Natalia Fernández Martínez, grupo A2
77449273Q

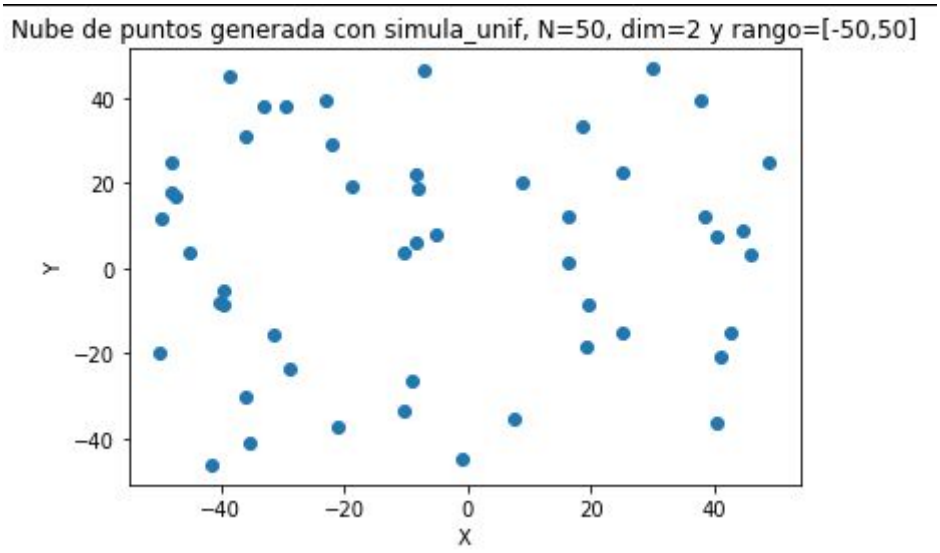
TRABAJO 2. Programación

1. Ejercicio sobre la complejidad de H y el ruido

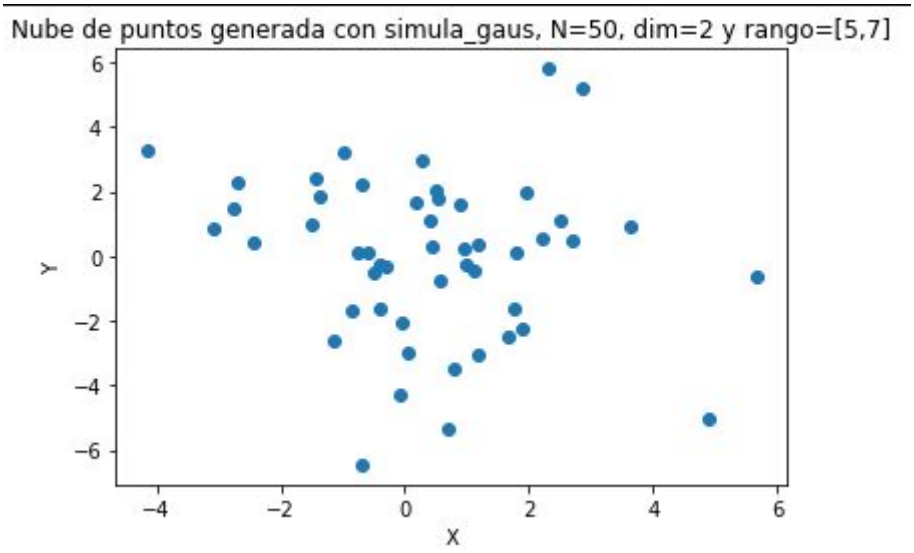
Apartado 1

Dibujar una gráfica con la nube de puntos de salida correspondiente.

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



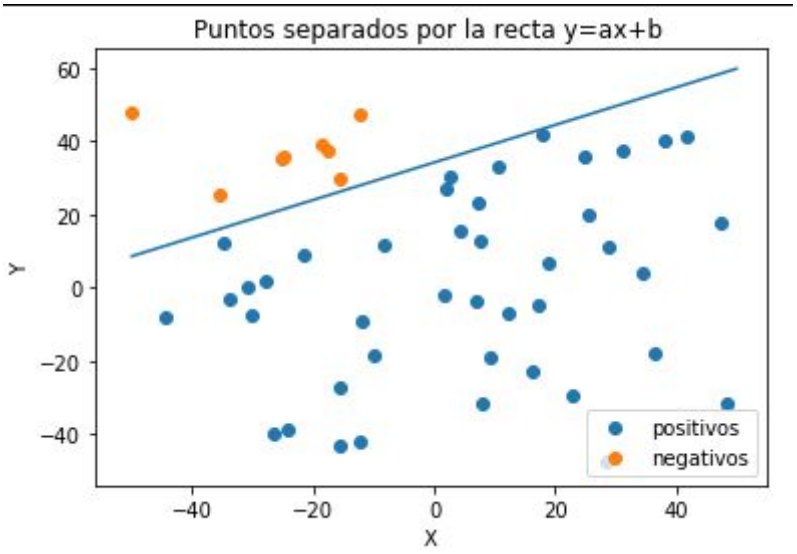
b) **Considere N= 50, dim= 2 y sigma= [5,7] con simula_gaus(N,dim,sigma)**



Apartado 2

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) =y−ax−b, es decir el signo de la distancia de cada punto a la recta simulada con simula_recta().

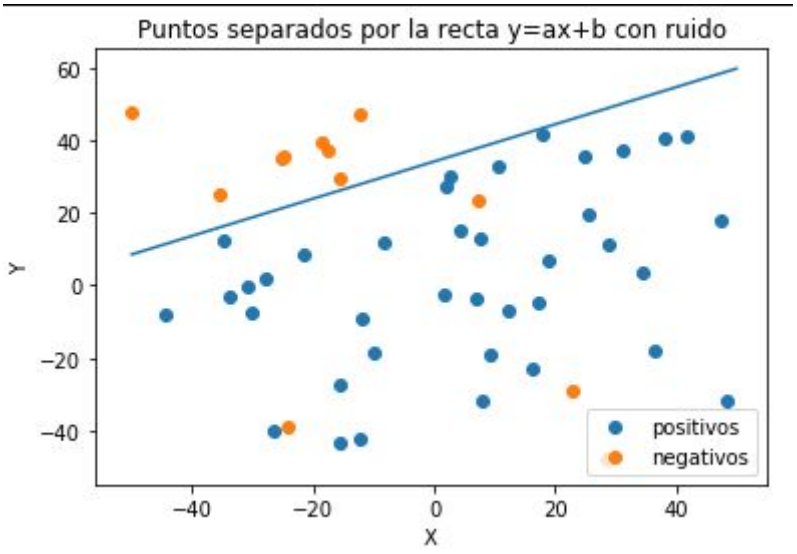
a) **Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello. (Observe que todos los puntos están bien clasificados respecto de la recta)**



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

Para introducir el ruido, he separado los puntos en positivos y negativos, y he tomado el 10% de cada vector. Si ese 10% es al menos 1, se transforma en un int, y ese es el número de puntos al que se le va a cambiar la etiqueta.

Para cambiar las etiquetas, se guarda ese 10% de cada vector en vectores auxiliares y se eliminan de sus vectores originales, y después se une cada uno con el vector opuesto.

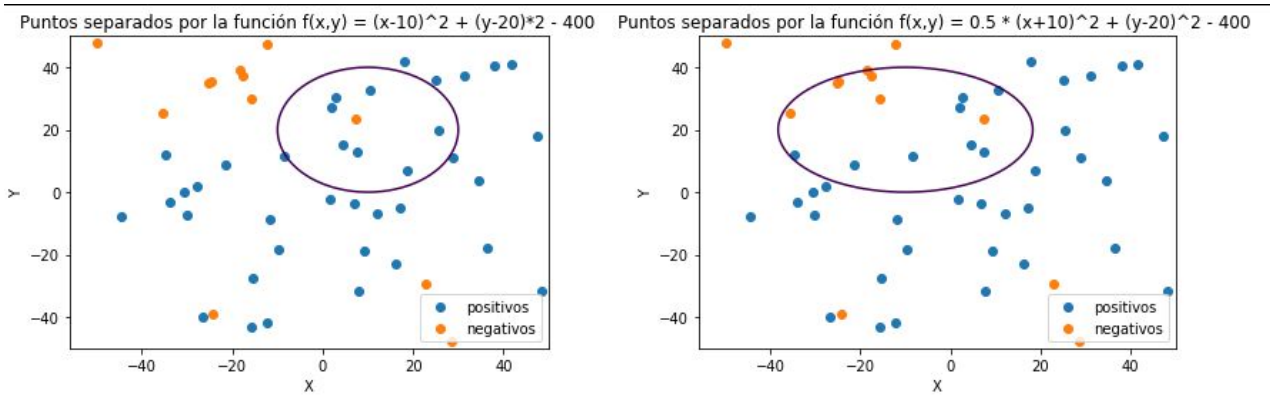


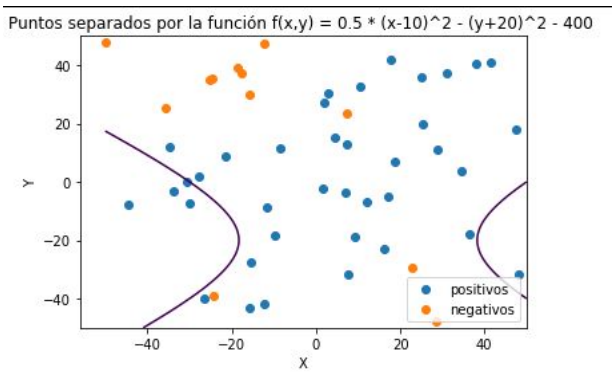
Apartado 3

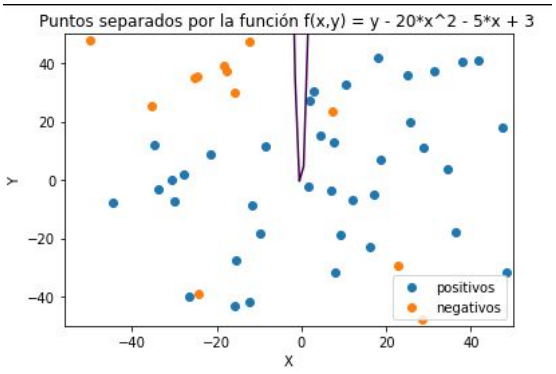
Supongamos ahora que las siguientes funciones definen la frontera de clasificación de los puntos de la muestra en lugar de una recta:

- $f(x,y) = (x-10)^2 + (y-20)^2 - 400$
- $f(x,y) = 0,5(x+10)^2 + (y-20)^2 - 400$
- $f(x,y) = 0,5(x-10)^2 - (y+20)^2 - 400$
- $f(x,y) = y - 20x^2 - 5x + 3$

Visualizar el etiquetado generado en 2b junto con cada una de las gráficas 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 clasificadores que la función lineal? ¿En qué ganan a la función lineal? Explicar el razonamiento.







En ninguno de los casos son mejores las funciones que la recta para estos puntos concretos. Como lo que tenemos es un conjunto de datos linealmente separable con ruido añadido, añadirle complejidad a la función no ayuda a clasificar mejor, ya que no resuelve el problema del ruido.
Si tuviéramos los datos distribuidos de manera diferente, por ejemplo agrupados en un círculo, una función más compleja si le ganaría a una función lineal, ya que una función lineal no puede separar bien un conjunto así, pero una función como las dos primeras sí puede.

2. Modelos lineales

Apartado 1. Algoritmo Perceptron

Implementar la función ajusta_PLA(datos,label,max_iter,vini) que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada datos es una matriz donde cada item con su etiqueta está representado por una fila 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 coeficientes del hiperplano.
Pseudocódigo del algoritmo PLA:

```
Inicializar vector de pesos con vini
Inicializar contador de iteraciones a 0
Inicializar contador de cambios para entrar al bucle

Itera mientras no se haya llegado a max_iter iteraciones o
no se hayan hecho cambios al vector de pesos después de
recorrer todos los datos:
     Inicializar el contador de cambios a 0
     Para cada elemento de datos:
```

```
        Si su clasificación usando el vector de pesos es
distinta del elemento de label correspondiente a ese dato:
            Modificar el vector de pesos
            Añadir uno al contador de cambios
    Añadir uno al contador de iteraciones
```

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.**

i) **Vector cero**

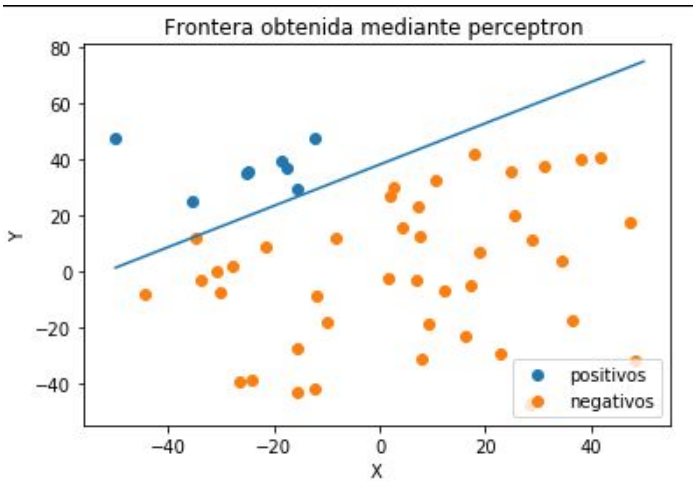
Usando el vector de ceros como punto inicial necesita 109 iteraciones.

ii) **Vectores de número aleatorios**

```
(b)
Número de iteraciones para el vector inicial  0 :  189
Número de iteraciones para el vector inicial  1 :  35
Número de iteraciones para el vector inicial  2 :  29
Número de iteraciones para el vector inicial  3 :  24
Número de iteraciones para el vector inicial  4 :  51
Número de iteraciones para el vector inicial  5 :  212
Número de iteraciones para el vector inicial  6 :  136
Número de iteraciones para el vector inicial  7 :  24
Número de iteraciones para el vector inicial  8 :  311
Número de iteraciones para el vector inicial  9 :  57
Número de iteraciones necesarias:  106.8
```

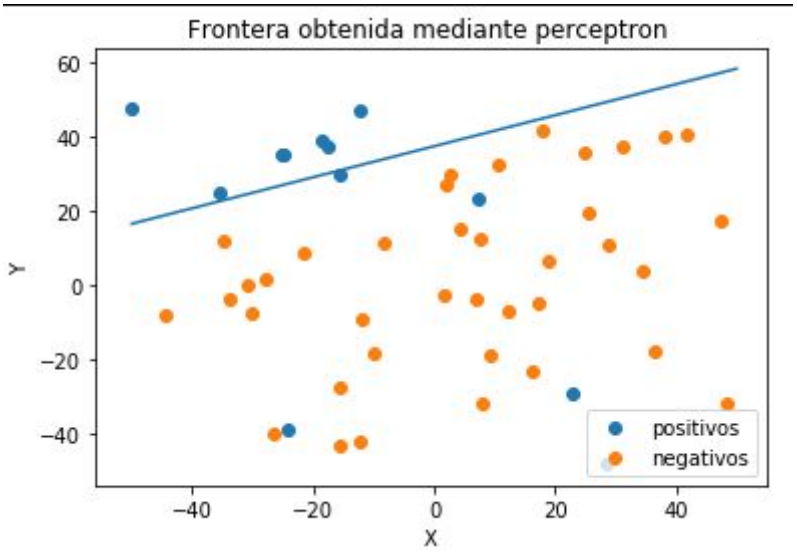
De media necesita aproximadamente el mismo número de iteraciones (106.8) que usando de punto de partida el vector de ceros. Sin embargo, si vemos cada iteración, vemos que por ejemplo la 3 y la 7 llegan en 24 iteraciones, que es bastante menos; pero con la 8 tarda casi 3 veces más que con el vector de ceros.

Frontera obtenida:



b) **Hacer lo mismo que antes usando ahora los datos del apartado 2b de la sección 1. ¿Observa algún comportamiento diferente? En caso afirmativo diga cual y las razones para que ello ocurra.**
Usando los datos del apartado 2b, que son los que tienen el 10% de error introducido, en todos los casos llega al número máximo de iteraciones que se le pasa como argumento. Esto se debe a que con el ruido, no va a encontrar nunca una recta que clasifique bien todos los datos, y como el algoritmo no para hasta que todos los datos estén bien clasificados, sigue iterando hasta que llega a max_iter.

Frontera obtenida:



Apartado 2. 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]×[0,2] con probabilidad uniforme de elegir cada x∈X. 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.

Los puntos han sido generados usando la función simula_unif, con N = 100, dimensión = 2, y rango entre 0 y 2; y después los he modificado para añadir una columna de unos.
Los dos puntos usados como frontera también han sido generados con simula_unif.

a) **Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD)**

Pseudocódigo:

```
Inicializar el contador de iteraciones a 0
Inicializar el vector de pesos (t)
Inicializar el vector de pesos (t-1)
Iterar mientras ||w(t-1)-w(t)||:
     Copiar pesos(t) en pesos(t-1)
     Desordenar los vectores de X e y
     Inicializar el tamaño de mini-batch
     Recorrer X en mini_batches, para cada batch:
          Actualizar el vector de pesos (t)
     Aumentar en 1 el contador de iteraciones
```

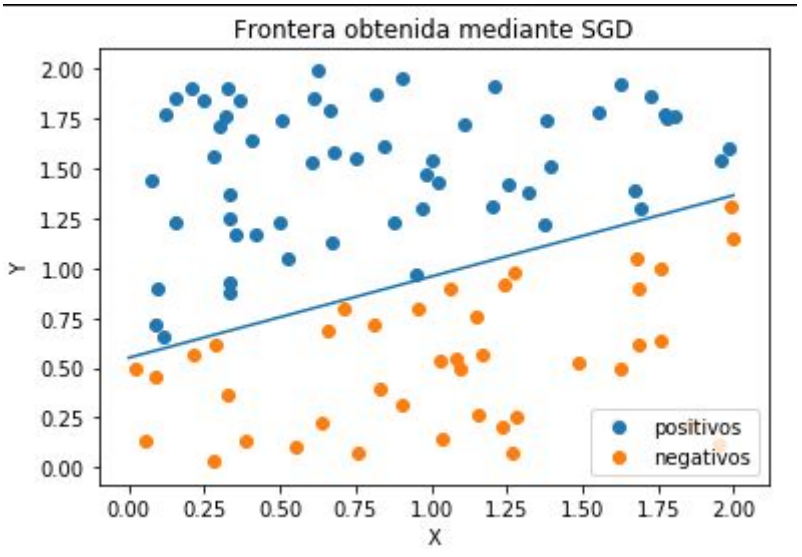
Para actualizar el vector de pesos uso la función actualizar_pesos(X,y,w,eta).
En esta función:

- Se calculan las probabilidades de que cada dato de X tenga label positivo haciendo el sigmoide del resultado de multiplicar X por el vector de pesos
- Se calcula el gradiente multiplicando $X^T \cdot$(predicciones-y)
- Se modifica el vector de pesos restándole gradiente*learning_rate y dividiendo por el tamaño de X

La función de sigmoide es:

$$sigmoide(x) = \frac{1}{e^{-x}+1}$$

Con implementación, el algoritmo converge en 305 iteraciones.



Para poder converger, he tenido que modificar el valor del learning rate a 0.1, ya que con 0.01 no funcionaba el algoritmo.

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

He generado 1000 puntos usando simula_unif, igual que los puntos usados para el entrenamiento.

Después he obtenido las etiquetas correspondientes; si el punto está por encima de la recta la etiqueta es +1, sino es 0.

Por último, usando el vector de pesos que hemos calculado en el apartado a, he calculado las predicciones para cada punto. Se multiplica w^T por cada fila; si es mayor que 0.5, tiene la etiqueta +1, y si es menor tiene la etiqueta 0.

Para calcular el error fuera de la muestra he hecho una función Err que toma como argumento las etiquetas correctas y las etiquetas calculadas con el vector de pesos. Lo que hace la función es restar los dos vectores, obteniendo un vector que contiene las diferencias entre etiqueta correcta y predicción para cada punto de X. Después toma este vector, y devuelve el número de elementos distintos de cero dividido por el número de puntos.

Usando esta medida de error, para estos puntos he obtenido un Eout de 0,05, es decir, un 5% de error.