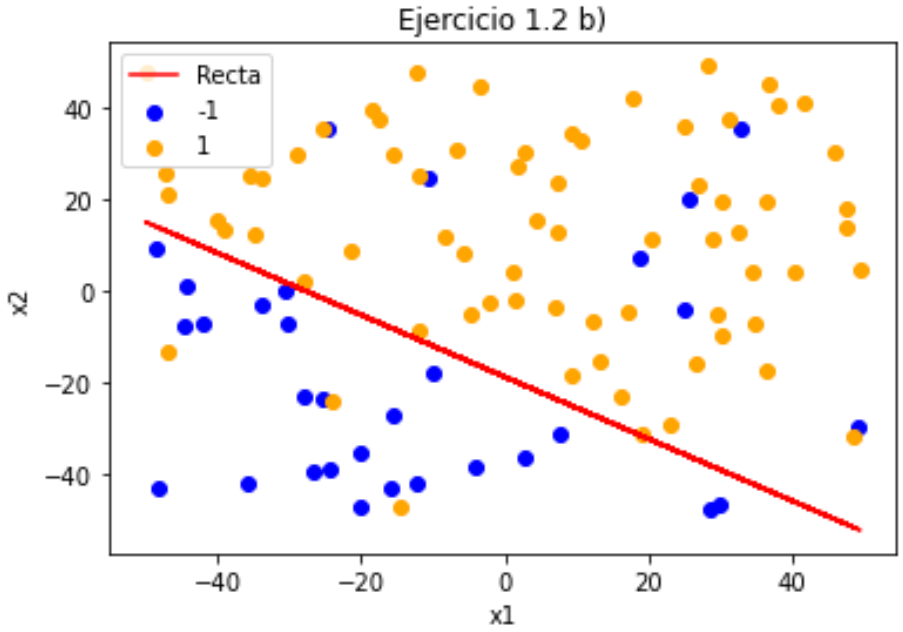
A continuación muestro el código:

```
### Como en el apartado siguiente vamos a calcular la accuracy del clasificador (TP+TN)/(P+
TN=len(y0[0]) #Numero de etiquetas con -1, pues antes de meter ruido la recta clasifica per
TP=len(y1[0]) #Numero de etiquetas con +1, pues antes de meter ruido la recta clasifica per

# Array con 10% de indices aleatorios para introducir ruido
y0=pd.DataFrame(data=y0[0]); #Convierto la matriz X en un Dataframe de Pandas, que es más c
y0=y0.sample(frac=0.10,random_state=1); #Hacemos que tome un 10% de los datos de forma alea
y0=y0.to_numpy()
for i in y0:
    y[i]=1

TN=TN-len(y0); #Como hemos etiquetado "mal" el 10% de los elementos pues actualizamos los T
```

Como comentarios al código, en primer lugar las variables TN y TP hacen referencia a los “True negatives” y “True positives”, es decir, aquellos puntos cuya etiqueta real coincide con la etiqueta de la función que estamos usando como clasificador. En este caso, antes de modificar las etiquetas, por lo comentado en el apartado anterior, la recta clasifica perfectamente los datos, es por eso que los TP coinciden con los “Positives” de la muestra y los TN coinciden con los “Negatives” de la muestra, y al actualizar el 10 % de las etiquetas , estos puntos pasarán automáticamente a estar mal clasificados por la recta y se debe restar el número de puntos elegido al valor de TP y TN que teníamos al principio. Esto se utilizará en el apartado siguiente, dónde compararemos la precisión (Accuracy) de distintas funciones al utilizarlas como clasificadores.

Finalmente comentar que el código es análogo para las etiquetas con +1.

Tras esta modificación, el gráfico queda de la siguiente forma:



- **Supongamos ahora que las siguientes funciones definen la frontera de clasificación de los puntos de la muestra en lugar de la recta. Visualizar el etiquetado generado en 2b) junto con cada una de las gráficas de cada una de las funciones. Compara las regiones positivas y negativas de estas nuevas funciones con las obtenidas en el caso de la recta. Argumente si estas funciones más complejas son mejores clasificadores que la función lineal. Observe las gráficas y diga que consecuencias extrae sobre la influencia del proceso de modificación de etiquetas en el proceso de aprendizaje. Explicar el razonamiento.**
  - $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
  - $f(x, y) = 0{,}5(x - 10)^2 + (y - 20)^2 - 400$
  - $f(x, y) = 0{,}5(x - 10)^2 - (y + 20)^2 - 400$
  - $f(x, y) = y - 20x^2 - 5x + 3$

Este apartado es el más interesante, pues vamos a comparar el clasificador obtenido en los apartados anteriores (recordemos, la recta $f(x, y) = y - ax - b$) con otras funciones que definen distintas fronteras de decisión.

En primer lugar aclarar como ya se comentó en el apartado anterior que para comparar las distintas funciones usaremos la precisión o accuracy ($\frac{TN+TP}{P+N}$) que nos proporciona el porcentaje de puntos bien clasificados por cada función. De hecho comenzamos calculando la precisión de la recta del apartado anterior, que tiene una precisión de 0.9. Tiene lógica que salga tan buena precisión pues es

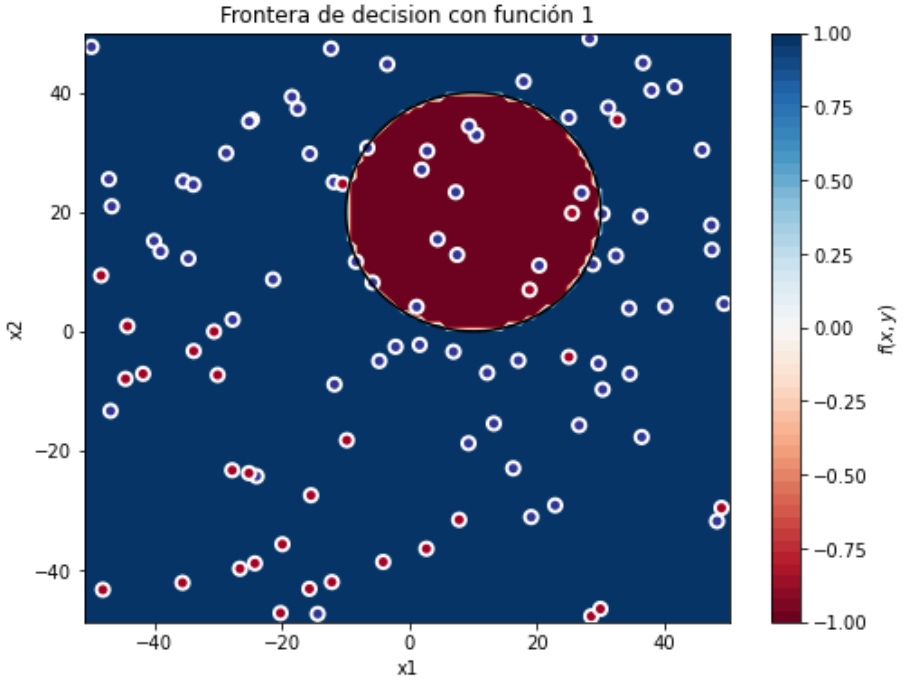
la función que hemos usado para clasificar la nube de puntos y después hemos metido un 10 % de ruido a sus etiquetas positivas y negativas, por lo que no debía bajar mucho la precisión del 100 %.

Dicho esto, realizaré una explicación del código usado para la primera función pues para el resto es completamente análogo:

```
def f1(x):
    y=[]
    for i in x:
        y.append((i[0]-10)**2 + (i[1]-20)**2-400)

    return np.asarray(y)

plot_datos_cuad(x,y,f1,'Frontera de decision con función 1', 'x1', 'x2')

imagenes=f1(x) #Capturo las imágenes de cada punto
TN=0
TP=0
cont=0

for i in imagenes:

    if i>0 and y[cont]>0: #Si tienen la misma etiqueta +1
        TP+=1
    if i<0 and y[cont]<0: #Si tienen la misma etiqueta -1
        TN+=1
    cont+=1

print("Mostramos la accuracy del método f1")
print ("ACCURACY= ", (TN+TP)/100.0)
```

En primer lugar definimos la función que vamos a usar para definir la frontera de decisión. Como comentario, para poder hacer uso posteriormente de la función plot_datos_cuad() que se nos proporciona en el template debemos definir la función de manera que dada la matriz de datos x, nos devuelva directamente el vector y de imágenes.

En segundo lugar, haciendo uso de la función anteriormente mencionada representamos las regiones de puntos con etiqueta +1 y -1 y los puntos del apartado anterior (así veremos visualmente si nuestro clasificador es bueno o no). Dicho gráfico es el siguiente:



La región azul representa la región de puntos con etiqueta +1 según nuestro nuevo clasificador y la región burdeos la región de puntos con etiqueta -1.

Finalmente, en la útlima parte del código calculamos los TP y TN de nuestra nueva función. Para ello, usando el vector de imágenes generado por la nueva función, vemos en que puntos coincide el signo del punto con el de la etiqueta, y cuando hacemos fracción nos da el siguiente resultado:

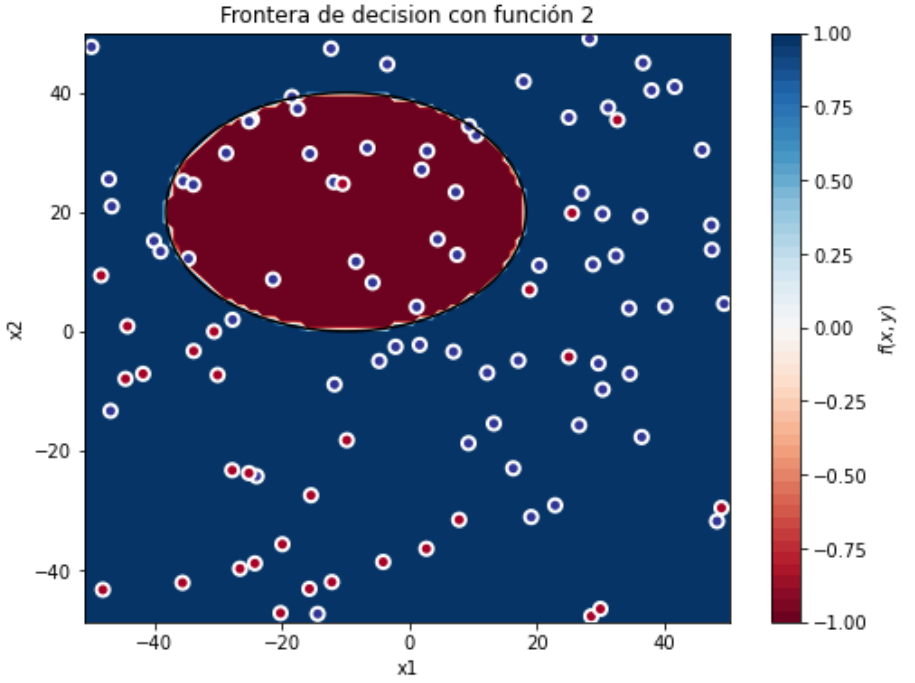
```
Mostramos la precisión del método del apartado anterior
ACCURACY=  0.9
Mostramos la accuracy del método f1
ACCURACY=  0.59
```

Como podemos observar, la nueva función no clasifica tan bien los datos como la del apartado anterior, y hacierta tan solo en un 59 % de las etiquetas, luego esto no lleva a pensar que si dicha función no representa una frontera de decisión que explique correctamente la muestra. A pesar de esto se puede pensar que la precisión tampoco es tan baja, dado que el área donde la función asigna etiqueta +1 es tan grande que engloba muchos puntos (que clasifica tanto bien como mal), y además en la región -1 ha dado la casualidad de que caen algunos de los puntos alterados, lo cual hace que los clasifique como correctos también. Pero claro, estas conclusiones las hemos sacado visualizando el gráfico, pues solo con

los datos podríamos haber pensado que quizá no era tan mal clasificador.

En las siguientes funciones procedemos de forma análoga.

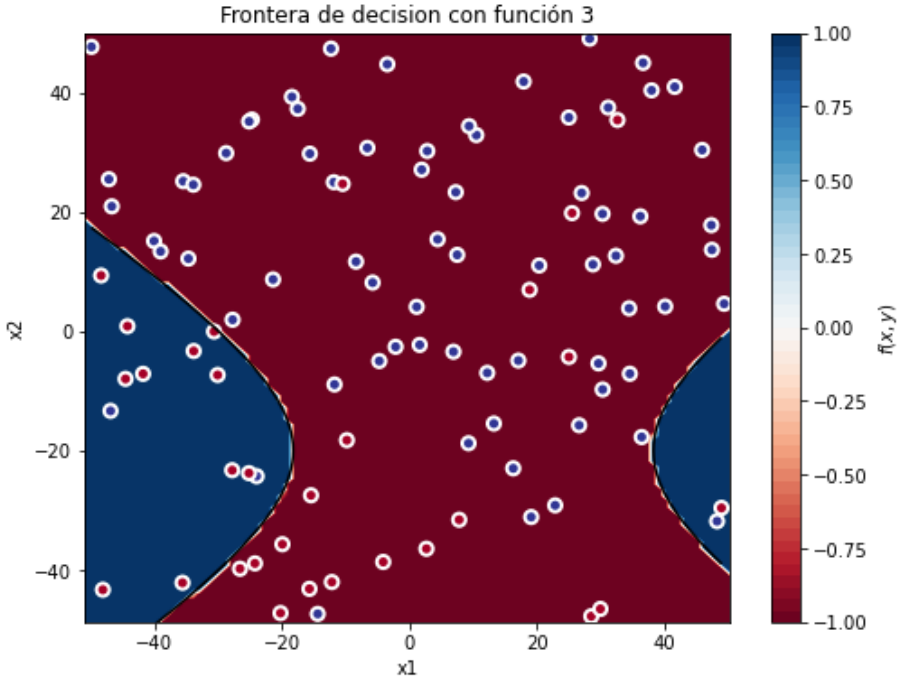
Para la función $f(x,y) = 0{,}5(x-10)^2 + (y-20)^2 - 400$ obtenemos los siguientes resultados:




```
Mostramos la precisión del método del apartado anterior
ACCURACY=  0.9
Mostramos la accuracy del método f2
ACCURACY=  0.51
```

Como podemos observar, en este caso pasa algo similar al anterior, la precisión comparada con la recta empeora bastante, y el porcentaje de aciertos que tiene se debe a lo mismo que pasaba en el apartado anterior, luego la conclusión es que tampoco es una buena función para definir la frontera de decisión.

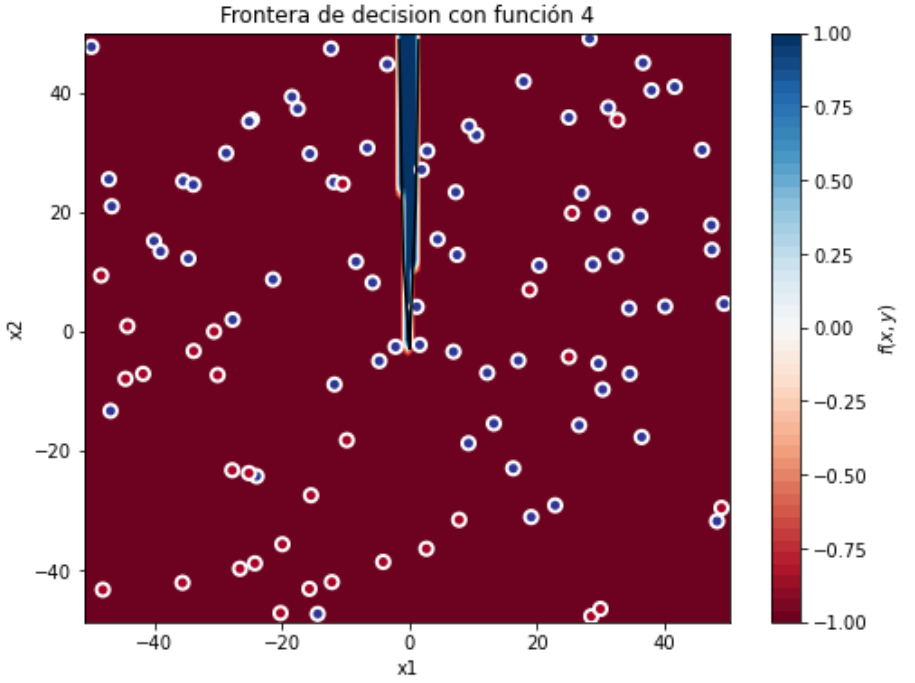
Para $f(x,y) = 0{,}5(x-10)^2 - (y+20)^2 - 400$ obtenemos los siguientes resultados:




```
Mostramos la precisión del método del apartado anterior
ACCURACY=  0.9
Mostramos la accuracy del método f3
ACCURACY=  0.22
```

Claramente esta función no explica la muestra, el porcentaje de acierto es de un 22 % frnte al 90 % de la recta y por la gráfica se da a entender que los puntos correctamente clasificados han sido por suerte. Luego como conclusión no es una función correcta para gneralizar y para definir la frontera de decisión.

Finalmente para $f(x, y) = y - 20x^2 - 5x + 3$ obtenemos los siguientes resultados:




```
Mostramos la precisión del método del apartado anterior
ACCURACY=  0.9
Mostramos la accuracy del método f4
ACCURACY=  0.31
```

De nuevo nos encontramos ante un mal clasificador para nuestra frontera de decisión, las regiones de +1 y -1 no se ajustan a los datos y las coincidencias se deben a que la región -1 es muy amplia y engloba a todos los datos con etiqueta -1, pero como también engloba a los de etiqueta +1 pues se comete un error muy alto.

Finalmente, la conclusión que podemos sacar sobre la influencia del ruido en la muestra y los posibles clasificadores que valoremos es que pueden influir positiva o negativamente en los clasificadores provocando que algunos clasificadores que a priori no explican en absoluto la muestra cometan un porcentaje de acierto mayor de lo esperado o que buenos clasificadores como la recta del apartado anterior cometan un error mayor del esperado. Algo que debemos tener en cuenta, por lo que si podemos representar los puntos y gráficas en un gráfico es recomendable hacerlo, para sacar conclusiones más allá de los números y porcentajes obtenidos.