

Trabajo 2: Programación

Patricia Córdoba Hidalgo

Índice

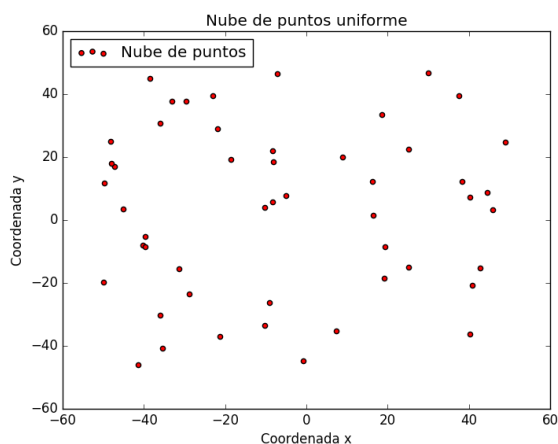
1. EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO	1
1.1. Apartado 1	1
1.2. Apartado 2	1
2. MODELOS LINEALES	4
2.1. Algoritmo Perceptron	4
2.2. Regresión Logística	5
3. BONUS	6

1. EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO

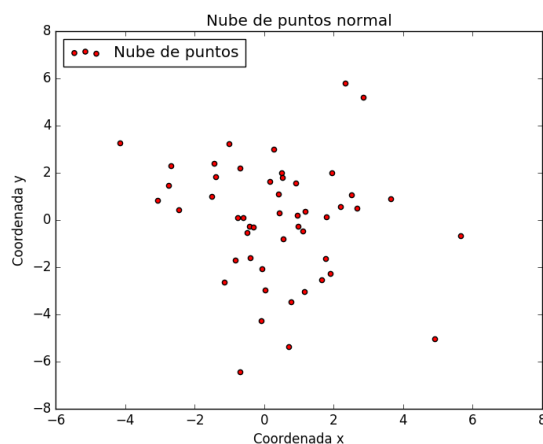
Este ejercicio está implementado en el fichero `ejercicio1.py`.

1.1. Apartado 1

Se generaron las dos siguientes nubes de puntos diferentes a partir de una muestra de tamaño 50.



(a) Nube generada uniformemente



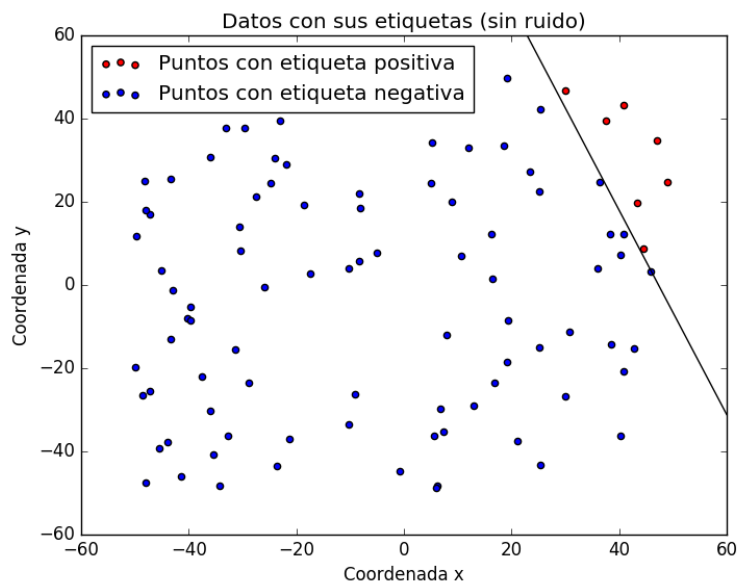
(b) Nube generada con una distribución Gaussiana

La primera se generó usando la función `simula_unif` y la segunda con la función `simula_gaus`, ambas nos las daban ya implementadas. Tras esto, se usa la función `muestra_datos` que representa los puntos de ambas nubes.

1.2. Apartado 2

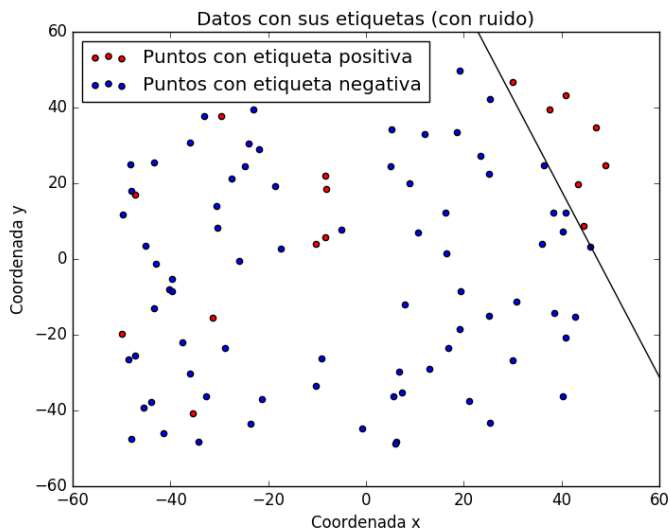
Etiquetamos una muestra de 100 puntos escogidos uniformemente en el cuadrado $[-50, 50] \times [-50, 50]$ según el signo de la función $f(x, y) = y + 2,45627963467703x - 116,24116219196964$, usando `simula_recta` para generar

la recta y `asigno_etiquetas` para etiquetar los datos. La gráfica de los puntos con su etiqueta correspondiente es:



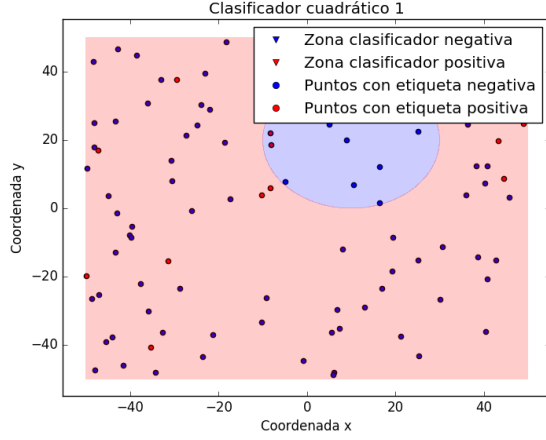
(c) Datos sin ruido

Modifiqué el 10 % (aproximadamente, ya que trunco a la baja) de las etiquetas de los datos con la función `asigna_etiquetas_ruido`. La gráfica con los datos nuevamente etiquetados es:

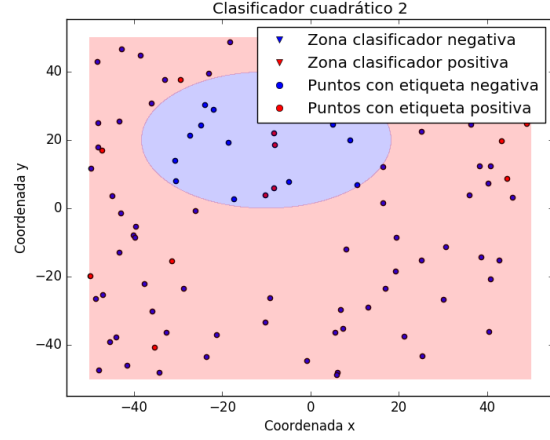


(d) Datos con ruido

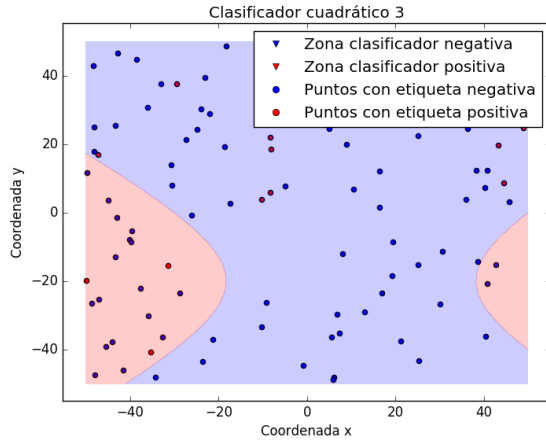
Tras esto, se usaron cuatro clasificadores diferentes sobre los datos etiquetados, ya que, al añadir ruido, pudiera ser que alguno de estos clasificadores de orden cuadrático cometiese un menor error que la recta que se usó para asignar las etiquetas originalmente, que ahora produce un error del 10 %. Las gráficas con esos clasificadores son:



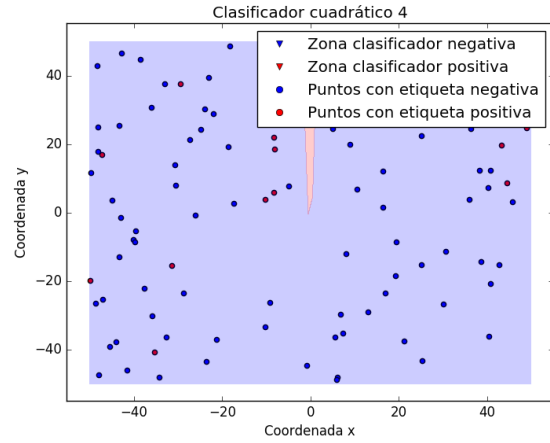
(e) $f1(x, y) = (x - 10)^2 + (y - 20)^2 - 400$



(f) $f2(x, y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$



(g) $f3(x, y) = 0,5(x - 10)^2 - (y + 20)^2 - 400$



(h) $f4(x, y) = y - 20x^2 - 5x + 3$

Estas graficas se representaron con la función **graficaf**. Usando como función de error el porcentaje de datos mal clasificados, implementada en la función **Err**, el error de cada clasificador es:

Función	Error
a) $(x - 10)^2 + (y - 20)^2 - 400$	75 %
b) $0,5(x + 10)^2 + (y - 20)^2 - 400$	75 %
c) $0,5(x - 10)^2 - (y + 20)^2 - 400$	32 %
d) $y - 20x^2 - 5x + 3$	16 %

Por tanto todos estos clasificadores tienen mayor error que la recta original, luego son peores que la función lineal.

En el proceso de aprendizaje, al extraer datos con sus etiquetas siempre estamos expuestos a que las etiquetas tengan un error respecto a la función real que las etiquetó. Es por esto que a veces, aunque la función real que etiqueta es una función lineal, por el comportamiento de los datos de la muestra, podríamos encontrar una función más compleja con menor error muestral. Con esto podemos producir sobreajuste, ya que intentar buscar funciones complejas que minimicen el error en la muestra puede acarrear un aumento del error E_{out} , ya que las clases de funciones complejas tienen una mayor varianza y con muestras muy pequeñas el error E_{out} varía mucho del E_{in} .

En nuestro caso específico, podemos ver que las funciones más complejas elegidas tienen mayor E_{in} que la original, luego no hay duda de que ésta es la mejor función encontrada para clasificar esta muestra.

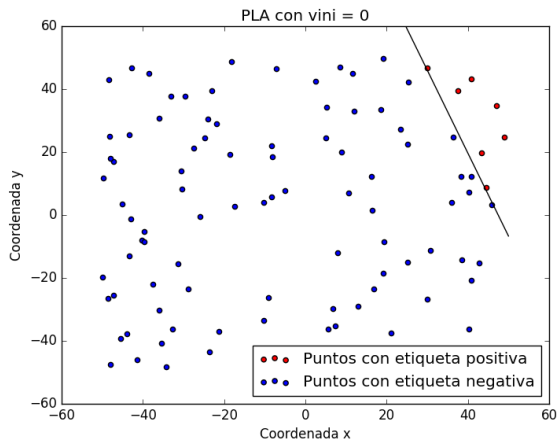
2. MODELOS LINEALES

Este ejercicio está implementado en el fichero `ejercicio2.py`.

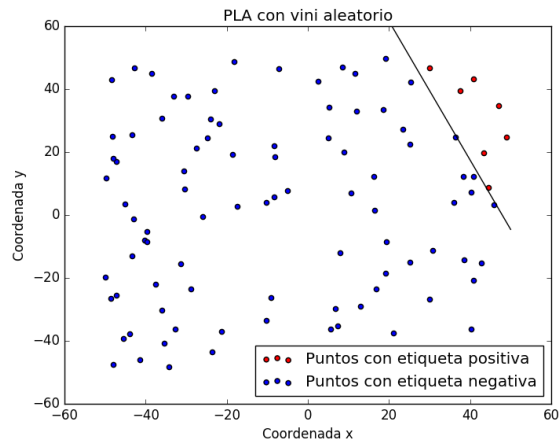
2.1. Algoritmo Perceptron

Programé la función `ajusta_PLA` para calcular el hiperplano solución al problema de clasificación. En mi caso, la entrada `datos` es una matriz donde cada fila es un ítem de la forma $x = [1 \text{ coordenada } X \text{ coordenada } Y]$, ya que meter las etiquetas en la matriz era redundante e innecesario al tener el vector de etiquetas `label`.

Ejecutamos el *Algoritmo del PLA* con la muestra del apartado 2 usando etiquetas sin ruido. Las gráficas obtenidas (en el caso del vector aleatorio con el último de los 10 usados) son:



(i) Vector inicial cero

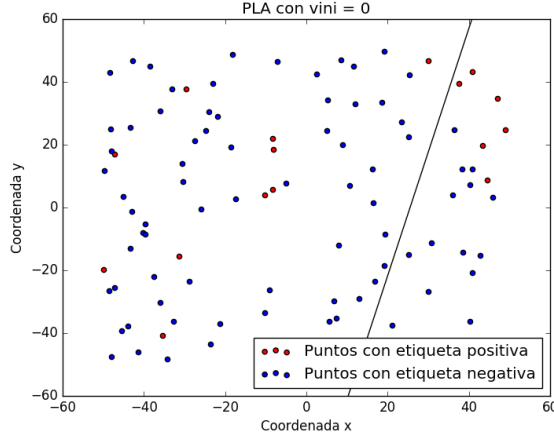


(j) Vector inicial aleatorio

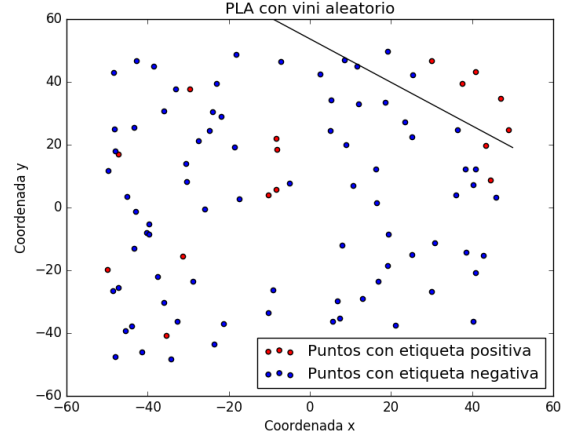
Usando como vector inicial el vector cero, obtenemos el mismo resultado en todas las ejecuciones, ya que el algoritmo no varía al no variar los datos de entrada, es determinista. El algoritmo converge en 444 iteraciones, obteniendo los pesos $w = [-2129 \ 44,85508745 \ 17,01070298]$.

Usando 10 vectores iniciales aleatorios, la media es de 285.3 iteraciones, y en la última iteración los pesos obtenidos son $w = [-412,45365318 \ 8,60651887 \ 3,90172468]$. Usando diferentes puntos iniciales, el número de iteraciones necesario para converger varía necesitando desde 17 iteraciones a 500 (o más) para converger.

Ejecutando el *Algoritmo del PLA* con la muestra del apartado 2 usando etiquetas con ruido obtenemos las siguientes gráficas:



(k) Vector inicial cero



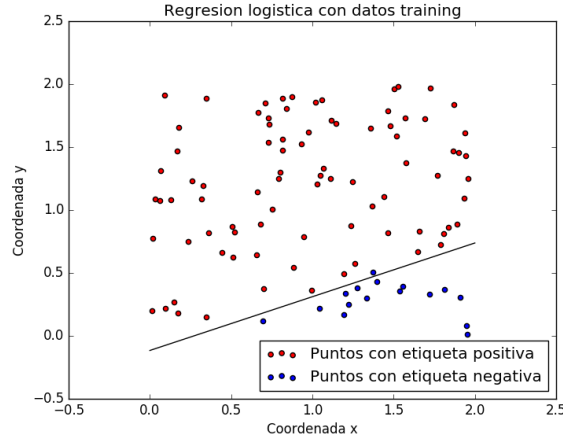
(l) Vector inicial aleatorio

En este caso los conjuntos no son separables, por lo que el *Algoritmo del PLA* no converge nunca. Por tanto, éste ciclará indefinidamente y devolverá el resultado de la última ejecución realizada, que no tiene por qué ser el resultado con menor error de todos los obtenidos en las diferentes iteraciones.

2.2. Regresión Logística

Usando la función `simula_unif` creamos una muestra de 100 puntos uniformemente distribuidos en el cuadrado $[0, 2] \times [0, 2]$ y con la función `simula_recta` seleccionamos la línea del plano que será la frontera de la función $f(x)$. Se asignan etiquetas a los puntos del cuadrado con `asigno_etiquetas`, dándole a los puntos por encima de la recta, que son aquellos con $f(x) = 1$ la etiqueta $y = 1$ y a los de debajo de la recta, que son aquellos con $f(x) = 0$, etiqueta $y = -1$.

Se implementó el algoritmo de *Regresión Logística* en la función `rl_sgd` obteniendo el siguiente resultado:

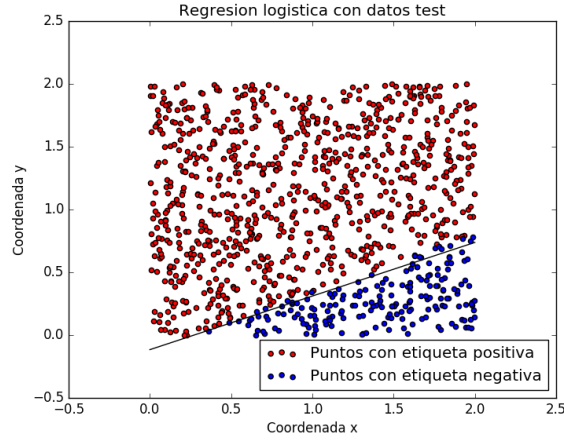


(m) Regresión Logística

La función obtenida con el algoritmo es $g(x) = \frac{e^{w^T x}}{1 + e^{w^T x}}$, donde $w = [0,85572891 \quad -3,1520009 \quad 7,38337662]$. El número de iteraciones necesarias para obtener este resultado es 335. El error $E_{in} = 0,0922856433393283$.

Se creó otra muestra de 1000 datos uniformemente distribuidos en el cuadrado con la función `simula_unif` cuyo error es $E_{out} = 0,0996398065527377$. Como podemos ver el error E_{out} difiere poco del error E_{in} , luego el ajuste es bueno.

El resultado obtenido con el conjunto de datos test es:



(n) Regresión Logística

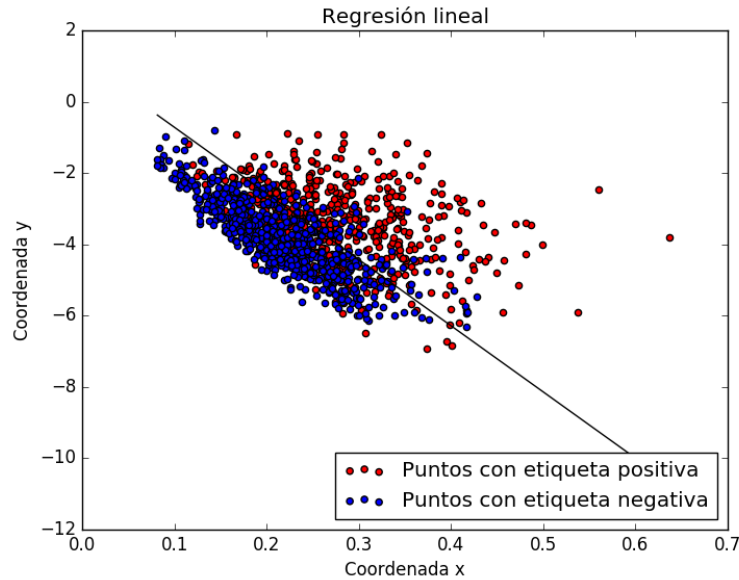
3. BONUS

Este ejercicio está implementado en el fichero `ejercicio3.py`.

El problema es clasificar un conjunto de dígitos en dos clases diferentes, los cuatros y los ochos. El modelo usado es el siguiente:

Los datos son de la forma $x_i = (1, x_1, x_2)$, donde x_1 es la intensidad promedio y x_2 es la simetría. Las etiquetas correspondientes son $y_i \in \{1, -1\}$. Inicialmente $y_i = 1$ si $x_i = 8$ y $y_i = -1$ si $x_i = 4$. Usando el algoritmo de *Regresión Lineal* y *PLA-Pocket* se obtiene w , que son los pesos de la función g que separa los datos de los dos conjuntos. Tenemos pues que $g(x, y) = w_0 + w_1x + w_2y$, con $w = (w_0, w_1, w_2)$. Si dado $x_i = (1, x_1, x_2)$ tenemos que $g(x_1, x_2) = w_0 + w_1x_1 + w_2x_2 > 0$, entonces el algoritmo predice una etiqueta es positiva ($y_i = 1$). En caso contrario, predice etiqueta negativa ($y_i = -1$).

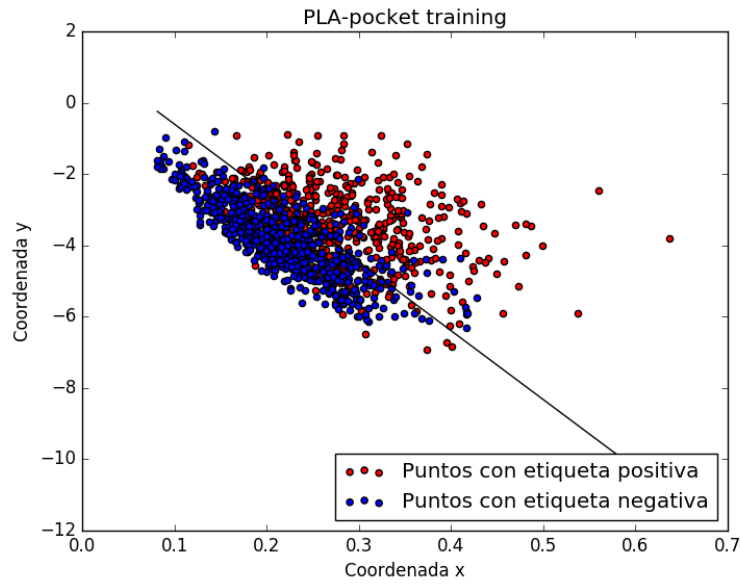
Con el algoritmo de *Regresión Lineal*, usando la matriz pseudoinversa, se consiguió $w = [-0,50676351 \quad 8,25119739 \quad 0,44464113]$. El resultado obtenido es:



(ñ) Regresión Lineal

El error E_{in} obtenido con esta recta es 0,22780569514237856.

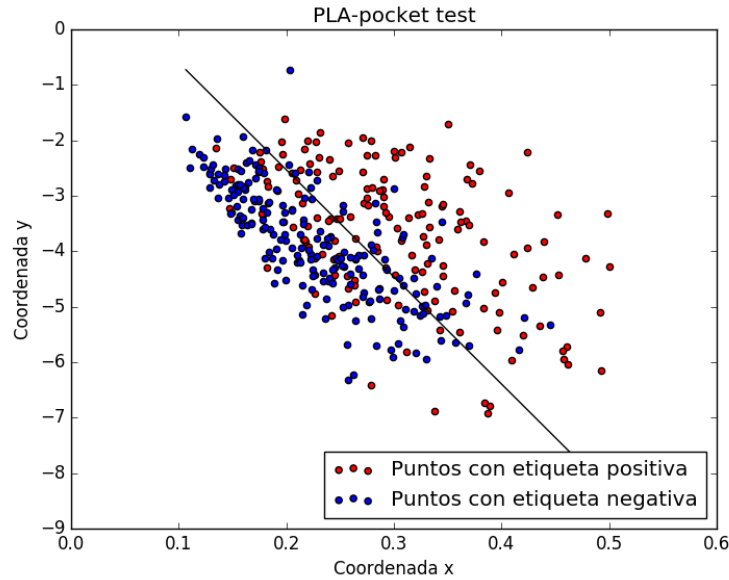
Usando este resultado como vector inicial del algoritmo de *PLA-Pocket* se consiguió $w = [-6,50676351 \quad 94,33278003 \quad 4,88432863]$. El resultado obtenido es:



(o) PLA-Pocket con datos de entrenamiento

El error E_{in} obtenido con esta recta es 0,22529313232830822, ligeramente menor que el obtenido con la recta de la regresión lineal.

Usando esta recta para clasificar los datos test, el resultado es:



(p) PLA-Pocket con datos test

El error E_{test} obtenido es 0,2540983606557377.

Para calcular la cota sobre E_{out} usamos la desigualdad de Hoeffding:

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{1}{2N} \log\left(\frac{2|H|}{\delta}\right)}$$

donde N es el número de datos de la muestra, $|H|$ es el número de funciones en la clase de funciones que escogemos para resolver el problema, en nuestro caso son las rectas, y δ es la tolerancia, es decir, esta cota se cumple con, al menos, probabilidad $1 - \delta$.

Como el conjunto de rectas en \mathbb{R}^2 es infinito, necesitamos hacer el truco de discretización para poder usarla. Como un ordenador moderno usa 64 bits para la representación de un escalar, para cada parámetro a determinar solo tenemos 2^{64} posibles valores, luego nosotros tenemos 2^{64*3} posibles valores en total, ya que una recta depende de tres parámetros (w tiene 3 componentes). Como para cada recta podemos etiquetar los puntos de dos formas diferentes (asignar las etiquetas positivas o negativas a un lado o a otro de la recta), el cardinal de H es $2^{64*3+1} = 2^{193}$.

En nuestro caso, usando $\delta = 0,05$, $N = 1194$, tenemos que:

$$E_{out}(h) \leq 0,22529313232830822 + \sqrt{\frac{1}{2 * 1194} \log\left(\frac{2 * 2^{193}}{0,05}\right)} = 0,4652211343$$

Para calcular la cota en función del E_{test} usamos la misma fórmula, teniendo en cuenta que ahora $N = 366$ y que la función que usamos está determinada, luego $|H| = 1$.

$$E_{out}(h) \leq 0,2540983606557377 + \sqrt{\frac{1}{2 * 366} \log\left(\frac{2}{0,05}\right)} = 0,3250874641$$

La cota con E_{test} es mejor, ya que la clase funciones la compone una sola función, no “infinitas”.