Aprendizaje automático

Trabajo 2
Programación

Johanna Capote Robayna
5 del Doble Grado en Informática y Matemáticas
Grupo A

UNIVERSIDAD
DE GRANADA

Índice

1. Ejercicio sobre la complejidad de H y el ruido

Este ejercicio se desarrolla en el archivo `ejercicio1.py`.

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)`.

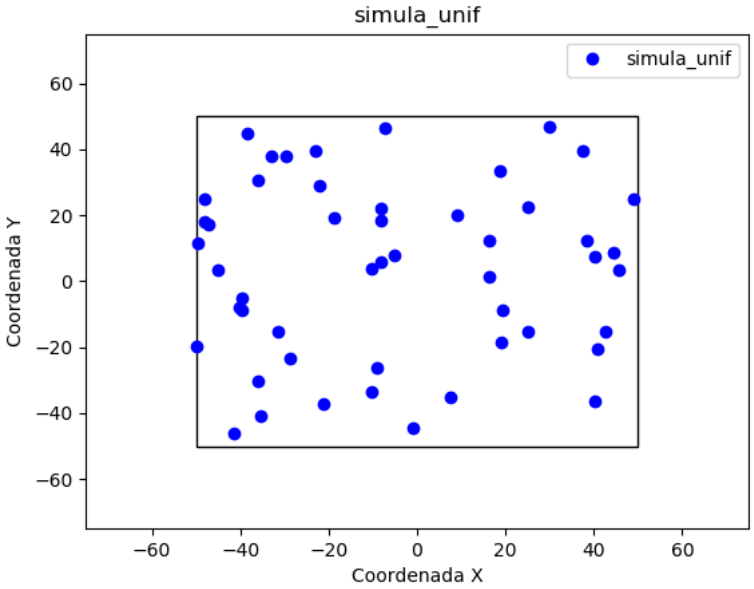

FIGURA 1: Nube de puntos generados uniformemente en $[-50, 50]x[-50, 50]$.

Para comprobar que efectivamente los puntos se encuentran dentro de las regiones se ha dibujado un rectángulo cuya esquina inferior izquierda es $[-50, -50]$ y con longitud de altura y anchura 100. Podemos comprobar que todos los puntos se encuentran dentro del rectángulo y que se distribuyen de manera uniforme.

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

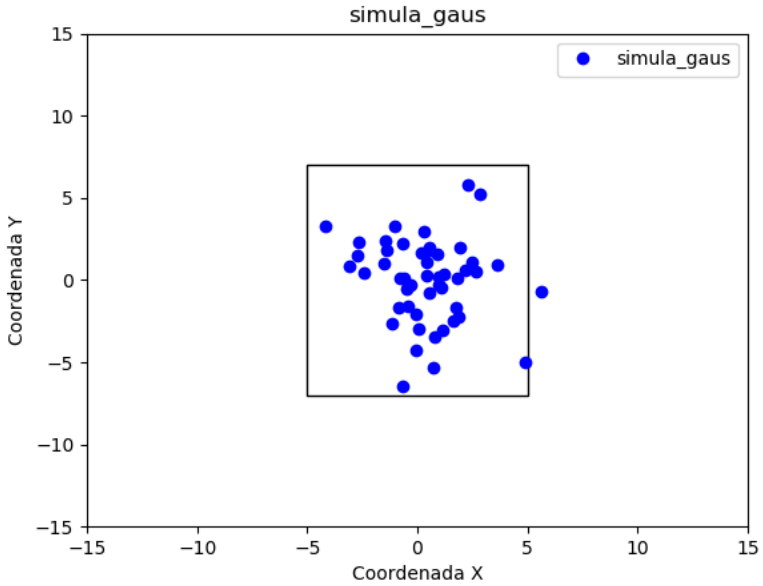

FIGURA 2: Nube de puntos generada a partir de una Gaussiana de media 0 y varianzas 5 y 7.

Por otro lado en este caso también se se ha dibujado un rectángulo cuya esquina inferior izquierda es $[-5, -7]$ y con longitud de altura 10 y anchura 14. En este caso la función no obliga a que lo puntos se encuentren dentro de los margenes, por lo que pueden haber puntos fuera pero pegados a la frontera como ocurre en este caso, por otro lado vemos como los puntos tienden a juntarse en el centro.

2. Vamos a valorar la influencia del rudio en la seleccion de la complejidad de la clase de funciones. Con ayuda de la función `simula_unif(100, 2, [-50, 50])` 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()`.

   a) Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello.

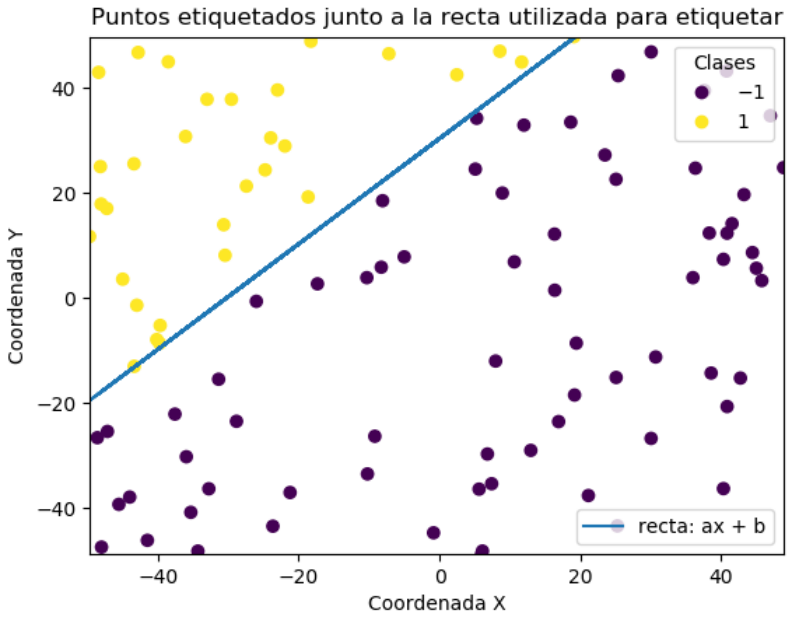

FIGURA 3: Nube de puntos etiquetados sin ruido.

b) Modifique de forma aleatoria un 10 de negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior.

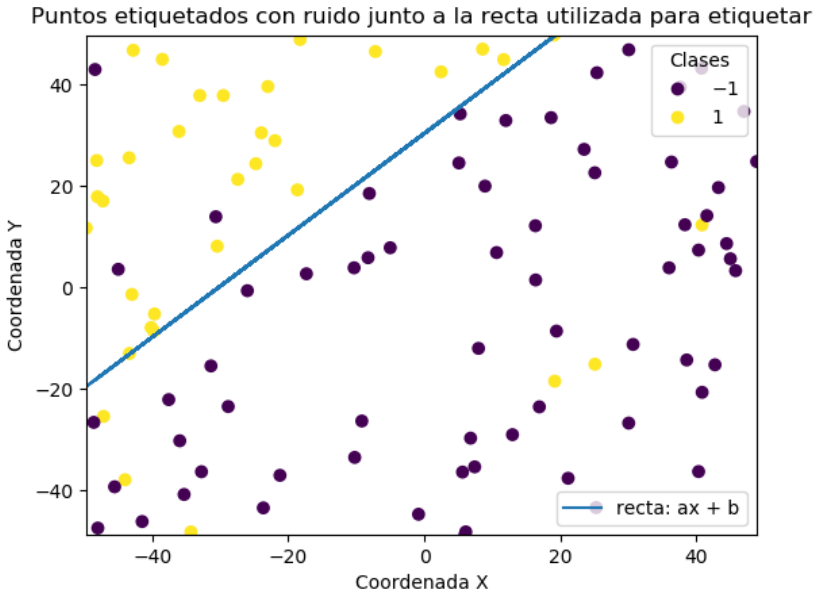

FIGURA 4: Nube de puntos etiquetados con ruido.

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

- $f(x, y) = (x10)^2 + (y20)^2400$
- $f(x, y) = 0,5(x + 10)^2 + (y20)^2400$
- $f(x, y) = 0,5(x10)^2(y + 20)^2400$
- $f(x, y) = y20x^25x + 3$

Visualizar el etiquetado generado en 2b junto con cada una de las gráficas de cada una de las funciones. Comparar 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? 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.

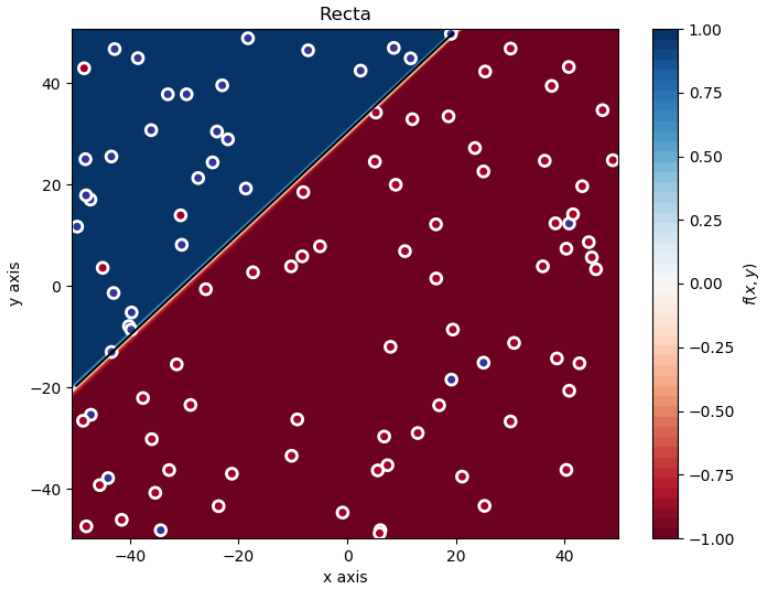

FIGURA 5: Recta $f(x, y) = y - ax - b$.

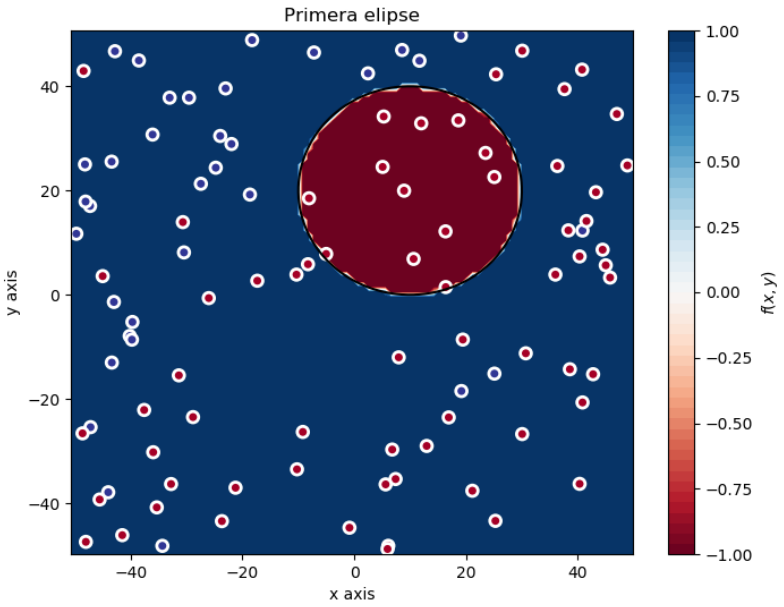

FIGURA 6: Elipse 1 $f(x, y) = (x10)^2 + (y20)^2 400$.

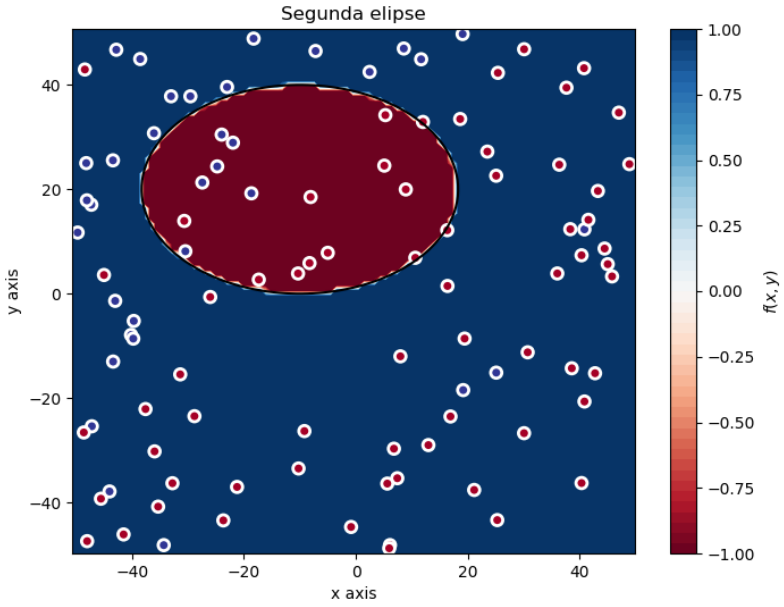

FIGURA 7: Elipse 2 $f(x, y) = 0,5(x + 10)^2 + (y20)^2 400$.

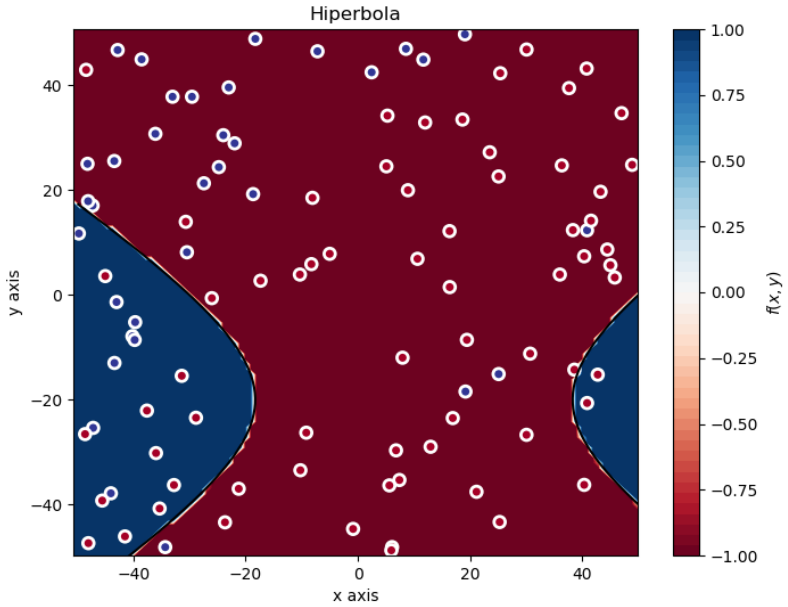

FIGURA 8: Hipérbola $f(x, y) = 0,5(x10)^2(y + 20)^2400$.

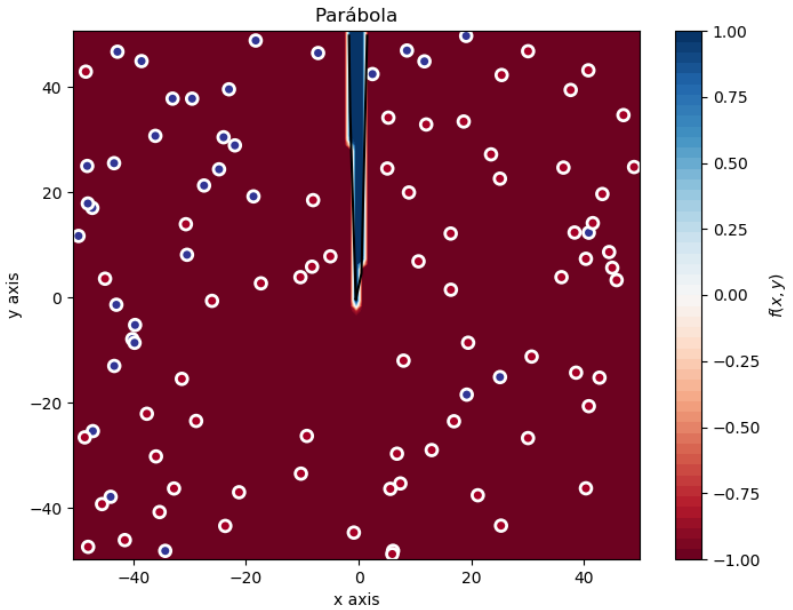

FIGURA 9: Parábola $f(x, y) = y20x^25x + 3$.

Para evaluar la bondad de estos clasificadores se pide que utilicemos como métrica el porcentaje de puntos bien clasificados.

```
Porcentaje de aciertos de la recta:
91.0
Porcentaje de aciertos de la primera elipse:
46.0
Porcentaje de aciertos de la segunda elipse:
38.0
Porcentaje de aciertos de la hiperbola:
61.0
Porcentaje de aciertos de la parábola:
66.0
```

Nos damos cuenta de que al aumentar la complejidad del clasificador no aumenta el porcentaje de aciertos, esto es debido a que hemos utilizado para etiquetar una recta y el porcentaje de ruido es bajo.

Por otro lado si nos fijamos en la parábola, esta clasifica la mayoría de puntos como "rojos"y ha dado la casualidad de que en nuestro conjunto la mayoría de puntos son rojos, por lo que este clasificador obtiene un porcentaje de aciertos bastante alto. Esto último es un error, ya que consideraríamos la parábola con un clasificador decente cuando a simple vista se comprueba que no. El error reside en que la métrica utilizada para establecer la bondad de los clasificadores no es la más oportuna.

2. Modelos lineales

Este ejercicio se ha implementado en el archivo `ejercicio2.py`.

2.1. Algoritmo Perceptron

1. 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.

Como el algoritmo es determinista solo ejecutamos el experimento con el vector cero una vez, ya que para un mismo punto inicial el algoritmo obtiene el mismo resultado.

Vemos que en el caso del vector cero necesita 34 épocas para converger mientras que empezando con distintos vectores aleatorios necesita de media 124.2 iteraciones. Podemos observar que el punto de inicio no influye en el número de iteraciones necesarias para converger.

```
Vector 0
Iteraciones: 34
Porcentaje correctos: 100.0

Vetores aleatorios
Valor medio de iteraciones necesario para converger: 124.2
Valor medio del porcentaje de aciertos: 100.0
```

En la siguiente gráfica vemos como el porcentaje de aciertos crece.

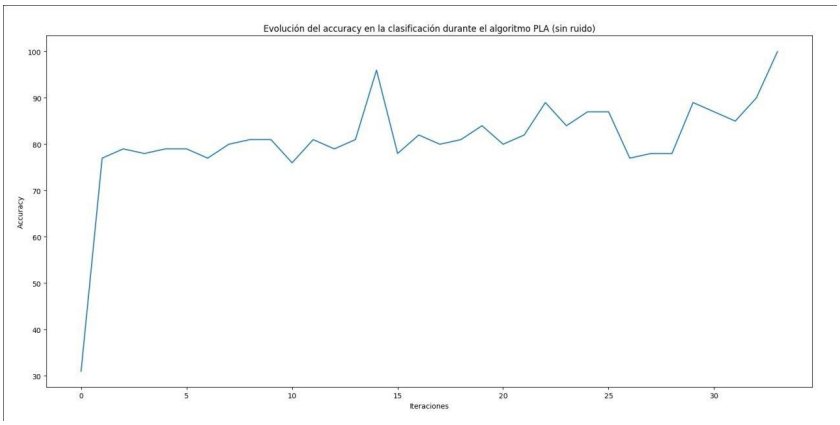

Figura 10: Evolución de la tasa de aciertos.

Además observamos que al no haber ruido en el conjunto de datos la recta obtenida con el algoritmo consigue separar los datos con un porcentaje de aciertos del 100 %.

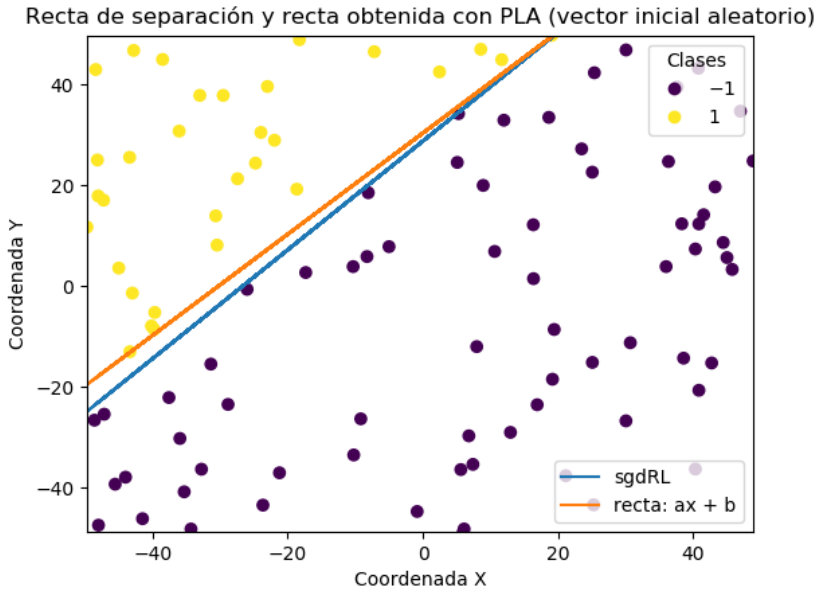

FIGURA 11: Recta separadora junto con recta obtenida con PLA.

2. 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.

   Al igual que antes ejecutamos el algoritmo con el vector cero solo una vez ya que el algoritmo es determinista.

   Observamos que en este caso al incorporar ruido en la muestra el algoritmo nunca llega a converger, es decir alcanza el número máximo de iteraciones (1000 épocas). Esto es lógico ya que es imposible llegar a separar correctamente todos los puntos mediante una recta.

```
Vector 0
Iteraciones: 1000
Procentaje correctos: 81.0


Vetores aleatorios
Valor medio de iteraciones necesario para converger: 1000.0
Valor medio del porcentaje de aciertos: 75.3
```

En la siguiente gráfica vemos como el porcentaje de aciertos oscila.

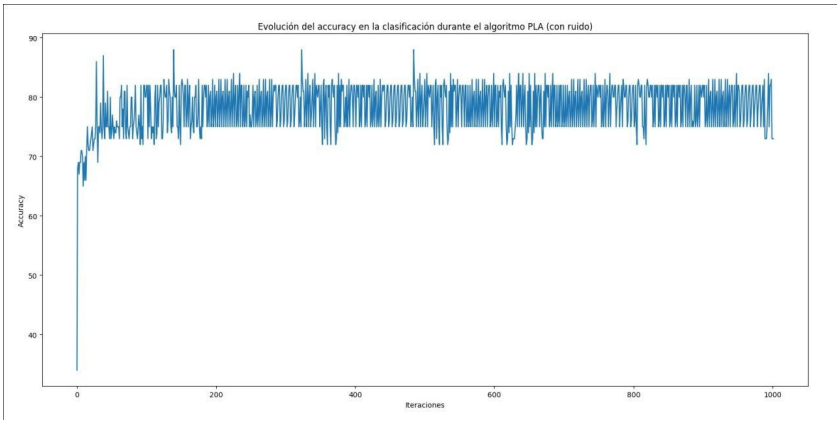

FIGURA 12: Evolución de la tasa de aciertos.

Comprobamos gráficamente los resultados obtenidos:

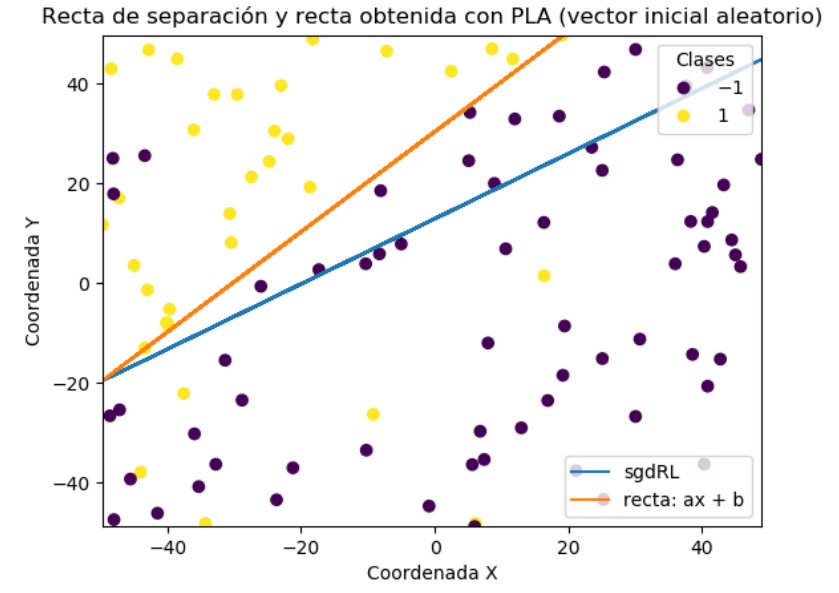

FIGURA 13: Recta separadora junto con recta obtenida con PLA.

2.2. Regresión logística

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

   Para implementar el algoritmo se han declarado dos funciones: `sgdRL` donde se encuentra el código princpial, en el que buscamos minimizar el error logístico con SGD, y `gradRL` donde se encuentra ya desarrollada la expresión del gradiente del error logistico:

   $$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} log(1 + e^{-y_n w^T x_n})$$

   $$\nabla_w E_{in}(w) = \frac{-1}{N}\sum_{n=1}^{N} \frac{y_n x_n}{1 + e^{y_n w^T x_n}}$$

2. Usar la muestra de datos etiquetada para encontrar nuestra solución g y estimar E_{out} usando para ello un número suficientemente grande de nuevas muestras (> 999).

   Con la muestra de datos etiquetadas utilizamos el algoritmo diseñado en el apartado anterior para obtener la función g. A continuación, generamos un conjunto *test* de datos uniformemente distribuidos de tamaño 1000, lo etiquetamos con la función del ejercicio 1 y calculamos el error logístico de g. Además añadimos el porcentaje de aciertos para comprobar la bondad del resultado.

   ```
   Bondad del resultado para grad. descendente estocastico:

   Eout:  0.11378017016196812
   Porcentaje de aciertos 98.6
   ```

   Vemos que el error logistico es muy pequeño y que el porcentaje de puntos mal clasificados es solo un 1,4 %.

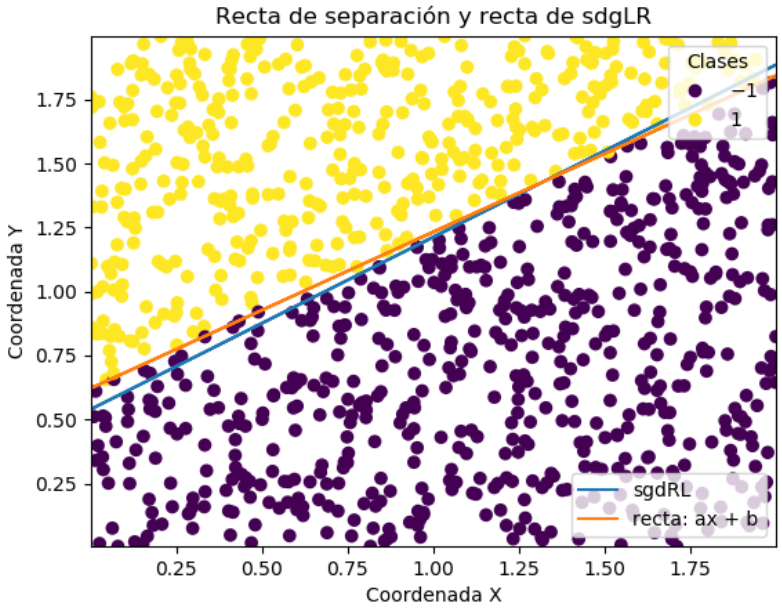

FIGURA 14: Conjunto de datos test junto con la recta separadora y la generada con sgdRL

3. BONUS

Este apartado se ha desarrollados en el archivo `bonus.py`.

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

   Planteamos un problema de clasificación con los siguientes elementos:

   - Espacio de entrada: $X = 1x\mathbb{R}^2$
   - Espacio de etiquetas: $Y = \{-1, 1\}$
   - Espacio de funciones candidatas: $H = \{f : X \rightarrow Y | \exists w \in \mathbb{R}^3 f(x) = w^T x\}$
   - Conjunto de muestra $D \subset XxY$ muestra aleatoria simple de tamaño 1194 de la distribución desconocida $P(X, Y)$ tal que $f(x) = P(y|x)$ donde f es la función que etiqueta.

- Criterio de aprendizaje (ERM): $w^* = argmin_w \frac{1}{1194} \sum_{i=1}^{1194} loss_w(x_i, y_i)$
- Función de pérdida: $loss_w(x, y) = [[w^T x = y]]$

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

   Como algoritmo para obtener la regresión lineal hemos utilizado el de la pseudo-inversa.

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

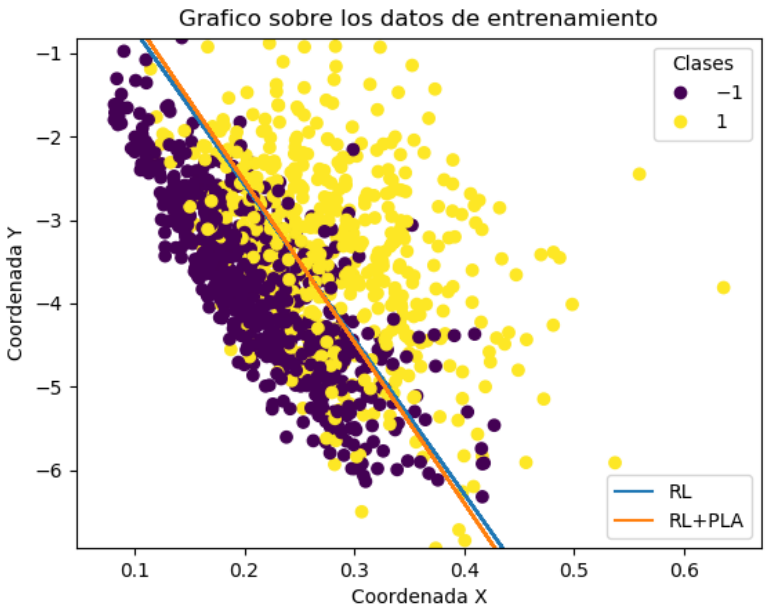

FIGURA 15: Conjunto de datos de entrenamiento junto con las funciones RL y RL+PLA.

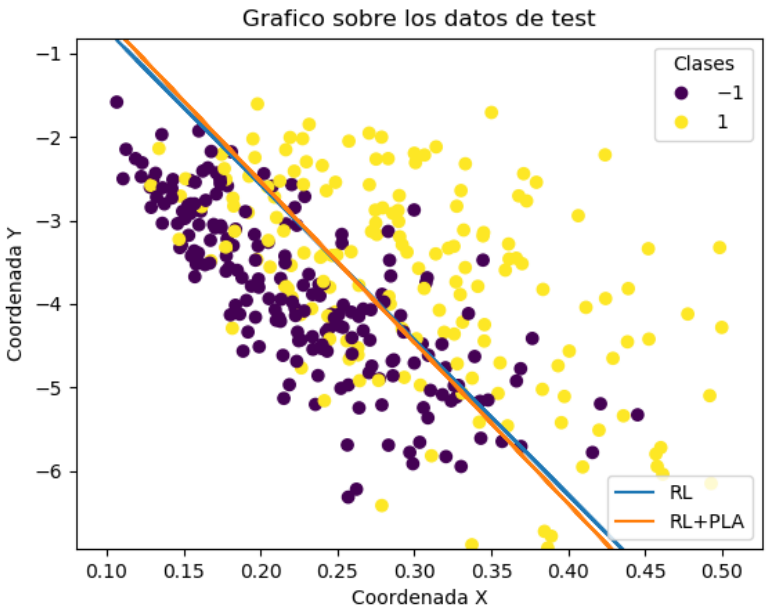

FIGURA 16: Conjunto de datos de test junto con las funciones RL y RL+PLA.

b) Calcular E_{in} y E_{test} (error sobre los datos de test).

Utilizamos el conjunto de test dado para calcular el error de E_{test}.

```
Bondad del resultado para RL:

Ein:  0.22780569514237858
Etest:  0.2513661202185792

Bondad del resultado para RL + PLA:

Ein:  0.22529313232830817
Etest:  0.25409836065573765
```

Obsevamos que apenas hay mejoría, aunque los resultados ya antes de aplicarle el PLA eran bastante buenos, por lo que no nos resulta raro que no haya mucha mejora.

c) Obtener cotas sobre el verdadero valor de E_{out} . Pueden calcularse dos cotas una basada en E_{in} y otra basada en E_{test} . Usar una tolerancia $\delta = 0.05$. ¿Que cota es mejor?

Por la desigualdad de Hoeffding sabemos que, con probabilidad $1 - \delta$:

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N_{in}} \cdot ln\frac{2M_{in}}{\delta}}$$

$$E_{out}(g) \leq E_{test}(g) + \sqrt{\frac{1}{2N_{test}} \cdot ln\frac{2M_{test}}{\delta}}$$

Donde N es el número de puntos en cada caso y M es el cardinal del conjunto H. En el caso de E_{in}, M_{in} sería infinito, por lo que introducimos el máximo de precisión en *bits* necesarios para representar tres flotantes ($2^{64.3}$). Por otro lado el valor de M_{test} es 1, porque ya hemos estimado la recta previamente y hemos elegido una concreta.

Veamos los resultados:

```
Cota superior de Eout con Ein:  0.46461547451143265
Cota superior de Eout con Etest:  0.32508746408835315
```

Vemos que la mejor cota es E_{test} porque es más pequeña que la de E_{in}.