Aprendizaje Automático
Trabajo 2:
Programación

Rafael Vázquez Conejo

Abril 2020

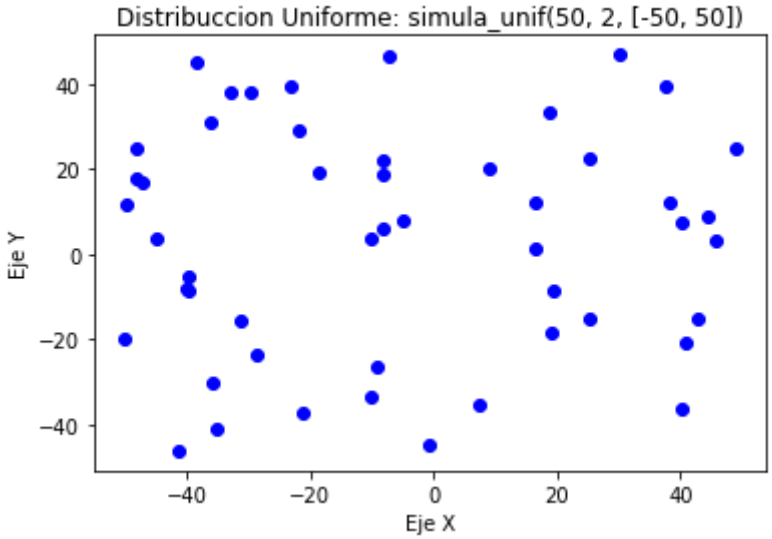
Índice

1. Ejercicio sobre la Complejidad de H y el Ruido

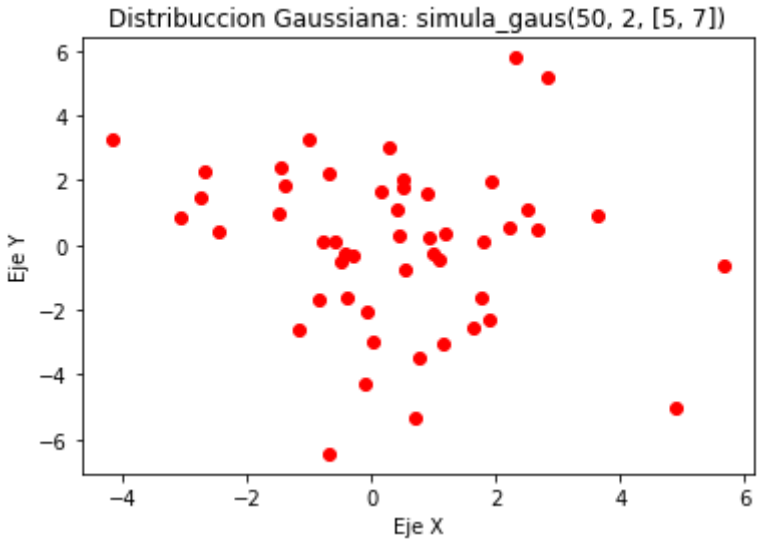
1.1. Dibujar una gráfica con la nube de puntos de salida correspondiente.

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

Para generar las nubes de puntos en ambos apartados he empleado las dos funciones proporcionadas para cada caso, las cuales son simula_unif(), función que genera una distribución uniforme en un determinado rango dado, y simula_gauss(), esta genera una distribución gaussiana.



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

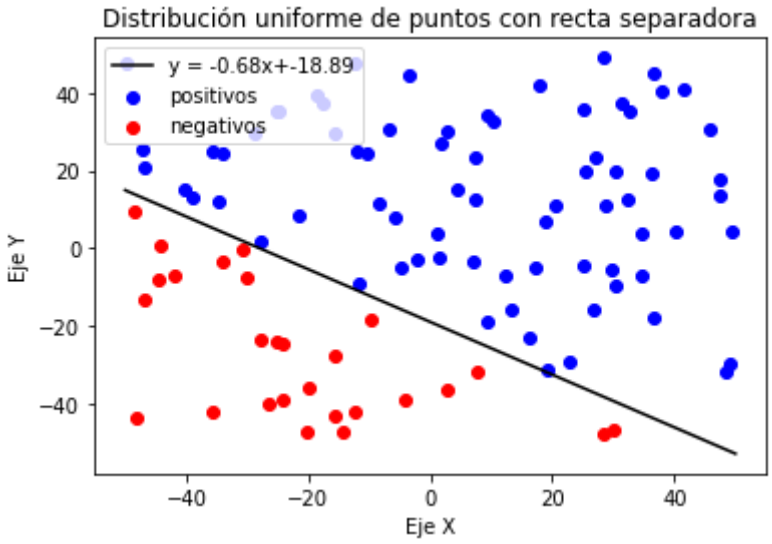


1.2. Vamos a valorar la influencia del rudio en la seleccion de la complejidad de la clase de funciones. Con ayuda de la función simula_unif (100, 2, [−50, 50]) generar una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función f (x, y) = y − ax − b, es decir el signo 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)

Tras generar la muestra de puntos he utilizado la función f(x, y) = y – ax – b para calcular la etiqueta de cada punto. Necesitaremos para ello obtener una recta aleatoria comprendida en nuestro rango, recta que obtendremos gracias a la función simula_recta() que nos devuelve una pendiente (a) y un termino independiente (b). Una vez tenemos nuestra recta podemos realizar el etiquetado de los puntos, el cual dependerá de la distancia de cada punto a la recta.

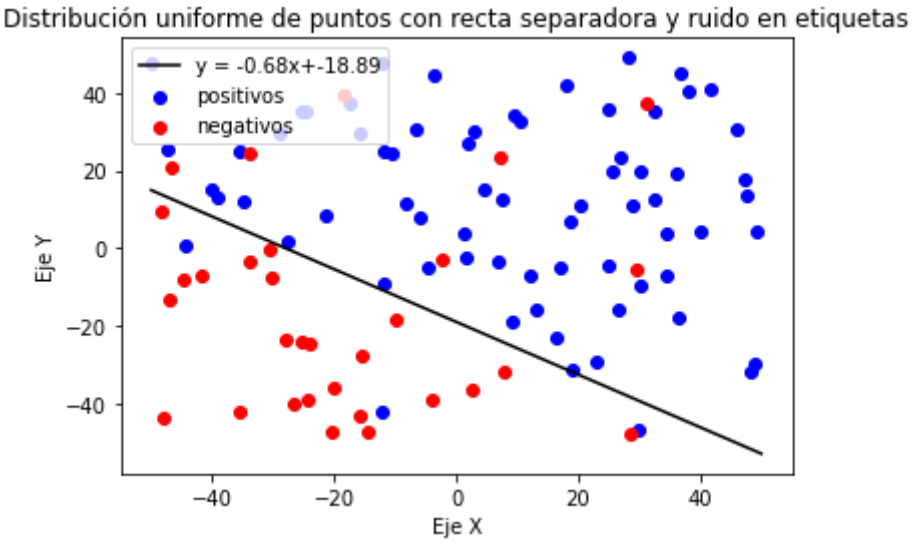
Los coeficientes de la recta aleatoria obtenida y = ax + b son → a = -0.677 y b = -18.89 aprox.



Efectivamente si primero generamos los coeficientes de la recta aleatoria, y tras ello generamos las etiquetas de los puntos a partir de ella, la recta que obtenemos debe separar todos los puntos perfectamente.

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)

Introducimos un 10% de ruido a cada grupo de puntos (negativos y positivos) obtenidos anteriormente. Debido a este ruido podemos observar los ejemplos mal clasificados respecto a nuestra recta.



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) =*** $(x-10)^2+(y-20)^2-400$
- ***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 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.

Todas las funciones presentadas describen superficies curvadas definidas en el espacio, lo sabemos ya que poseen 2 variables. Pintaremos las gráficas en 2D para su observación junto con el conjunto de puntos, para ello obtengo las curvas cuyo intersección con el plano z es igual a 0, es decir, para las que se cumple que f(x, y) = 0.

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

Calculo f(x, y) = 0, obteniendo:

$$(x-10)^2+(y-20)^2-400 = 0$$

Despejo la y:

$$y=20\pm\sqrt{300+20x-x^2}$$

A continuación debemos encontrar el intervalo de valores para los cuales este polígono de segundo grado, que encontramos dentro de la raíz, tenga valores positivos (300 +20x + x^2 debe ser positivo).

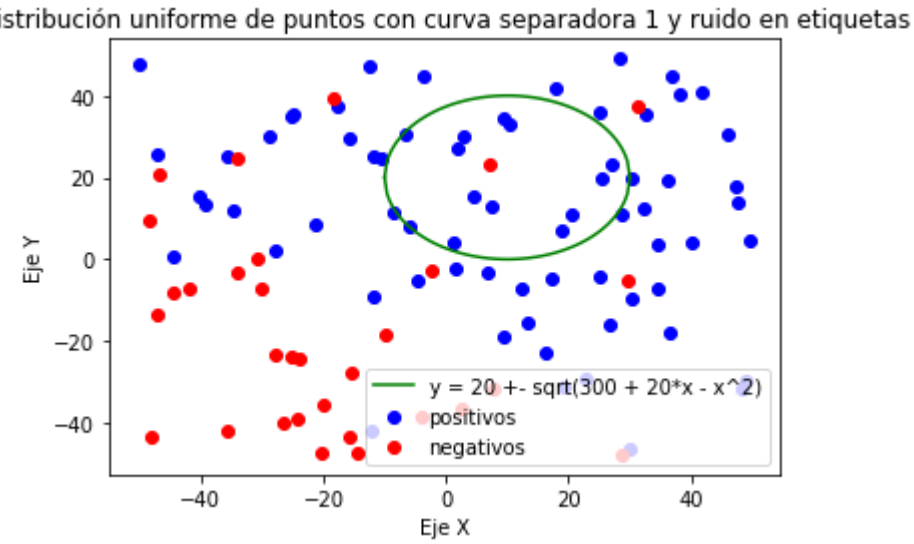
La razón es que las raíces cuadradas de número negativos no están definidas en $\mathbb{R}$.

Observamos que el coeficiente principal ($-x^2$) es negativo, por este motivo la parábola que describe será cóncava, y por lo tanto las raíces del polígono ($x = x_o$, $x = x_1$) serán positivas dentro del intervalo [x_o, x_1].

Las raíces que obtenemos tras despejar son x = 10 y x = 30, aplicando lo anterior nuestro intervalo será [-10, 30].

Finalmente separamos la expresión en dos diferentes, ya que “y” puede tomar 2 valores de “x”. Definimos una función para cada expresión:

- ➔ $y_2 = y=20+\sqrt{300+20x-x^2}$
- ➔ $y_2 = y=20-\sqrt{300+20x-x^2}$



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

Seguimos el mismo procedimiento que en el caso anterior, igualamos f(x, y) a 0:

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

Despejo la y:

$$y=20\pm\sqrt{-(\frac{1}{2}x^2)-10x+350} = y=20\pm\sqrt{\frac{1}{2}}\sqrt{-x^2-20x+700} =$$

$$y=20\pm\frac{1}{2}\sqrt{2}\sqrt{-x^2-20x+700} = y=\frac{1}{2}(40\pm\sqrt{2}\sqrt{-x^2-20x+700})$$

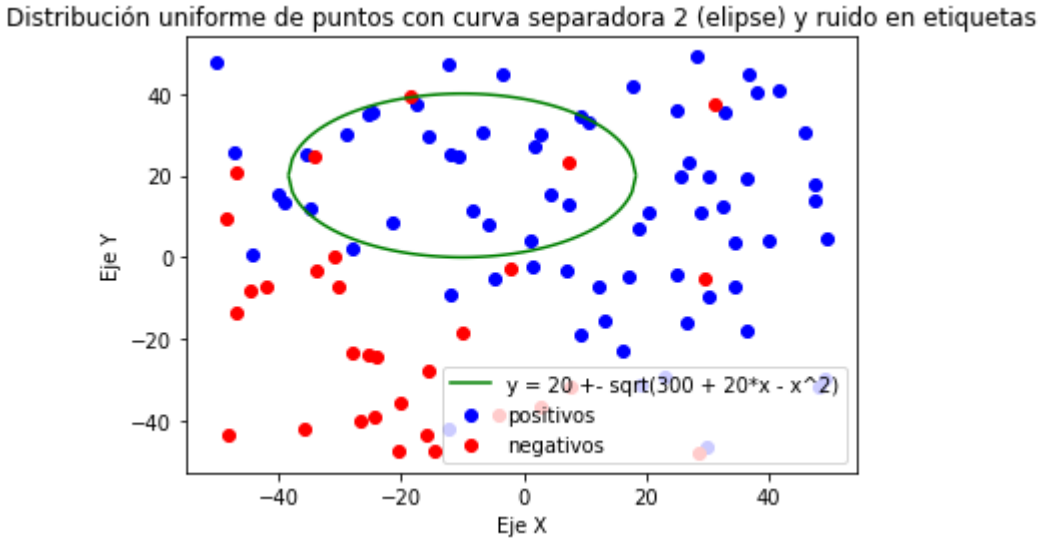
Como en el caso anterior tenemos un polinomio de segundo grado cóncavo dentro de la raíz.

Las raíces en este caso son x = -38.28 y x = 18.28, por ello el intervalo que debemos representar en este caso es [-38.28, 18.28]

Y por la misma razón que antes separamos la ecuación en dos expresiones:

➔ $y_2 = y=\frac{1}{2}(40+\sqrt{2}\sqrt{-x^2-20x+700})$

➔ $y_2 = y=\frac{1}{2}(40-\sqrt{2}\sqrt{-x^2-20x+700})$



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

Repetimos los procesos anteriores, igualamos f(x, y) a 0:

$$0{,}5(x-10)^2-(y+20)^2-400 \quad = 0$$

Despejo la y, de una forma muy similar al caso anterior:

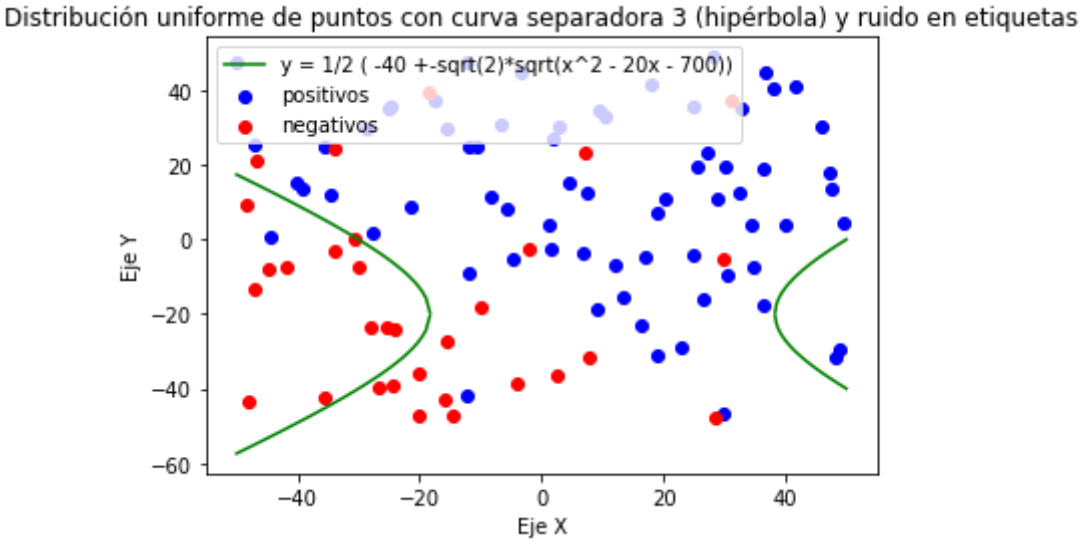
$$y=\frac{1}{2}(-40\pm\sqrt{2}\sqrt{x^2-20x-700})$$

En este caso, a diferencia de los anteriores, nos encontramos con un coeficiente principal del polinomio positivo, por lo tanto el polinomio es convexo.

Sus raíces son x = -18.29 y x = 38.29, en este caso el intervalo en el que toma valores positivos y debemos representar será (- ∞ , -18.29] U [38.29, + ∞).

No representaremos hasta el infinito, sino que acotaremos el intervalo al mismo en el que hemos definido los puntos, es decir, [-50, 50], por lo cual obtenemos el siguiente intervalo:

[-50, -18.29] U [38.29, 50]

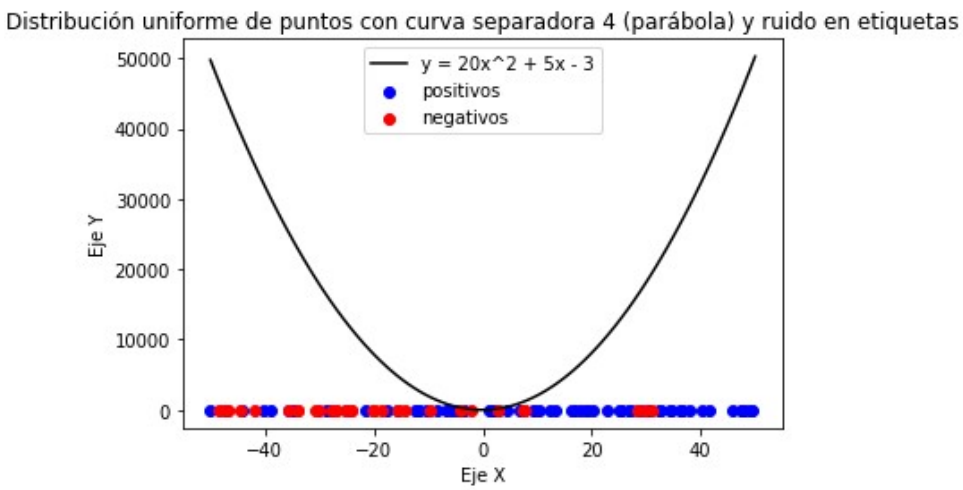


- f (x, y) = $y-20x^2-5x+3$

Repetimos los procesos anteriores, igualamos f(x, y) a 0:

$$y-20x^2-5x+3 \quad = 0 \qquad => \qquad y=20x^2+5x-3$$

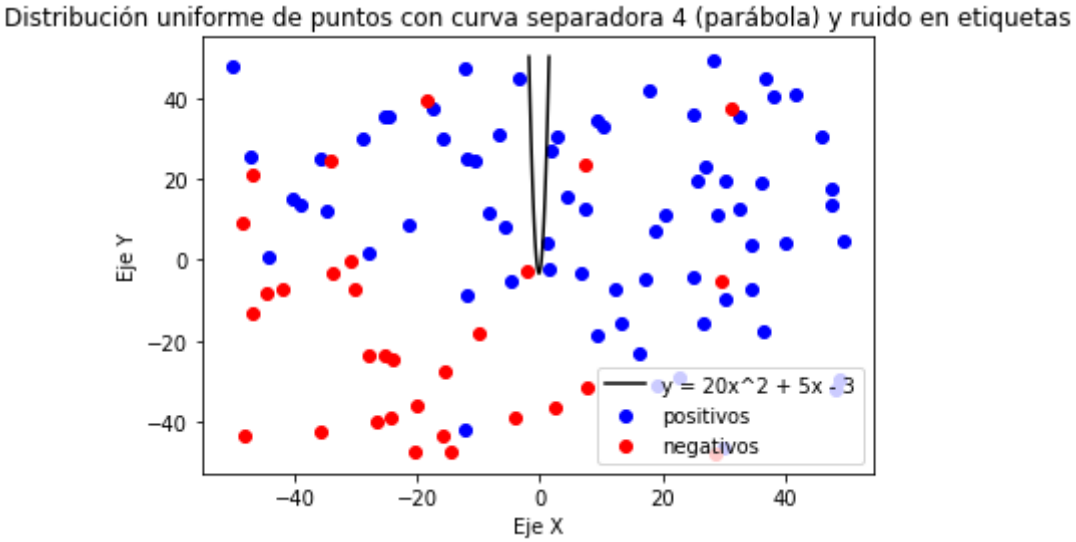
En este caso no existen restricciones sobre el dominio en el que se debe mostrar la función, usaremos el mismo rango en el que se sitúan los puntos [-50, 50].



Observamos que la parábola crece muy rápido lo que nos impide observarla junto a los puntos. Para solucionarlo limitaremos el dominio de los valores que puede tomar la componente x, tomando solo aquellos que alcanzan un valor y = 50.

$20x^2+5x-3=50$ => $20x^2+5x-53=0$

De esta forma obtenemos como raíces, x = -1.76 y x = 1.51 aproximadamente. Nuestro intervalo será [-1.76, 1.51]



Conclusión

Estas últimas cuatro funciones utilizadas para “separar” los datos de nuestra muestra corresponden a una circunferencia, una elipse, una hipérbole y una parábola, todos casos en los que aparecen variables elevadas al cuadrado. Es evidente que estas funciones por lo tanto son más complejas, y si ajustamos correctamente los coeficientes podrían servir como fronteras de clasificación para datos que no sean linealmente separables.

Sin embargo ninguna de las curvas dibujadas en los cuatro casos hacen una buena separación en nuestro conjunto de datos. Partimos de que nuestro conjunto no es linealmente separable, pero debemos recordar que no lo es porque hemos introducido una pequeña cantidad de ruido aleatoria en nuestro conjunto inicial, que sí era linealmente separable. Es decir, hemos convertido un conjunto de datos linealmente separable en uno que no lo es porque hemos introducido ruido en él, y no porque siga un comportamiento cuadrático o siguiendo una curva tónica (casos en los que no sería linealmente separable).

Además, aunque alguno de estas funciones nos sirviera como frontera de clasificación de nuestros datos, deberíamos ajustar sus coeficientes con un algoritmo de aprendizaje.

2. Modelos Lineales

2.1. Algoritmo Perceptron: Implementar la función

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.

El algoritmo Perceptron es un método de regresión lineal.
PLA es un algoritmo sencillo en el que recorremos diversas veces el conjunto de datos, y cada vez que encontramos un dato “x” mal clasificado actualizamos el vector de pesos “w” de nuestro modelo lineal, siguiendo la siguiente regla:

$$w(i + 1) = w(i) + x(i) * y(i)$$

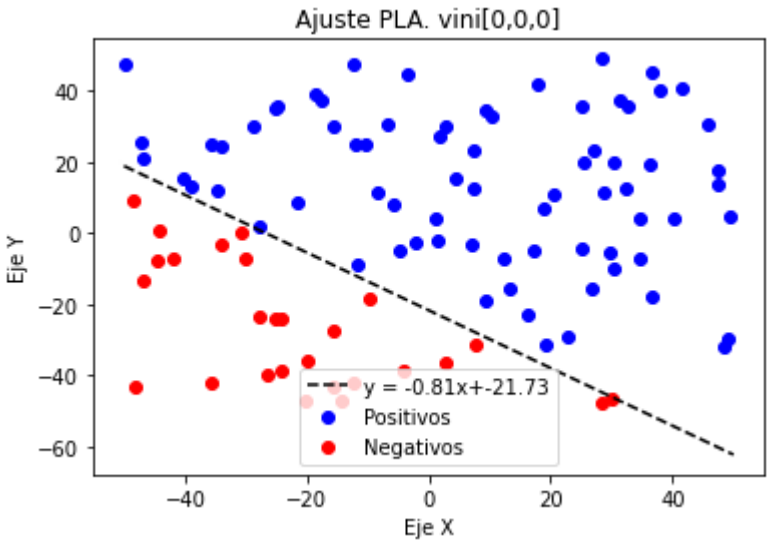
Siendo “i” la iteración actual e “y” la etiqueta de nuestro dato de “x”
Encontraremos un dato mal clasificado cuando la etiqueta obtenida mediante $sign(w^t * x_i)$ sea distinta a la etiqueta real del valor. En caso de que la etiqueta esté bien clasificada el vector de pesos no cambia. Esto nos permitirá redireccionar el vector de pesos de forma que pueda obtener una mejor clasificación de los datos.
Nuestro algoritmo PLA recibe como parámetros los pedidos en el enunciado y devuelve como salida tanto el vector de pesos “w” de nuestro modelo lineal, como el número de iteraciones.
Respecto a la condición de parada hemos establecido dos. La primera se dará cuando el vector de pesos “w(i + 1)” al final de una iteración sea igual a “w(i)” , ya que esto supone que tenemos una clasificación correcta de todos los datos. La segunda condición de parada es el número máximo de iteraciones que establecemos.
Mencionar que hemos establecido un máximo de iteraciones de 1000 en todos los casos.

Ejecución del algoritmo PLA con datos sin ruido.

Tomamos como conjunto de datos el simulado en el apartado 2a del ejercicio 1 (sin ruido). Sobre este conjunto de datos realizamos dos experimentos, a) w inicial a 0.0 y b) w inicial aleatoria entre [0, 1].

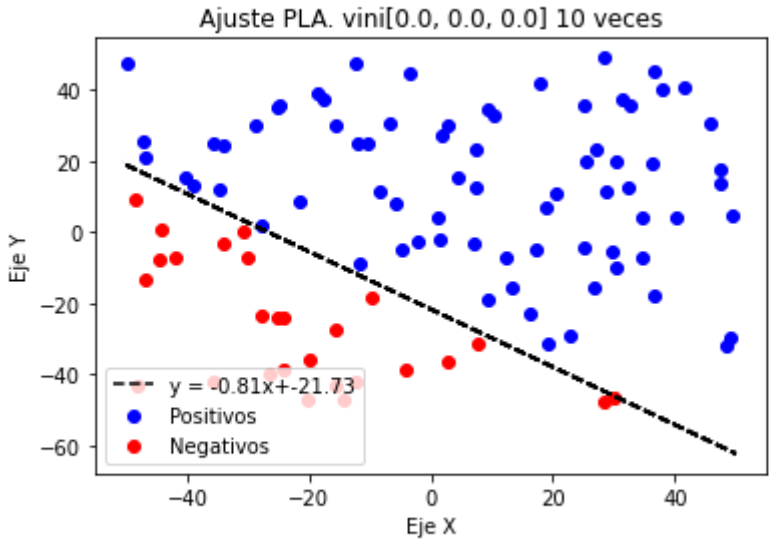
a) w = [0.0, 0.0, 0.0]

Vector de pesos inicial de 0 (vini = 0). PLA clasifica nuestro conjunto de datos tras un total de 61 iteraciones.



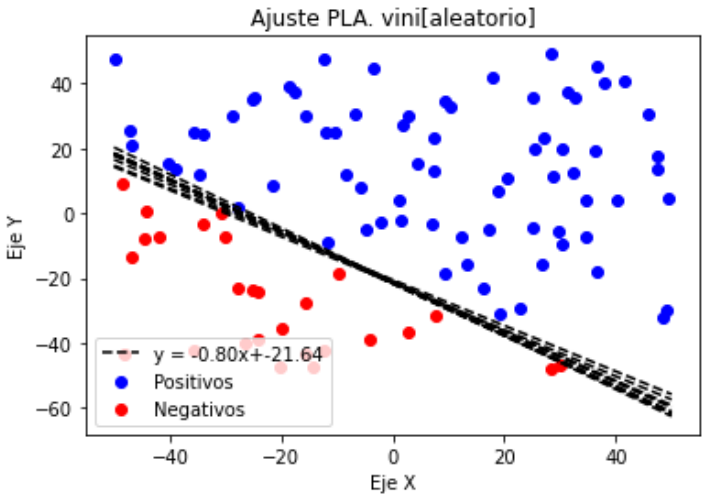
Al ejecutarlo 10 veces siempre obtengo el mismo resultado. Comprobamos que si empezamos por el mismo punto obtenemos los mismos resultados por ello podemos intuir que tendrá gran importancia el punto de inicio que tomemos.

```
Si lo ejecuto 10 veces:
```




```
Numero de iterationes (vector 0):  [61, 61, 61, 61, 61, 61, 61, 61, 61, 61]
Media del número de iteraciones: 61.0
```

b) w aleatorio entre [0, 1]




```
Numero de iterationes (vector inicial aleatorio):  [59, 59, 118, 63, 62, 63, 68, 63, 56, 63]
Media del número de iteraciones: 67.4
```

Volvemos a aplicar el PLA sobre el mismo conjunto de datos unas 10 veces más, pero esta vez con pesos iniciales aleatorios con valores entre 0 y 1. Como tenemos datos linealmente separables siempre obtendremos una clasificación perfecta, siempre y cuando el número de iteraciones máximo sea lo suficientemente grande.

Al variar el punto de inicio obtenemos ajustes distintos en cada caso, algunos consiguen la separación en el mismo número de iteraciones que otros, y todos hacen una separación perfecta de los datos.

Observamos una variación considerable en algunos casos del número de iteraciones necesarias para finalizar, lo que nos da una idea de la influencia del valor inicial al ajustar un modelo lineal con PLA.

En la gráfica representada se dibujan las 10 rectas de separación obtenidas sobre el conjunto de datos, podemos observar la cercanía que existe entre ellas.

Resultados PLA con datos sin ruido		
Valor Inicial	Ajuste Obtenido	Iteraciones
(0.69, 0.55, 0.86)	(497, 18, 23)	59
(0.74, 0.26, 0.99)	(511, 18, 24)	59
(0.57, 0.64, 0.49)	(711, 24, 34)	118
(0.89, 0.87, 0.85)	(523, 19, 24)	63
(0.65, 0.59, 0.28)	(512, 20, 24)	62
(0.99, 0.79, 0.50)	(524, 19, 24)	63
(0.77, 0.12, 0.92)	(545, 18, 25)	68
(0.04, 0.23, 0.84)	(526, 19, 24)	63
(0.30, 0.48, 0.65)	(486, 18, 23)	56
(0.14, 0.30, 0.89)	(526, 19, 24)	63

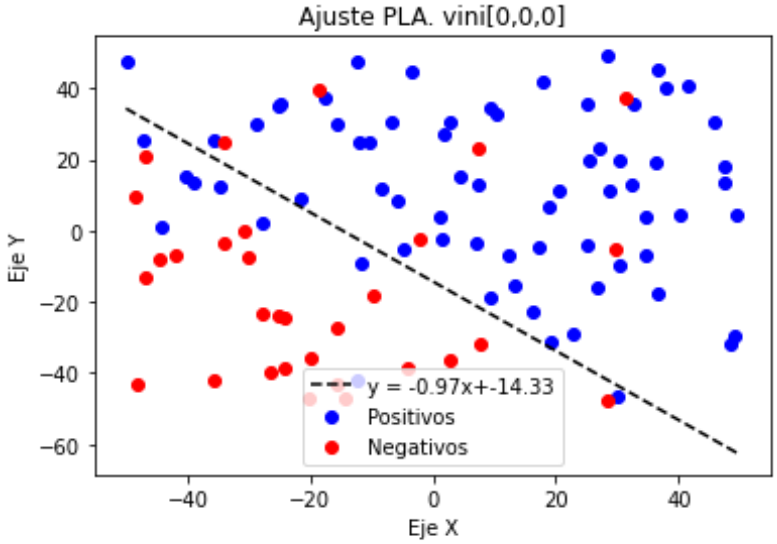
Introduzco valores aproximados para facilitar la observación de los resultados y sencillez de copia de dichos valores. En la tabla podemos confirmar la cercanía de algunos ajustes obtenidos, tal y como se observa en la gráfica.

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.

Hemos introducido ruido en los datos por lo que ya no son linealmente separables y PLA nunca conseguirá una clasificación perfecta.

a) w = [0.0, 0.0, 0.0]

Caso en el que vini = 0, observamos que el resultado con estos nuevos datos del apartado 2b es muy distinto al obtenido con los de 2a. Lo más destacable es que el número de iteraciones que emplean es el mismo que el fijado como máximo de iteraciones, es decir, 1000 en nuestro caso.

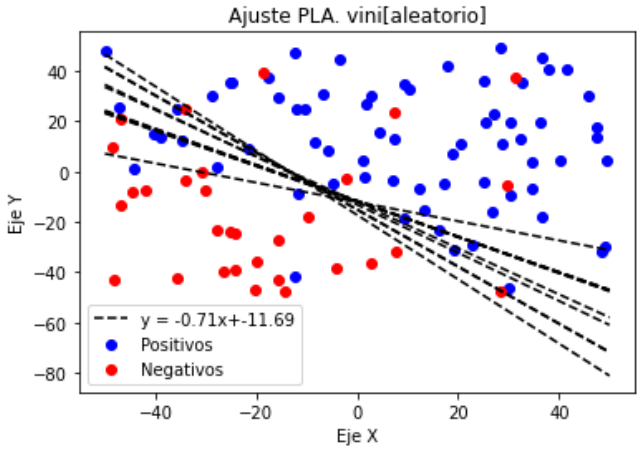



```
Numero de iterationes (vector inicial a 0):  1000
```

Si ejecutamos 10 veces este caso en el que empezamos por el mismo punto obtendremos el mismo resultado las 10 veces como pasaba en el apartado anterior.

b) w aleatorio entre [0, 1]

Repetimos ahora el proceso en 10 ejecuciones, esta vez con diferentes vectores de peso iniciales, con valores aleatorios entre 0 y 1.




```
Numero de iterationes (vector inicial aleatorio):  [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000,
1000]
Media del número de iteraciones: 1000.0
```

De nuevo, como en el apartado anterior, hemos representado todas las rectas resultantes en la misma representación de la muestra.

Destaca el hecho de que los resultados siempre llegan al máximo número de iteraciones establecido. Esto es debido a que, como ya hemos mencionado, los datos no son linealmente separables y por ello nuestro algoritmo PLA ya no es capaz de terminar su ejecución encontrando una clasificación perfecta. No importa cuando aumentemos el número de iteraciones máximas, no la encontrará.

Resultados PLA con datos con ruido		
Valor Inicial	Ajuste Obtenido	Iteraciones
(0.88, 0.51, 0.27)	(634, 20, 52)	1000
(0.88, 0.50, 0.33)	(638, 39, 54)	1000
(0.89, 0.20, 0.82)	(635, 47, 42)	1000
(0.84, 0.52, 0.16)	(636, 47, 41)	1000
(0.95, 0.59, 0.89)	(648, 48, 53)	1000
(0.57, 0.37, 0.64)	(644, 47, 37)	1000
(0.13, 0.41, 0.36)	(635, 37, 53)	1000
(0.15, 0.86, 0.10)	(635, 45, 47)	1000
(0.95, 0.17, 0.44)	(631, 37, 52)	1000
(0.16, 0.88, 0.48)	(638, 38, 54)	1000

Solo he copiado valores aproximados para facilitar la creación de la tabla. Los datos con su valor completo se pueden obtener desde terminal al ejecutar el programa.

Los ajustes obtenidos, al igual que en el apartado anterior, son relativamente similares. La causa de esto probablemente sea la proximidad de los valores iniciales, ya que estos se sitúan es un intervalo de 0 a 1.

2.1. Regresión Logística

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

- ***Inicializar el vector de pesos con valores 0.***
- ***Parar el algoritmo cuando ||w (t−1) − w (t) || < 0,01, donde w (t) denota el vector de pesos al final de la época t. Una época es un pase completo a través de los N datos.***
- ***Aplicar una permutación aleatoria, 1, 2, . . . , N , en el orden de los datos antes de usarlos en cada época del algoritmo.***
- ***Usar una tasa de aprendizaje de η = 0,01***

La implementación del SGD es similar a la de la práctica 1. Como salida la función devolverá el vector de pesos del ajuste obtenido y el número de épocas que ha necesitado el algoritmo (iteraciones).

Este algoritmo consiste en la búsqueda del vector de pesos que obtenga una buena clasificación de los datos. El algoritmo de Regresión Logística es un caso especial del SGD, en el que el error que pretendemos minimizar es el opuesto del neperiano de la verosimilitud entre el tamaño de la muestra. N es el tamaño de la muestra.

$$E_{in}(w) = -\left(\frac{1}{N}\right)\ln(L(w)) = -\left(\frac{1}{N}\right)\ln\left(\prod_{n=1}^{N}(P(y_n|x_n))\right) = \frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{1}{\theta(y_n w^T x_n)}\right) =$$

$$= \frac{1}{N}\sum_{n=1}^{N}\ln\left(1+e^{-y_n w^T x_n}\right)$$

Podemos minimizarla con SGD porque esta función es estrictamente decreciente, para ello calcularemos su gradiente:

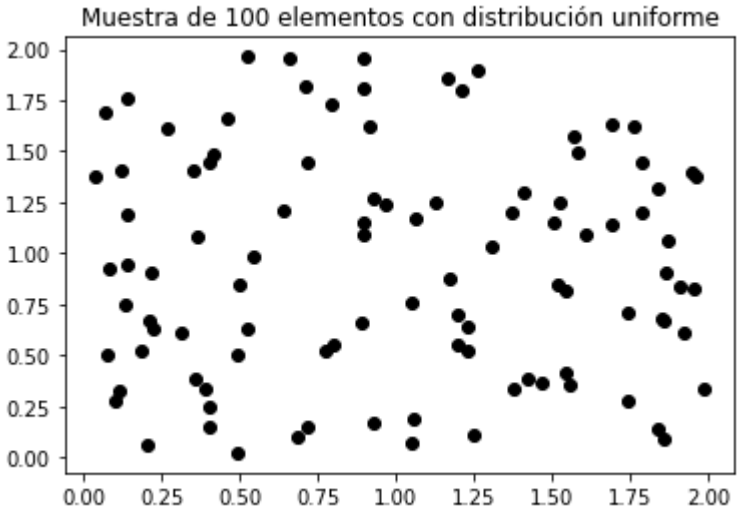
$$\nabla E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{y_n x_n}{1+e^{y_n w^T x_n}}\right)$$

Respecto a los datos de entrada de la función, esta recibirá como parámetros la función de error definida anteriormente (grad_fun), nuestra muestra de 100 puntos de dimensión 2 con valores comprendidos en el rango [0, 2] (x), las etiquetas asignadas a la muestra (y), la tasa de aprendizaje (lr = 0.01), el número máximo de iteraciones que será igual al número de puntos en la muestra (max_iter = 100), epsilon (0.01) y el tamaño del minibatch (1).

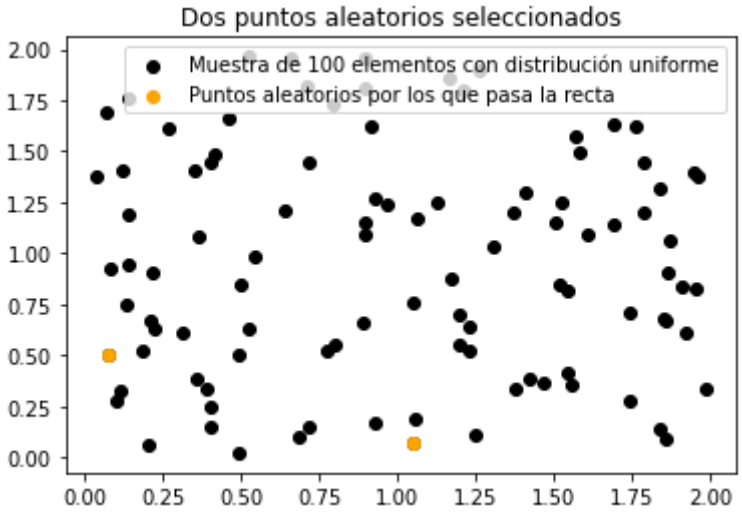
Como salida recibiremos un vector de pesos, el cual nos permitirá calcular la recta de separación, y el número de iteraciones consumidas.

b) 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).

Comenzamos generando un conjunto de 100 puntos de dimensión 2 comprendido en el rango [0, 2].



De nuestro conjunto seleccionamos 2 puntos al azar.



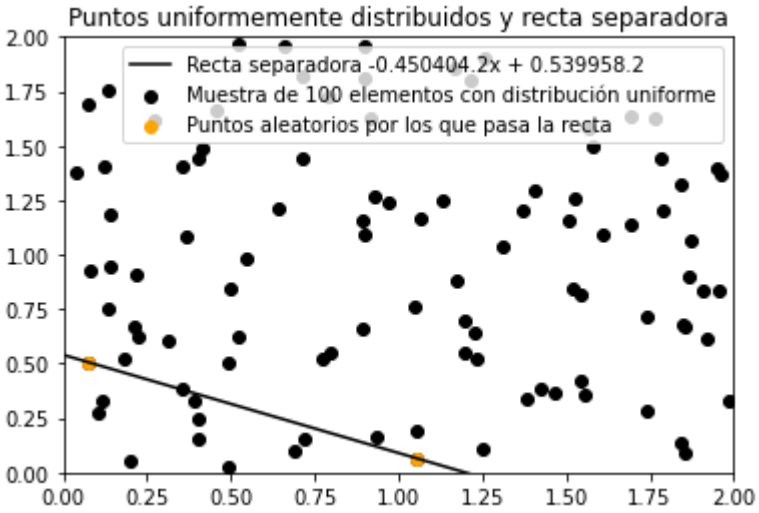
Seguidamente trazaremos una recta que pase por esos 2 puntos y que nos permitirá etiquetar los puntos. Para ello usamos la ecuación de la recta pendiente: Dados 2 puntos la pendiente será:

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$

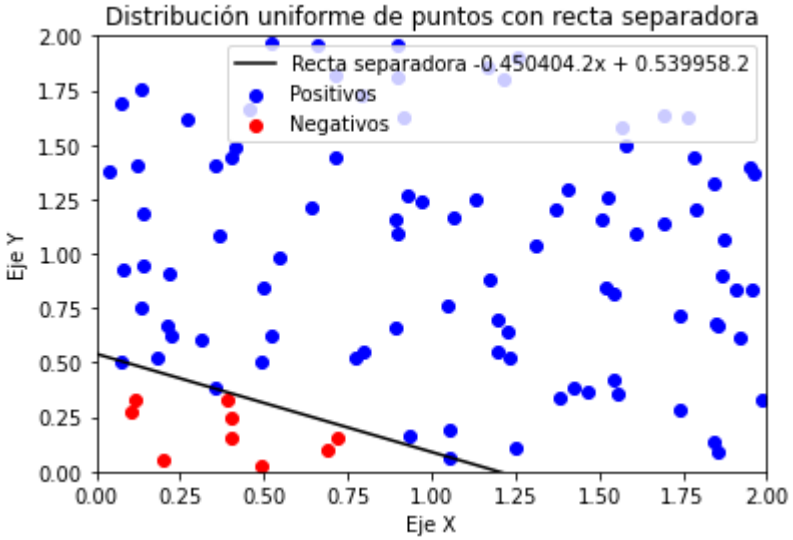
Y para el termino independiente despejamos:

$$y_2 - y_1 = a*(x_2 - x_1) \quad = \quad y = ax + (y_1 - ax_1)$$

De esta forma tenemos los coeficientes a y b de una recta y = ax + b



Finalmente etiquetamos nuestra muestra según la recta obtenida.



Aplicaremos ahora nuestro algoritmo definido de SGD sobre la muestra obtenida, de esta forma obtendremos un vector de pesos (w) que nos permitirá calcular el error existente, utilzando la fórmula definida en el apartado a). De esta forma obtenemos un error igual a 0.642.

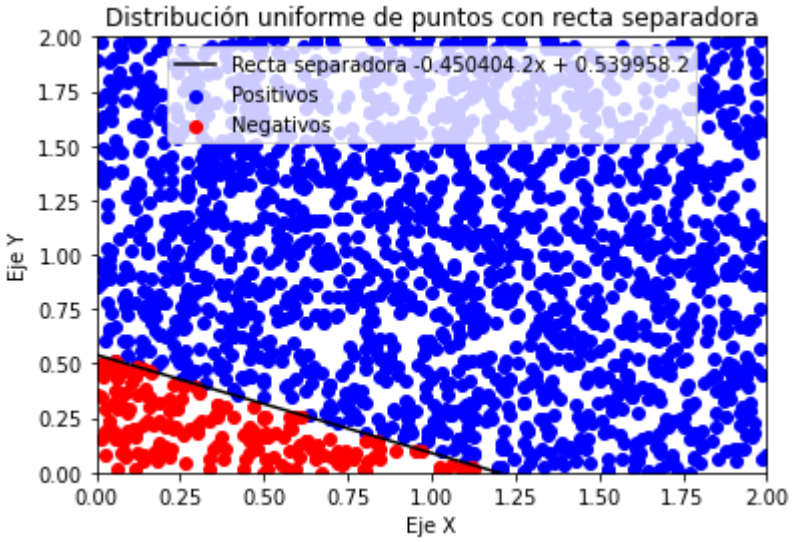
Generaremos a continuación otra muestra uniformemente distribuida en el mismo intervalo de valores pero esta vez con 2000 puntos con el objetivo de obtener E_{out}.

Utilizaremos la w que acabamos de obtener en nuestra muestra de 100 puntos para obtener las probabilidades de tener una etiqueta u otra, gracias a la función *prediccion_log()* que he definido y que realiza lo siguiente:

$$\theta(w^t x) = \frac{e^{w^T x}}{1+e^{w^T x}}$$

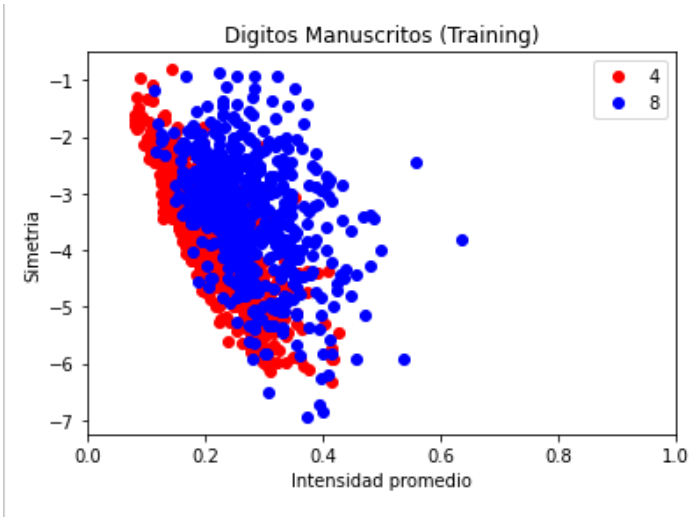
A partir de la función anterior asignaremos las etiquetas +1 ó -1 según la probabilidad de que un dato sea mayor o menor-igual a una frontera de 0.5 respectivamente.

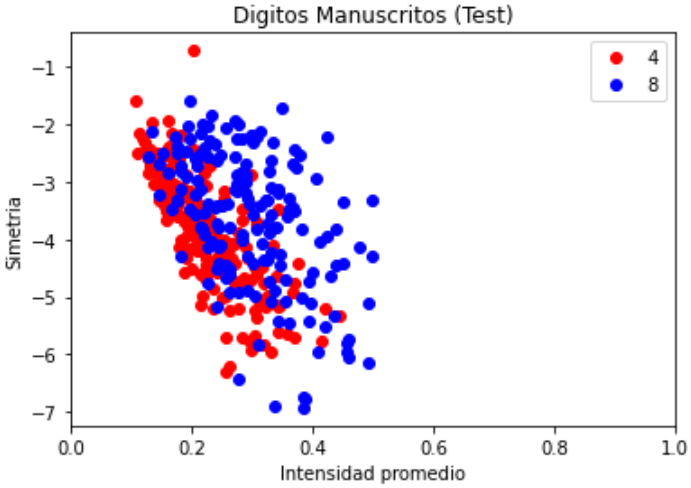
De esta forma obtenemos un $E_{out} = 0.122$



3. Bonus

Comenzamos presentado los datos de entrada, tanto los datos de training como los test leídos.





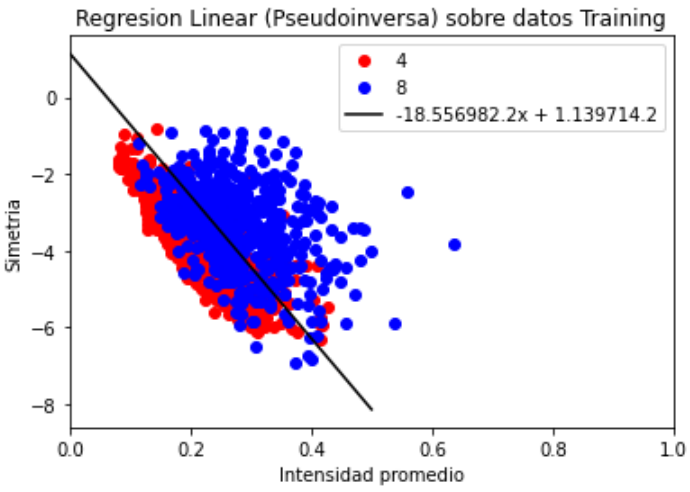
3.1. Regresión Lineal.

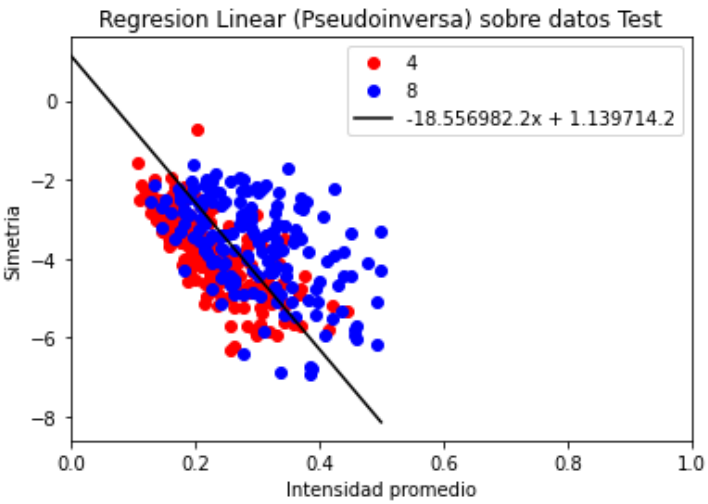
Clasifico los datos mediante una regresión lineal, para ello he utilizado la pseudoinversa cuya fórmula es $w = ((X^T X)^{-1} X^T) * y$

Con este modelo obtengo los siguientes errores de entrada y salida:

E_{in} = 0.22780569514237856

E_{out} = 0.25136612021857924

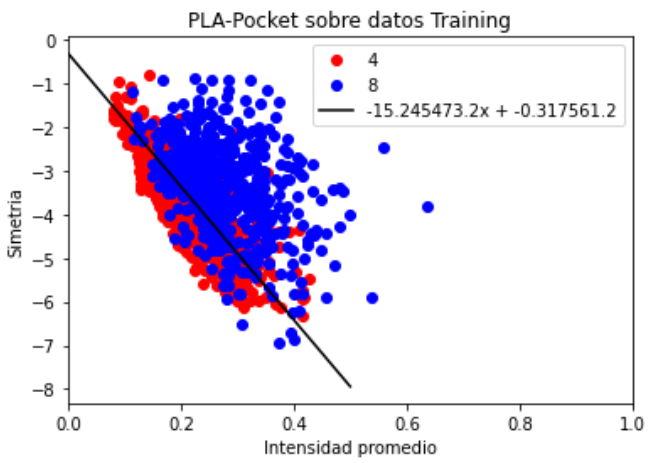


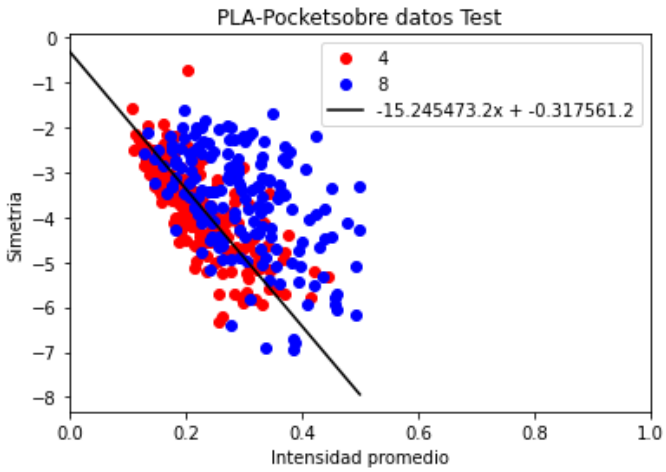


3.2. PLA-Pocket.

Este algoritmo es una mejora del algoritmo implementado en el ejercicio 2 de perceptrón.

PLA-Pocket almacena en cada iteración la mejor solución encontrada de forma que si llegamos al máximo de iteraciones se devuelva el mejor resultado de todos los explorados.
Inicializamos el vector de pesos respecto al "w" obtenido en el apartado anterior con la regresión lineal. En nuestro algoritmo actualizaremos este valor de "w" cuando la etiqueta de algún valor del conjunto de datos sea distinto a la etiqueta real del dato.
Como condición de paradas establecemos que ||w_anterior – w_actual|| no sea menor que el epsilon establecido (0.01) o que no se supere el número máximo de iteraciones.





Con este modelo obtengo los siguientes errores de entrada y salida:

E_{in} = 0.24623115577889448

E_{out} = 0.3224043715846995

No he conseguido mejorar los resultados respecto a los de la pseudoinversa. Aún así los errores se siguen manteniendo pequeños como en el caso anterior.

3.3. Cotas.

Para el calculo de las cotas he utilizado la siguiente fórmula:

$$\text{cota} = err + \sqrt{\frac{8}{tamaño}\ln\left(\frac{4\left((2\cdot tamaño)^{dimension}+1\right)}{tolerancia}\right)}$$

De esta forma obtenemos que:

Cota de E_{in} = 0.677167661249598

Cota de E_{out} = 1.0492414927034732

Observamos que la cota de E_{out} es mayor, esto se puede deber a que existe una menor diversidad entre los puntos que forman el conjunto de training respecto a los que forman el de test.