Aprendizaje Automático

Búsqueda Iterativa de Óptimos y Regresión Lineal

Práctica 2

Juan José Herrera Aranda - juanjoha@correo.ugr.es

4 de mayo de 2022

Índice

1. Introducción

Esta práctica se divide en tres partes. La primera parte se corresponde a la complejidad de la clase de funcione y el ruido. Etiquetaremos un conjunto de puntos y estudiaremos si funciones más complejas propician una mejor clasificación o no, además incluiremos algo de ruido para ver el efecto que tiene sobre la función que ha etiquetado los puntos y sobre las funciones que los clasifican. La segunda parte corresponde a modelos lineales, en particular, estudiaremos el perpceptrón haciendo incapié en la importancia de la separabilidad lineal de los datos y la regresión logística para la nube de puntos generada en la primera parte.

Por último, en la parte del Bonus, nos enfrentaremos a un problema más cercano al aprendizaje automático y más complejo que los anteriores. Emplearemos las técnicas aprendidas tanto en esta práctica como en las siguientes. Además se implentará el algoritmo PLA Pocket que es una extensión del PLA y compararemos los resultados de varios modelos entre sí. Se usarán luego los pesos obtenidos por el método de la pseudoinversa para ver si la solución que se obteiene es mejor que la solución obtenida usando como punto de partida el origen de coordenadas y por último se dará una cota para el error fuera de la muestra basándonos en el error dentro de la muestra y en el error en el conjunto de test.

2. Complejidad de H y el ruido.

2.1. Parte 1

En esta parte usaremos dos funciones proporcionadas por el profesorado. La primera de ellas se llama *simula_unif* y tiene tres argumentos *N,dim,rango* y calcula una lista de N vectores de dimensión *dim* donde cada lista contiene tantos números aleatorios uniformes como indique *dim* en el intervalo *rango* . La segunda, *simula_gauss* tiene los dos primeros parámetros iguales cambiando solo el tercero, en vez de *rango* es *sigma*. Lo que hace es calcular otra lista de longitud N de vectores de dimensión *dim* donde cada posición del vector contiene un número aleatorio extraído de una distribución Gaussiana de media cero y varianza dada, para cada dimensión, por la posición del vector *sigma*. Graficamos una nube de puntos (Fig. 1) para cada una de las funciones. Los números aleatorios (*pseudoaleatorios*) se han generado empleando como semilla *seed = 1*.

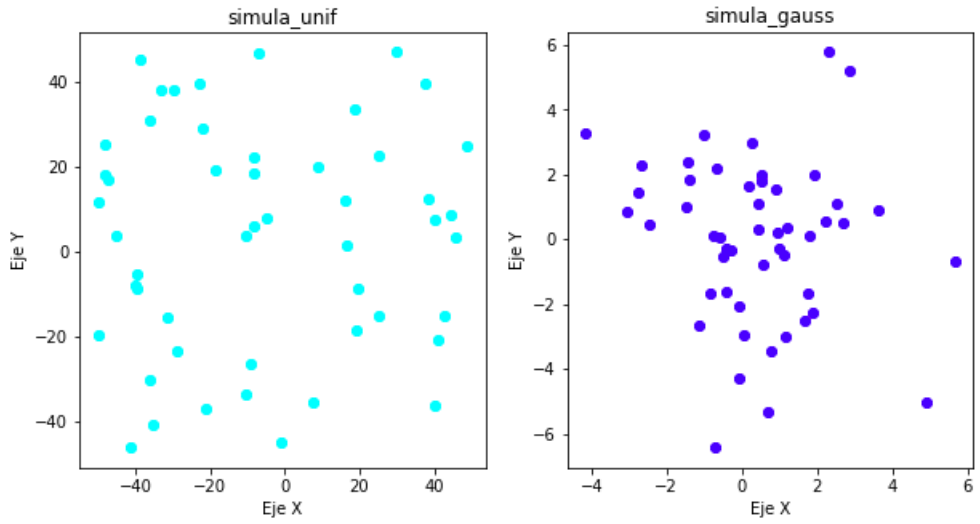
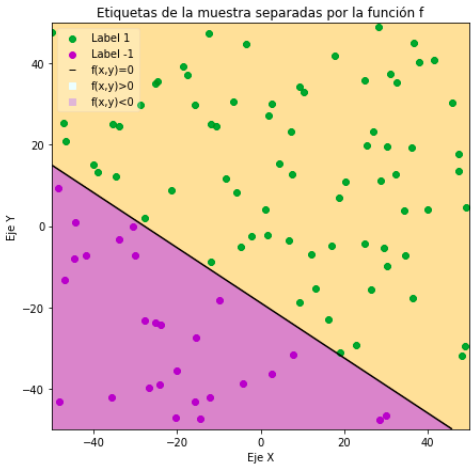


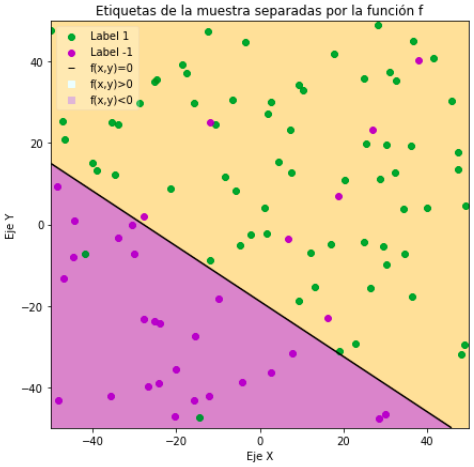
Figura 1: Muestras generadas por las funciones (*simula_unif*) con parámetros *(N=50, dim=2,rango=[-50,50])* y simula_gauss con parámetros *(N=50,dim=2,sigma=[5,7])*

2.2. Parte 2

A continuación vamos a valorar la influencia del ruido en la selección de la complejidad de la clase de funciones. Vamos a emplear la función *simula_unif* con parámetros (*100,2,[-50,50]*) para generar una muestra de puntos 2D. A dicha muestra le vamos a añadir una etiqueta (Fig. 2a) usando el signo de la función $f(x_1, x_2) = x_2 - ax_1 - b$ donde los parámetros $a = -0{,}677158$ y $b = -18{,}890228$ son la pendiente y la ordenada en el origen de una recta que corta al cuadrado $[-50, 50]x[-50, 50]$ la cual la simulamos previamente (de manera aleatoria). Tras ello, añadimos un 10 % de ruido en las etiquetas positivas y otro 10 % en las etiquetas negativas. (Fig. 2b).



(a) Puntos etiquetados según la función signo(f(x,y))



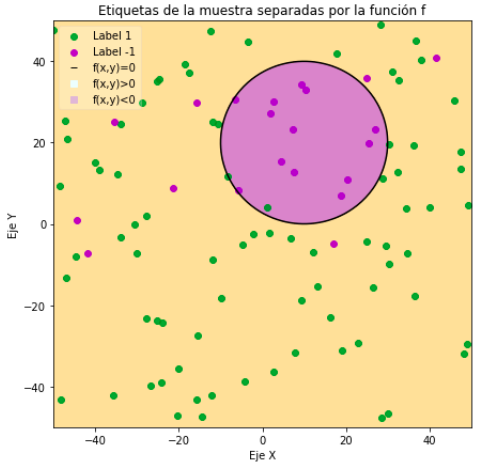
(b) Puntos etiquetados según la función signo(f(x,y)) añadiendo un 10 % de ruido

A continuación, vamos a dejar fijos los puntos de la muestra, pero los vamos a etiquetar con otras funciones y añadimos ruido como antes. Las funciones son las siguientes:

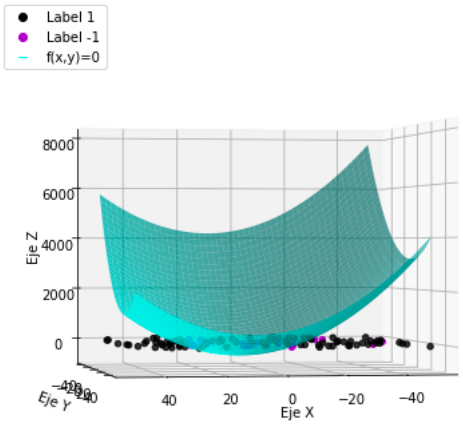
- $f_1(x_1, x_2) = (x_1 - 10)^2 + (x_2 - 20)^2 - 400$
- $f_2(x_1, x_2) = \frac{1}{2}(x_1 + 10)^2 + (x_2 - 20)^2 - 400$
- $f_3(x_1, x_2) = \frac{1}{2}(x_1 - 10)^2 1(x_2 + 20)^2 - 400$
- $f_4(x_1, x_2) = x_2 - 20x_1^2 - 5x_1 + 3$

Podemos observar en las imágenes de abajo la frontera de clasificación para cada una de las funciones representadas en 2D y 3D. En dos dimensiones tenemos la proyección sobre el plano Z=0 mientras que en tres dimensiones, se observa de manera más intuitiva, como la superficie formada por las gráficas de estas funciones separan a la muestra. Notamos también la presencia de un 10 % de ruido. En la tabla 1 presentamos los errores dentro de la muestra. Para medir el error hemos usado como función,

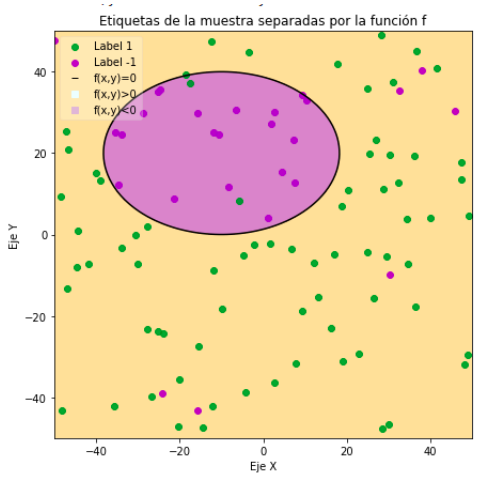
$$E_{in} = \frac{1}{N} \sum_{j=1}^{N} [\![sg(w^T x_j) \neq y_j]\!] \tag{1}$$



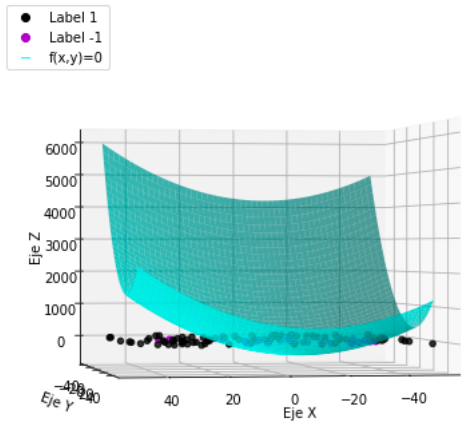
(a) Separación por f_1 en 2D



(b) Separación por f_1 en 3D



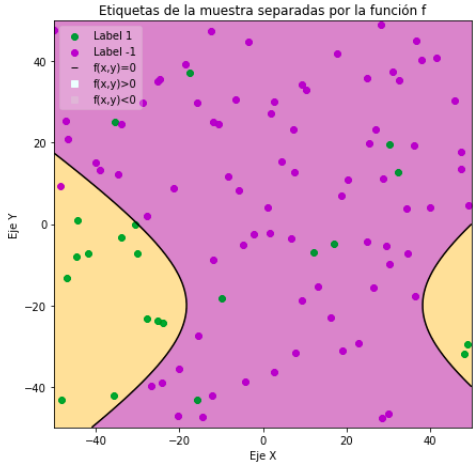
(c) Separación por f_2 en 2D



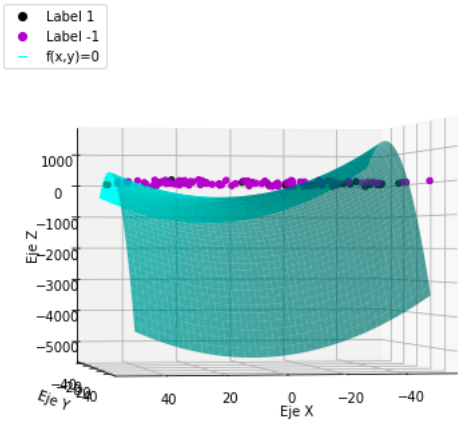
(d) Separación por f_2 en 3D

En aprendizaje automático se intenta aproximar a la función que realiza el etiquetado la cual es desconocida. En este ejercicio realizamos varios etiquetados con las funciones f_i, $i = 1, 2, 3, 4$, le vamos a meter un poco de ruido (10 %) y vamos a usarlas para clasificar su propio etiquetado y discutiremos cuál de ellas realiza una mejor clasificación.

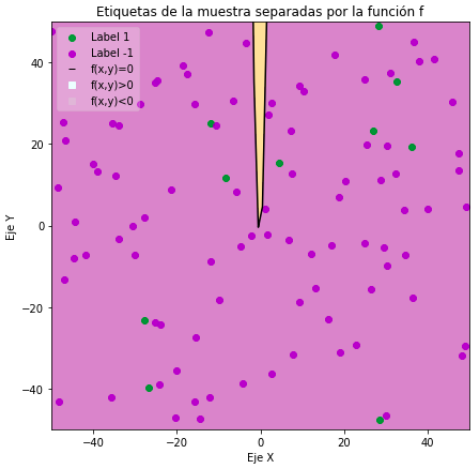
Claramente, si no llegamos a introducir ruido en la muestra tendríamos que $E^i_{in} = 0$ para $i = 1, 2, 3, 4$. Sin embargo, al introducir ruido, tenemos que el $E^i_{in} \neq 0$ va variando, y lo hace una cantidad de acuerdo al ruido introducido. La aleatoriedad es un fenómeno el cual no se puede modelar, y en este caso, el ruido tampoco. Nos hacemos la pregunta de que fijada una $i \in \{1, 2, 3, 4\}$ arbitraria, ¿Podemos encontrar una función g tal que $E^g_{in} \leq E^i_{in}$?, es decir, que tenga un error dentro de la muestra inferior al de la función objetivo. La respuesta es que sí, podemos crear una función g polinomial con el grado lo suficientemente alto como para que ajuste a la perfección la muestra con el ruido. Sin embargo, eso genera un gran inconveniente, y es que la función está también ajustando el ruido, es decir, la función g está formada de acuerdo a las peculiaridades de la muestra con el ruido *in situ*. Por lo que si consideramos un conjunto de test u otra muestra vamos a tener un E_{out} bastante elevado, y se pretende justamente lo contrario. Esto se debe a que g pretende aproximarse a f^i, y para ello está usando un orden polinomial mucho mayor que el de f^i, ajustándose dentro de la muestra mejor que la propia función objetivo, pero alejándose bastante de ésta.



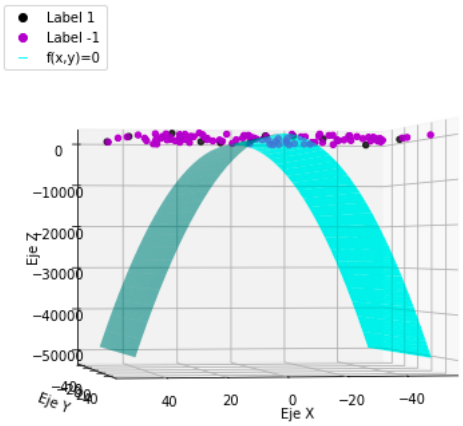
(a) Separación por f_3 en 2D



(b) Separación por f_3 en 3D



(c) Separación por f_4 en 2D



(d) Separación por f_4 en 3D

Tabla 1: E_{in} para cada función

Función	**Error**
f_1	0.09
f_2	0.09
f_3	0.09
f_4	0.1

Ahora cambiamos de pregunta, ¿Es posible encontrar una función g del mismo orden que f_i tal que $E^g_{in} \leq E^i_{in}$? La respuesta es que no. El objetivo de g es aproximarse todo lo posible a f^i que es la función objetivo y la que produce el etiquetado, por lo que g, con suerte, será como mucho f. Si no tuvieramos ruido, obviamente sería imposible obtener un error inferior a 0, que es el que obtiene f^i, ya que por la definición de error, nunca puede ser negativo. Con ruido, la intuición engaña, se puede pensar que sí que sería posible, ya que en este caso $E^i_{in} > 0$ y podríamos encontrar una función que ajuste mejor, sin embargo, es falso. Ya que como antes se ha dicho, el ruido es un fenómeno aleatorio el cual no se puede modelar. Ahora nuestra función objetivo no sería f^i, sino $f^i + \epsilon$ donde ϵ es el ruido aleatorio. No se puede aproximar g de manera determinística ateniéndose a un factor aleatorio, por lo que va a resultar imposible aproximarla a la función objetivo. Y no solo eso, la función que etiqueta es f^i y es la mejor función la cual

se puede usar como clasificador, entonces si añadimos un poco de ruido, el error que tengamos va a ser el mejor error que podamos encontrar, por lo que en ninguno de los casos obtendríamos una función del mismo orden que f_i que aproximase mejor dentro de la muestra.

En la tabla 1 vemos que f_4 tiene un error mayor que el de las otras, pero eso no es un indicio para decir que clasifica mejor o peor. Ya que por una parte, cada una genera un etiquetado y clasifica con la propia función objetivo. Además, el ruido introducido es aleatorio con probabilidad 10 % y puede generar un error cercano a 0.1 en función en como esté la frontera. Concluimos diciendo que ninguna de esas funciones clasifica mejor que otras, pues en todos los casos, se ha empleado la misma función para clasificar que para etiquetar.

Ahora, vamos a dar un giro de tuerca a lo anterior. Tenemos un total de 5 funciones $\{f_0 = f, f_1, f_2, f_3, f_4\}$. Para cada $i \in \{0, 1, 2, 3, 4\}$ vamos a etiquetar la muestra con usando f_i y vamos a clasificar usando el resto de las funciones. Podemos observar en la tabla, la métrica resultante de dividir el número de instancias mal clasificadas por el total de instancias. La diagonal coincide con la tabla 1 ya que se ha etiquetado y clasificado con la misma función.

Lo más destacable de la tabla es que no hay ninguna función que clasifique mejor que la función con la cual se ha etiquetado y se puede ver de manera visual fijándose en los valores de la diagonal, que son los menores valores existentes en las columnas. Al clasificar con la misma función que se etiqueta, los resultados serán los mejores a los que podamos aspirar teniendo en cuenta siempre el sesgo del ruido.

	f0	**f1**	**f2**	**f3**	**f4**
f0	0.09	0.42	0.5	0.83	0.67
f1	0.4	0.09	0.23	0.64	0.8
f2	0.48	0.21	0.09	0.58	0.76
f3	0.75	0.66	0.6	0.09	0.21
f4	0.68	0.79	0.73	0.22	0.1

Tabla 2: Columnas: Función que realiza etiquetado. Filas: Función que realiza la clasificación. Una entrada ij, corresponde a la precisión obtenida de realizar la clasificación usando la función f_i sobre la muestra etiquetada según la función f_j

A modo de conclusión, comentamos que no necesariamente, funciones más complejas son mejores clasificadores. Sí es bien que pueden representar un error dentro de la muestra que sea prácticamente cero, pero va en detrimento de la generalización. También comentar que es imposible pretender aproximarse a la función objetivo y superar el porcentaje de error del 10 % que se introdujo a modo de ruido.

3. Modelos Lineales

3.1. Algoritmo Perceptrón (PLA)

Se va a implementar el algoritmo del Perceptrón (PLA) para calcular el hiperplano solución a un problema de clasificación binaria. La entrada de datos es una matriz donde cada ítem está representado en una fila de la matriz. Las etiquetas tienen como valores $\{1, -1\}$. Vamos a emplear primeramente los datos del apartado 2a del ejercicio anterior, los cuales son linealmente separables y después, vamos a hacer lo mismo pero empleando los del apartado 2b, que no son linealmente separables y vamos a estudiar qué es lo que ocurre. Los detalles sobre el algorito junto con su implementación se encuentran en el código.

De manera adicional a la práctica, vamos a comentar la relación que existe entre el gradiente descendente y el PLA. La función de error a minimizar sería la de la ecuación 1, lo que ocurre es que no es continua y por tanto, tampoco derivable. Entonces usamos como alternativa la función,

$$error(w^T x_n, y_n) = max(0, -y_n w^T x_n). \tag{2}$$

En consecuencia, el gradiente del error en un punto (x_n, y_n) es 0 si $sg(w^T x_n) = y_n$ y $-y_n x_n$ si $sg(w^T x_n) \neq y_n$. EL PLA lo que hace es mirar uno por uno los datos (x_n, y_n) y comprueba si están bien clasificados, es decir, si $sg(w^T x_n) = y_n$. En caso afirmativo, el vector de pesos no sufre ningún cambio, pero en caso negativo sufre la actualización,

$$w_{update} = w_{current} + x_n y_n, \tag{3}$$

esto es, se actualiza en la dirección negativa del gradiente con una tasa de aprendizaje de $\mu = 1$.

El PLA suele acabar o bien cuando especificamos un número máximo de iteraciones o bien cuando los pesos no se actualizan durante una vuelta completa en el conjunto de entrenamiento (lo cual significa que todos están bien clasificados). Si los datos son linealmente separables, sabemos que vamos a obtener la solución óptima en tiempo finito, por lo que podemos especificar el primer criterio. En cambio, si los datos no son linealmente separables, el algoritmo va a ciclar y el segundo criterio es el que se debiera usar.

Primeramente, vamos a emplear los datos de la figura 2a. Hemos realizado 11 ejecuciones del PLA variando los pesos iniciales. En la primera ejecución estos son cero y las siguientes, números aleatorios en el rango $[0, 1]$. Al ser los datos linealmente separables vemos que en los resultados proporcionados por la tabla 3 el error en todos los casos es cero. El que menos tarda en converger lo hace en 23 iteraciones y el que más en 286. Por otra parte, vemos que los hiperplanos obtenidos son en cada caso diferentes aunque los hay algunos muy parecidos entre sí. Esto es porque la solución no es única y podemos tener el hiperplano desplazado un poco hacia una dirección u otra y seguir teniendo error cero, esto lo podemos ver en la figura 5 donde lo mostramos de manera visual.

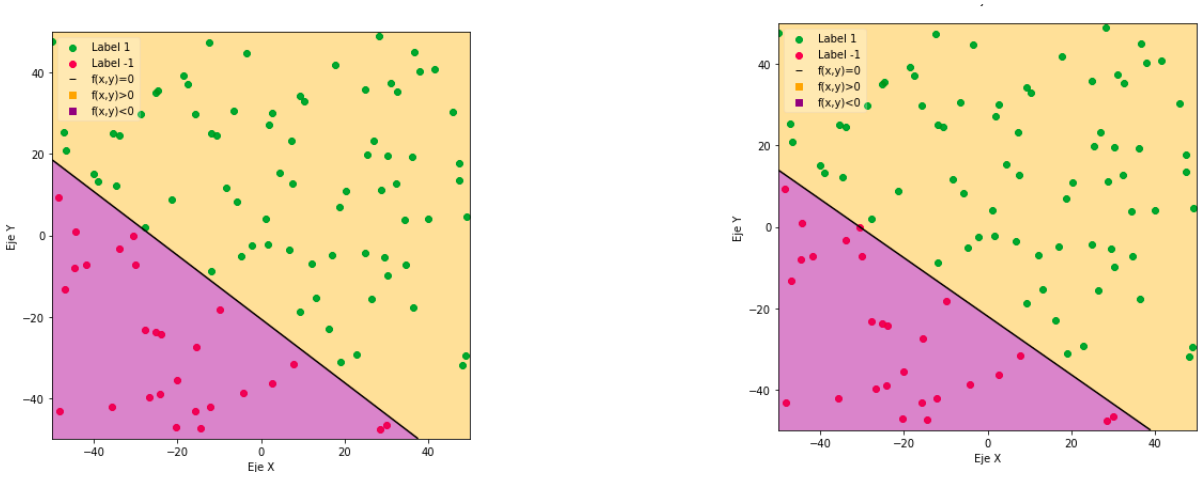


Figura 5: Dos hiperplanos que separan a los datos obtenido de dos de las 11 ejecuciones realizadas. Se observa que ambos son un poco distintos pero separan con error cero. Esto pone de manifiesto la no unicidad de solución para este problema en concreto.

Tabla 3: Resultados PLA usando los datos del ejercicio 1.2a

	vini	**fini**	**it**	**error**
0	[0,0,0]	[661 , 23.202 , 32.391]	75	0.0
1	[0.006, 0.613, 0.0311]	[1124.006 ,38.938 , 59.415]	260	0.0
2	[0.062, 0.257 ,0.591]	[1163.062 , 40.012, 61.521]	286	0.0
3	[0.088, 0.861, 0.072]	[558.088, 20.149, 30.224]	62	0.0
4	[0.328, 0.8107 0.933]	[815.328,2 29.19, 43.786	118	0.0
5	[0.246, 0.554, 0.560]	[464.246, 15.279,7 23.80]	43	0.0
6	[0.477, 0.449, 0.935]	[654.477, 21.406, 29.8]	70	0.0
7	[0.390, 0.046, 0.255]	[1111.39, 43.406, 61.929]	255	0.0
8	[0.432, 0.294, 0.541]	[653.43,3 23.09 32.0113	75	0.0
9	[0.760, 0.906, 0.771]	[654.76, 23.523 31.707,]	70	0.0
10	[0.564, 0.171, 0.425]	[1122.564, 38.641, 59.609,]	267	0.0

Ahora vamos a repetir el procedimiento anterior pero usando los datos del apartado 2b del ejercicio 1 (Fig. 2b). Estos datos tienen la peculiaridad de no ser separables, por lo que el algoritmo va a ciclar siempre a no ser que se le indique un máximo de iteraciones.

El PLA actualiza los pesos ejemplo tras ejemplo provocando que ejemplos que estén bien clasificados en un momento dado, acaben mal clasificados en otro momento ya que no se tienen en cuenta a los anteriores ejemplos. Una consecuencia de esto es que el PLA no converge conforme el número de iteraciones aumenta y ocurre que para una iteración anterior puede ocurrir que se obtengan mejores resultados que en iteraciones posteriores. Esto es más fácil de entender si pensamos que para clasificar correctamente un ejemplo en una iteración actualizamos los pesos provocando que otro ejemplo bien clasificado pase a estar mal clasificado y en la iteración siguiente, para clasificar bien otro ejemplo, hacemos que en la actualización dos o más ejemplos que están bien clasificados pasen a estar mal clasificados. De este modo, en la primera actualización para clasificar bien uno, estropeamos otro, y en la segunda, para clasificar bien uno, estropeamos dos. En la tabla 4 podemos observar este comportamiento ya que en 7000 iteraciones obtenemos un error menor que si por ejemplo lo ejecutamos durante 10000.

Tabla 4: Resultados PLA con vector de inicialización $[0, 0, 0]$ y distintas iteraciones para datos no separables

	vini	**fini**	**it**	**error**
0	[0 0 0]	[398 , 7.204 , 37.423]	500	0.2
1	[0 0 0]	[401, 10.749, 43]	1000	0.17
2	[0 0 0]	[389, 3.323, 24.389]	5000	0.18
3	[0 0 0]	[399, 12.216, 21.764]	7000	0.12
4	[0 0 0]	[381, 15.952, 45.996]	10000	0.16

Cabe decir que el número de iteraciones óptimas depende de los pesos iniciales, en este caso, el mejor valor para el número de iteraciones es 7000 entre todos los valores que hemos usado. Pero eso no quiere decir, que si empleásemos otros pesos iniciales, este valor fuera el mejor, ya que por ejemplo, podemos ver en la tabla 3 que para distintos puntos de partida, se llega a una solución en distintas iteraciones. Por lo que para cada punto de partida, habría que estudiar dentro de un rango de valores, cuál es el valor óptimo del número de iteraciones a usar. Para tener las mismas condiciones en todas las ejecuciones, nos apoyamos en esta premisa para justificar que el número de iteraciones van a ser 1000 para las ejecuciones restantes empleando distintos puntos iniciales.

En la tabla 5 residen los resultados del ejecutar el PLA con pesos aleatorios en el rango $[0, 1]$ usando la muestra del ejercicio 1.2b. Como se puede apreciar, los errores ya no son cero debido a la no separabilidad lineal de los datos. Exceptuando la ejecución 0 y 3 que son las que presentan un error algo más elevado, el resto de ejecuciones presentan unos resultados similares.

	vini	**fini**	**error**
0	[0.151, 0.635, 0.847]	[396.151, 9.207, 47.665]	0.2
1	[0.821, 0.628, 0.956]	[403.821, 20.837, 25.197]	0.13
2	[0.589, 0.197, 0.429]	[395.589, 11.339, 17.553]	0.11
3	[0.336, 0.991, 0.380]	[414.336, 4.435, 36.847]	0.21
4	[0.992, 0.518, 0.172]	[392.992, 8.688, 23.745]	0.14
5	[0.074, 0.370, 0.123]	[399.074, 22.691, 30.406]	0.14
6	[0.634, 0.413, 0.99]	[406.634, 19.007, 33.427]	0.11
7	[0.929, 0.149, 0.394]	[403.929, 20.881, 25.360]	0.13
8	[0.461, 0.56, 0.78]	[393.461, 14.969, 25.632]	0.12
9	[0.487, 0.419, 0.261]	[398.487, 11.298, 17.367]	0.11

Tabla 5: Resultados del PLA en un conjunto de datos linealmente no separable usando inicializaciones aleatorias en $[0, 1]$ para los pesos iniciales.

Resumiendo lo anterior decimos que en función de si los datos son linealmente separables o no, o bien vamos a poder ejecutar el PLA en tiempo finito para obtener la solución óptima o bien el algoritmo ciclará y en su defecto, habrá que estudiar el número de iteraciones óptimas a realizar. En la práctica, la propiedad que un conjunto de datos sea o no linealmente separable no es algo tan intuitivo como este ejemplo y se tendrá que profundiza más en la naturaleza de los datos para obtener más información acerca de ellos.

3.2. Regresión Logística

Vamos a usar al función *simula_unif* con parámetros $N = 100$, $d = 2$, y $rango = [0, 2]$ para simular un conjunto de datos $\mathcal{D} \subset \mathcal{X} = [0, 2]x[0, 2]$ con probabilidad uniforme de tomar $x \in \mathcal{X}$. Tomamos dos puntos aleatorios dentro de ese rango y simulamos una recta, en concreto la que tiene como pendiente $a = -1{,}072$ y como ordenada en el origen $b = 1{,}603$ y etiquetamos la muestra usando como frontera dicha recta (Fig. 6).

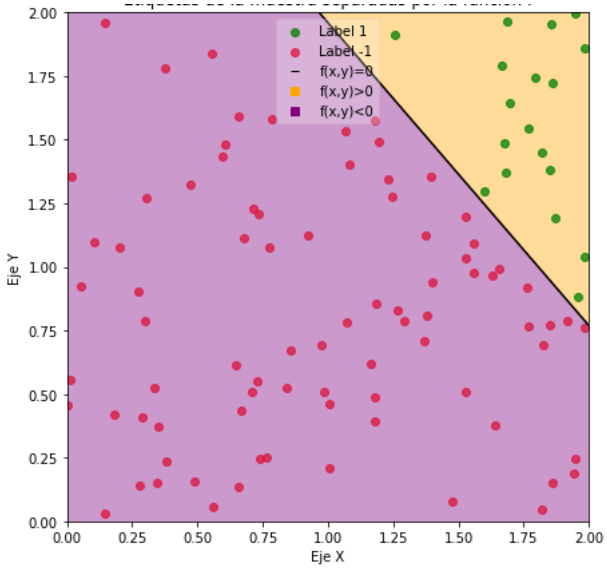


Figura 6: Nuevo conjunto de datos a usar en este subapartado

A continuación implementamos el algoritmo de la regresión logística usando SGD inicializando los pesos iniciales con el vector cero. El criterio de parada empleado es detener el algoritmo cuando $||w^{(t+1)} - w^{(t)}|| < 0{,}01$ donde $w^{(t)}$ representa el vector de pesos al final de la época t. Y en cada época se reordenarán los ejemplos de entrenamiento de manera aleatoria.

El error ahora viene dado por la función

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

que tiene la peculiaridad de que aunque estén todos los puntos bien clasificados es difícil que se anule, ya que para eso el producto escalar de la exponencial tendría que diverger positivamente en todos los sumandos. La solución obtenida aplicando regresión logística es de la forma $g(x) = \sigma(w^T x)$ siendo σ la función sigmoidal. La función g asigna a un vector de característica la probabilidad de que la etiqueta sea $+1$. A la hora de clasificar asignamos la etiqueta $+1$ si la probabilidad es mayor o igual que $\frac{1}{2}$ o -1 en caso contrario.

Como ya sabemos, el gradiente descendente tiene para tunear tres hiperparámetros que son el tamaño del minilote, el learning rate, y el número de épocas. En este caso usaremos 5000 épocas que son más que suficientes para que el algoritmo pare usando el criterio de parada que acabamos de comentar. Para la elección del tamaño del lote y del learning rate, hemos realizado un proceso experimental que queda reflejado en al tabla 6. En ella se observa que los hiperparámetros que nos dan el error más bajo son $batch\_size = 1$ y $lr = 0{,}1$. Aunque a modo general, todas las ejecuciones presentan errores muy cercanos a cero, esto no quiere decir que el ajuste sea el mejor, ya que los datos son linealmente separables y no llegamos a obtener el hiperplano óptimo.

	batch	lr	it	error
0	1	0.01	501	0,124
1	1	0.05	1035	0,047
2	1	0.1	1435	0,03
3	2	0.01	372	0,184
4	2	0.05	758	0,072
5	2	0.1	1045	0,046
6	4	0.01	264	0,276
7	4	0.05	554	0,109
8	4	0.1	763	0,072
9	8	0.01	264	0,276
10	8	0.05	562	0,108
11	8	0.1	767	0,072
12	16	0.01	263	0,276
13	16	0.05	558	0,108
14	16	0.1	765	0,072
15	32	0.01	264	0,276
16	32	0.05	557	0,108
17	32	0.1	771	0,071

Tabla 6: Proceso experimental para la elección de los hiperparámetro de la regresión logística con SGD. La columna *it* se refiere al número de épocas que han sido necesarias para parar usando el criterio de parada

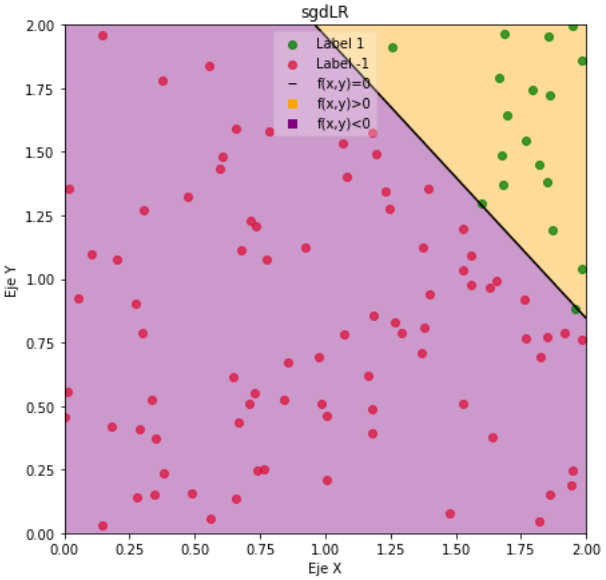


Figura 7: Hiperplano realizado por la regresión logística empleando SGD con $lr = 0{,}1$ y $batch\_size = 1$

Ahora nos creamos un conjunto de test con 2000 ejemplos en las mismas condiciones con las que nos creamos el anterior conjunto y clasificamos los puntos usando del conjunto de test usando los pesos obtenidos de aplicar regresión logística con SGD . Los resultados están presentes en la tabla 6 y la gráfica del hiperplano en la figura **??**, y como era de esperar, E_{out} es más elevado que E_{in}, aunque la diferencia radica en una centésima, por lo que podemos decir que están muy próximos entre sí y tenemos una buena generalización.

Batch size	1
Learning Rate	0,1
Ein	0,029
Eout	0,041

Tabla 7: Hiperparámetros usados en la ejecución de la regresión logística y resultados obtenidos

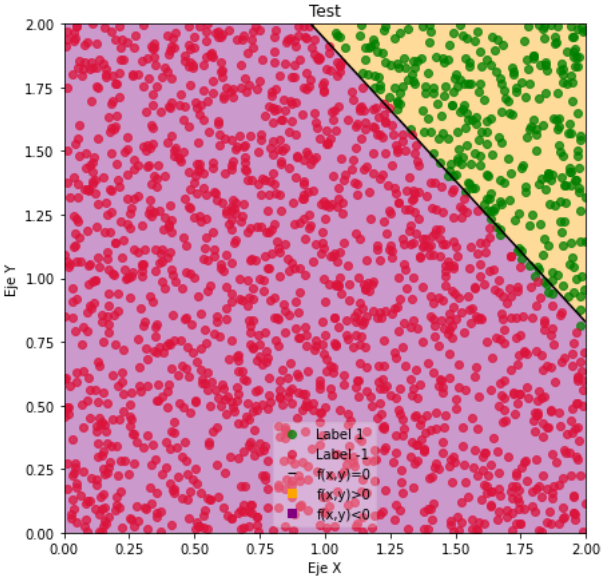


Figura 8: Hiperplano en el conjunto de test.

Ahora repetimos el experimento anterior unas 100 veces, esto implica, crear de nuevo una muestra de entrenamiento usando el mismo proceso que realizamos para crear la anterior, así como también un conjunto de test. Tras ello ejecutamos regresión logística con los hiperparámetros anteriormente elegidos para SGD y hacemos la media de los errores y de las épocas:

$$E_{in}^{mean} = 0{,}0308 \quad E_{out}^{mean} = 0{,}0401 \quad Epochs^{mean} = 1249{,}7$$

4. Bonus

Tenemos ahora un problema distinto y más práctico a los anteriores. Se nos presentan un conjunto de dígitos manuscritos los cuales pretendemos clasificar en dos clases que corresponden al dígito 4 y al 8. Las etiquetas son -1 y 1 respectivamente. Las características consideradas son la intensidad de gris promedio, la cual denotaremos como x_1 y la simetría respecto al eje vertical que la denotaremos como x_2. Tenemos dos conjuntos, uno que usaremos para entrenar distintos modelos, y otro conjunto que lo usaremos de test. El primero contiene 1194 ejemplos y el segundo 336. Cada elemento se representará como un vector $x = (1, x_1, x_2)$. El objetivo es aprender una función $g(x) = sg(w^T x)$, para un cierto vector de pesos.

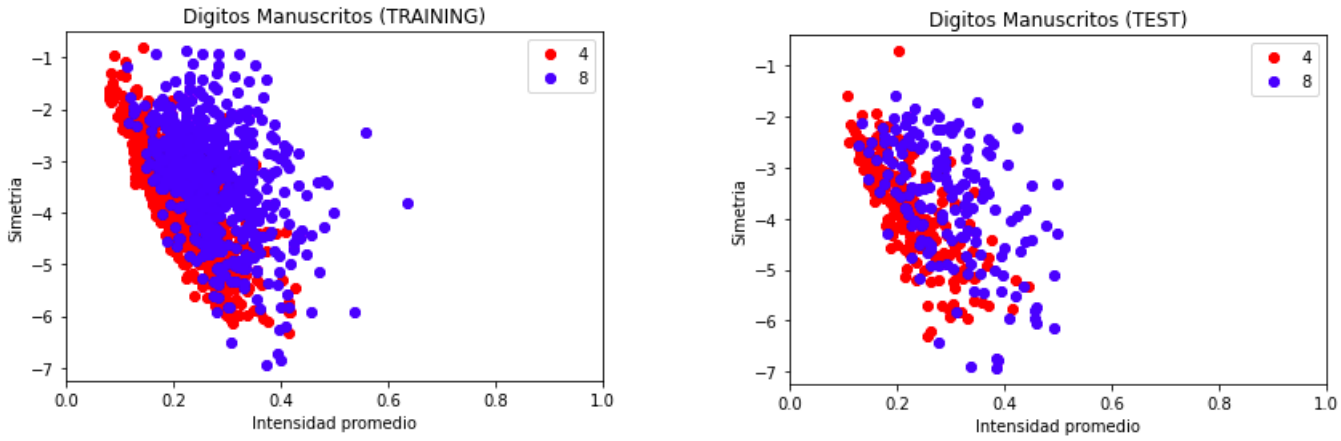


Figura 9: (Izquierda) Representación de los datos de entrenamiento. (Derecha) representación de los datos de test

Como primera aproximación aplicaremos regresión lineal usando el algoritmo de la pseudo-inversa visto en la práctica anterior, PLA, PLA Pocket y regresión logística. El algoritmo PLA Pocket no lo hemos usado anteriormente y es una extensión del PLA de tal manera que se va guardando en una variable los mejores pesos hasta el momento. Se suele usar cuando los datos no son linealmente separables y así evitar que el PLA devuelva una aproximación no lo bastante buena.

La tabla **??** muestra los resultados de las ejecuciones. En el pie de la figura se indican como se ha ejecutado cada algoritmo. Para escoger los hiperparámetros adecuados de la regresión logística se ha hecho un estudio experimental probando distintos valores para el learning rate y el tamaño del batch. Los resultados (apreciables en la tabla **??**) son muy similares entre sí, aunque los mejores son los proporcionados por $\mu = 0{,}05$ y $batch\_size = 8$ y son los que hemos seleccionado.

	batch	lr	error
0	1	0.1	0,467
1	2	0.1	0,466
2	4	0.1	0,464
3	8	0.1	0,466
4	16	0.1	0,572
5	1	0.01	0,464
6	2	0.01	0,471
7	4	0.01	0,467
8	8	0.01	0,468
9	16	0.01	0,467
10	1	0.05	0,465
11	2	0.05	0,521
12	4	0.05	0,522
13	8	0.05	0,466
14	16	0.05	0,463

Tabla 8: Proceso experimental para la elección de los hiperparámetros en regresión logśitica.

En la tabla de abajo podemos obsevar que el método que mejor se ajusta a los datos dentro de la muestra es la regresión logística ya que presenta una menor proporción de instancias mal clasificadas. En cuanto al método que mejor resultados presenta en test es el PLA Pocket, el cual obtiene el óptimo en la iteración 26. Como era de esperar, sus resultados son mejores que el método del PLA asecas. Por otra parte, el método de la pseudoinversa es famoso por presentar los resultados óptimos y sin embargo, no es el que mejores resultados presenta. Esto se debe a que en cada método se usan métricas de error distintas, por lo que no son comparables desde ese punto de vista. Para poder compararlos, se ha usado la métrica de la proporción de instancias mal clasificadas.

	E in	E test	E. Prec. in	E. Prec. test	Pesos
R. Linear Pseudoinversa	0,642	0,708	0,227	0,251	[-0.506, 8.251, 0.444]
PLA	0,231	0,248	0,231	0,248	[-10, 169.748, 10.045]
PLA Pocket	0,228	0,245	0,228	0,245	[-8, 138.572, 8.095]
RL	0,462	0,532	0,224	0,256	[-1.485, 28.815, 1.541]

Tabla 9: Resultados. El PLA y PLA Pocket se han ejecutado un máximo de 5000 iteraciones con vector de pesos iniciales $[0, 0, 0]$. La regresión logística también ha usado el vector origen de partida. Primera y segunda columna corresponden al error dentro de la muestra y al de test respectivamente. Notamos que cad método tiene su función de error asociada, de este modo, la regresión linear tendrá el error cuadrático medio, la regresión logística la entropía cruzada, y el PLA el número de instancia bien clasificadas partido por el total. Las siguientes dos columnas corresponden a la proporción de instancias mal clasificadas dentro de la muestra y en el conjunto de test respectivamente. Notamos que coinciden con los errores del PLA y PLA Pocket.

En los anteriores modelos hemos usado como vector de partida el origen de coordenadas en el espacio. Ahora usaremos los pesos obtenidos a través del método de la pseudoinversa. Gracias a la tabla de abajo, (10), podemos ver que los resultados no mejoran apenas a los anteriores. Se suelen emplear los pesos de la pseudoinversa para inicializar los pesos iniciales de otros modelos lineales ya que así se obtiene una buena solución en pocas iteraciones por estar cercanas. Sin embargo, en este caso los mejores pesos parecen ser el origen de coordenadas.

Tabla 10: ResultadosRL2

	E in	**E test**	**E Prec. in**	**E. Prec. Test**	**Pesos**
PLA	0,295	0,297	0,295	0,297	[-10.506, 172.652, 12.256]
PLA Pocket	0,225	0,254	0,225	0,254	[-6.506, 94.332, 4.884]
RL	0,557	0,572	0,227	0,251	[-0.506, 8.251, 0.444]

Sabemos que gracias a la desigualdad de Hoeffding y la dimensión VC, una cota para E_{out} es:

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{8}{N} \log \frac{4((2N)^{d_{VC}} + 1)}{\delta}}$$

donde N es el número de datos de la muestra, d_{VC} es la dimesión de VC de la clase de funciones considerada y δ es la tolerancia (la desigualdad se cumple con probablidad de al menos $1-\delta$).

Como estamos usando como clase de funciones la clase del perceptron 2D, tenemos que $d_{VC} = 3$. y como $N = 1194$ en el conjunto de entrenamiento, la cota que obtenemos para E_{out} basada en E_{in} es la siguiente:

$$E_{out}(h) \leq E_{in}(h) + \sqrt{\frac{8}{1194} \log \frac{4((2 \times 1194)^3 + 1)}{0{,}05}} \approx 0{,}22529 + 0{,}43093 = 0{,}65622$$

con probabilidad al menos 0.95.

Podemos también obtener una cota para el error en el conjunto de test (E_{test}) de manera parecida a la anterior usando también la desigualdad de Hoeffding,

$$E_{out}(g) \leq E_{test}(g) + \sqrt{\frac{1}{2N} \log \frac{2}{\delta}} = E_{test}(g) + \sqrt{\frac{1}{2 \times 366} \log \frac{2}{0{,}05}} \approx 0{,}25409 + 0{,}07098 \approx 0{,}32508$$

con probablidad al menos 0.95. Hemos usado que para nuestro conjunto de test $N = 366$, y $\delta = 0{,}05$.

Cabe notar que ahora la hipótesis g está fija, es decir, la hemos elegido y por tanto no hay más funciones como en el caso anterior. Esto hace que la cota obtenida usando E_{test} sea bastante mejor que la obtenida usando E_{in}, pues como vemos es mucho menor. Además, el hecho de usar datos que no se han utilizado para el aprendizaje también hace que esta cota sea más representativa (es independiente de los datos de entrenamiento).