Práctica 2 | Complejidad de $\mathcal{H}$ y modelos lineales

Ejercicio 1. Complejidad de $\mathcal{H}$ y ruido

Ejercicio 1.1. Dibujar gráficas con las nubes de puntos simuladas con las siguientes condiciones:

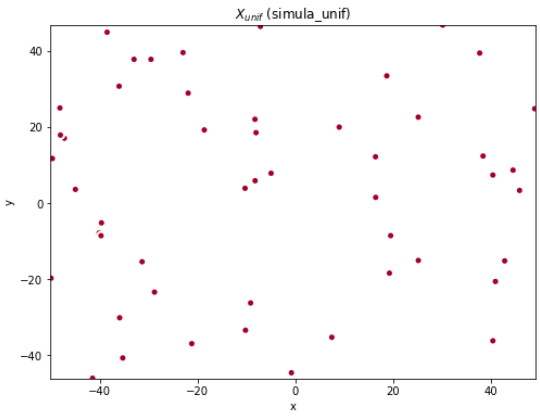
a) Considere `N` = 50, `dim` = 2 y `rango` = [–50, +50] con `simula_unif(N, dim, rango)`.

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

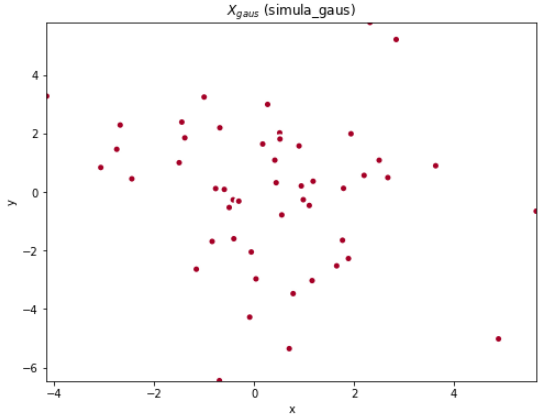
Hacemos uso de las funciones sugeridas. Las distribuciones uniforme $X \rightsquigarrow \mathcal{U}(a, b)$ y gaussiana $Y \rightsquigarrow \mathcal{N}(\sigma, \mu)$ vienen determinadas por la funciones de densidad:

$$f_X(x) = \frac{1}{b-a}, \quad a < x < b \qquad\qquad f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}$$

Siguiendo la dimensión especificada (que nos permite su correcta visualización) y los rangos y desviaciones, obtenemos las siguientes nubes de puntos:



Gráfica 1.1.1



Gráfica 1.1.2

Es interesante resaltar cómo los datos en la distribución gaussiana están más acumulados en el centro. Esto es consecuencia de las propiedades de la distribución: la mayoría de los valores se concentran en la región $[-\sigma_x, \sigma_x] \times [-\sigma_y, \sigma_y]$ (el 68.26 % de ellos), que en nuestro caso, dado que $\sigma_x = \sqrt{5}, \sigma_y = \sqrt{7}$, se trata de la región $[-2.23, 2.23] \times [-2.64, 2.64]$.

Ejercicio 1.2. Vamos a valorar la influencia del ruido en la selección de la complejidad de la clase de funciones. Con ayuda de la función `simula_unif`$(100, 2, [-50, 50])$ generamos una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función $f_1(x, y) = y - ax - b$, es decir el signo de la distancia de cada punto a la recta simulada con `simula_recta`().

a) Dibujar un gráfico 2D donde los puntos muestren (use colores) el resultado de su etiqueta. Dibuje también la recta usada para etiquetar. (Observe que todos los puntos están bien clasificados respecto de la recta).

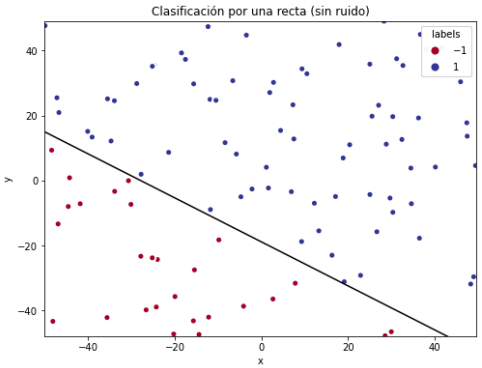
b) Modifique de forma aleatoria un 10 % de las etiquetas positivas y otro 10 % de las negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior. (Ahora habrá puntos mal clasificados respecto de la recta).

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

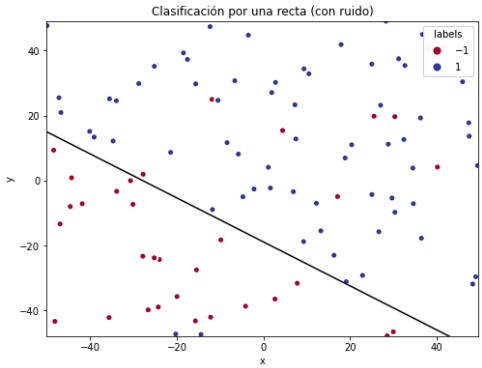
$$f_2(x, y) = (x - 10)^2 + (y - 20)^2 - 400$$
$$f_3(x, y) = 0.5(x + 10)^2 + (y - 20)^2 - 400$$
$$f_4(x, y) = 0.5(x - 10)^2 - (y + 20)^2 - 400$$
$$f_5(x, y) = y - 20x^2 - 5x + 3$$

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. Argumente si estas funciones más complejas son 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.

Primero, realizamos lo indicado en el apartado a), obteniendo una clasificación perfecta (*Gráfica 1.2.1*). Al introducir un 10 % de ruido sobre las etiquetas de cada clase, como se explicita en el apartado b), obtenemos que el porcentaje de datos bien clasificados se reduce al 91 % (*Gráfica 1.2.2*).

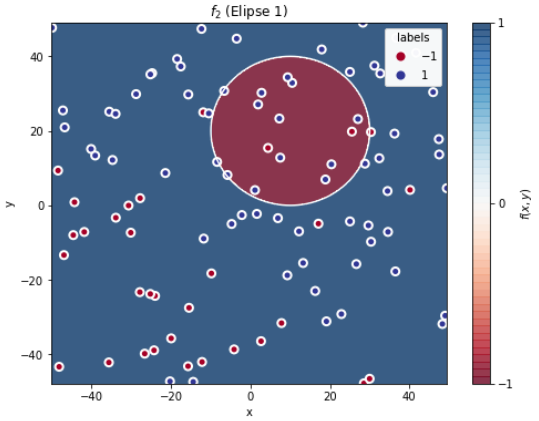


Gráfica 1.2.1

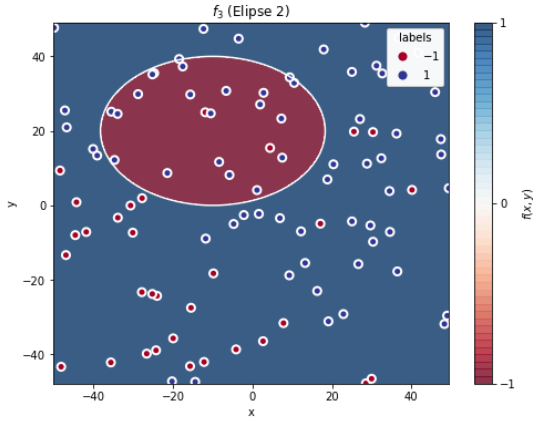


Gráfica 1.2.2

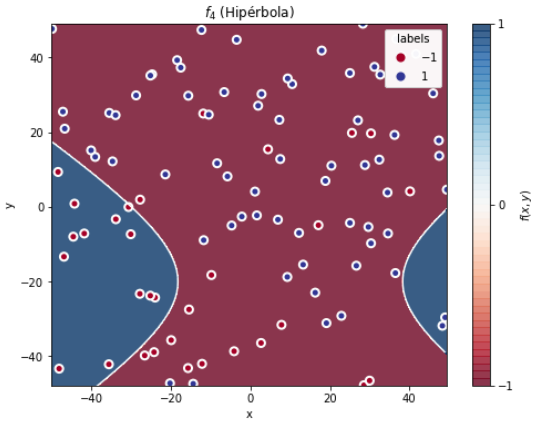
A continuación, de acuerdo al apartado c), estudiaremos cómo se comportan clasificadores más complejos al clasificar los datos perturbados por el ruido. Tengamos en cuenta que hemos obtenido las etiquetas con un clasificador sencillo: la recta f_1. Nos preguntamos por tanto qué ocurre si al aumentar la complejidad de la clase de funciones $\mathcal{H}$ nuestros datos serán mejor clasificados. Para ello, probamos con las funciones f_2 a f_5 del enunciado de este ejercicio. Obtenemos de este modo las siguientes regiones de clasificación. Nótese que, dada la fórmula que seguimos para clasificar, la curva que separa las regiones de clasificación +1 y –1 es la que viene determinada de forma implícita por $f(x, y) = 0$, que destacamos en blanco para cada clasificador.



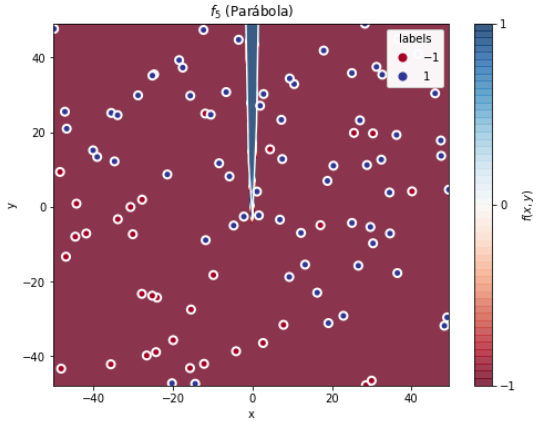
Gráfica 1.2.4



Gráfica 1.2.5



Gráfica 1.2.6

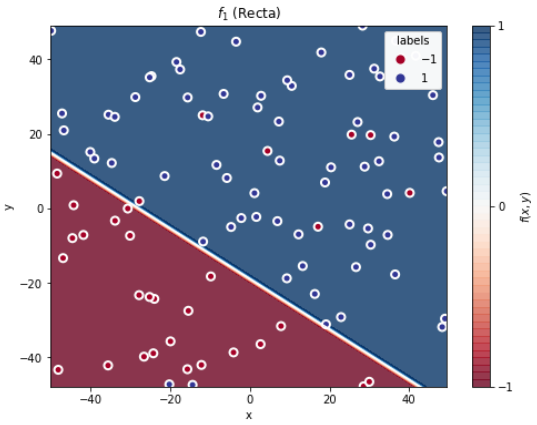


Gráfica 1.2.7

Vemos cómo, a simple vista, aun siendo las funciones más complejas, no consiguen clasificar los datos correctamente. Cerciorémonos de esto viendo el porcentaje de datos bien clasificados para cada caso:

Clasificador	Bien clasificados (%)
f_1 (Recta)	91.0
f_2 (Elipse 1)	58.0
f_3 (Elipse 2)	50.0
f_4 (Hipérbola)	21.0
f_5 (Parábola)	32.0

En efecto, vemos que ninguna de las funciones clasificadores, pese a ser más compleja que f_1 (la recta), logra clasificar mejor los datos.



Gráfica 1.2.3

Esto nos conduce a la conclusión de que el aumentar la complejidad de la clase de funciones no implica un mejor ajuste. De hecho, es posible que, haciendo que las funciones sean muy complejas, lleguemos a entrar en el problema de *overfitting* (sobreajuste), impidiendo una buena generalización.

En definitiva, en este ejercicio hemos visto la importancia y dificultad de elegir un buen $\mathcal{H}$.

Ejercicio 2. Modelos lineales

Ejercicio 2.a) (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.

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.
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 cuál y las razones para que ello ocurra.

Para poder realizar este ejercicio, implementamos en primer lugar el algoritmo perceptron (PLA, *Perceptron Learning Algorithm*). Éste es un algoritmo que calcula el hiperplano solución (en forma de vector de pesos) a un problema de clasificación binaria. La idea es sencilla: para cada dato $\mathbf{x}(t)$, si el vector de pesos hasta ese momento $\mathbf{w}(t)$ no lo ajusta correctamente según $y(t)$, es decir, si $\text{sign}(\mathbf{w}(t)^T\mathbf{x}(t)) \neq y(t)$ actualizamos $\mathbf{w}$ de acuerdo a la regla $\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t)$. Ejecutaremos PLA en los mismos datos, etiquetados de dos formas diferentes:

1) Usando las etiquetas del apartado 2a de la sección 1. Las etiquetas se generaron a partir de una recta y sin ruido, por lo que, en este caso, estamos ante datos **linealmente separables**.
2) Usando las etiquetas del apartado 2b de la sección 1. Las etiquetas son las del apartado anterior, con cierto ruido, luego estamos ante datos **no linealmente separables**.

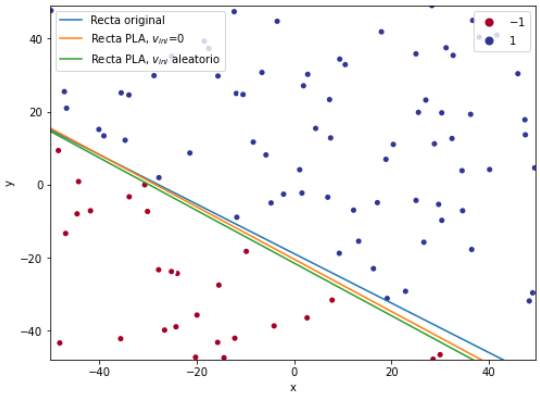
Esta propiedad de separabilidad lineal será esencial para comprender cómo actúa el algoritmo perceptron. En ambos casos ejecutaremos el algoritmo con diversos `vini` (w iniciales): partiendo del vector cero y de 10 vectores aleatorios. Los resultados medios que obtenemos son los siguientes:

Etiquetas	Vector inicial	Num. reps.	Num. iteraciones	Bien clasificadas (%)
Sin ruido	Ceros	1	75.0000	100.0000
	Aleatorio	10	157.7000 ± 94.1754	100.0000 ± 0.0000
Con ruido	Ceros	1	1000.0000	83.0000
	Aleatorio	10	1000.0000 ± 0.0000	80.9000 ± 4.1340

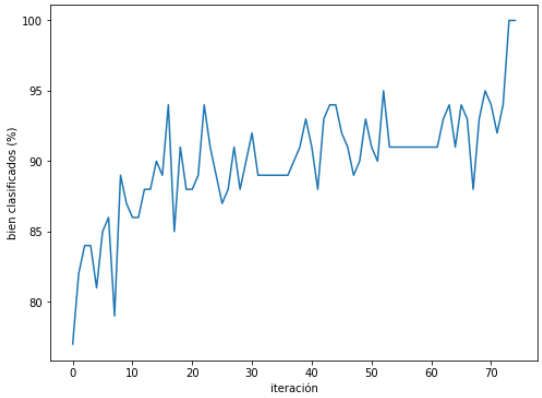
Analicemos primero el resultado teniendo en cuenta el punto de inicio, en el caso de las etiquetas con ruido (pues si tienen ruido, PLA no converge, como veremos en un momento). En efecto, la elección del punto de inicio es relevante de cara a analizar la convergencia de nuestro algoritmo (el número de iteraciones necesarias). Vemos cómo al iniciar el algoritmo con el vector cero, las iteraciones necesarias son en general menores que la media en el caso de inicios aleatorios. Sin embargo, notemos cómo la desviación en el caso de los vectores iniciales aleatorios es considerable. Esto nos conduce a pensar que una buena elección de vector inicial es esencial, pues el número de iteraciones necesarias para converger (en caso de que lo haga) está directamente relacionada con la elección de tal vector.

Hagamos un comentario adicional en el caso de las etiquetas sin ruido. Vemos cómo, dado que los datos son linealmente separables, conseguimos que la clasificación sea perfecta. Sin embargo, en la *Gráfica 2.a).1* podemos ver cómo el clasificador que consigue un ajuste perfecto no es único. Por otra parte, podemos ver en la *Gráfica 2.a).2* cómo, aunque la cantidad de puntos bien clasificados en PLA va oscilando, tiende a aumentar.

Etiquetas sin ruido

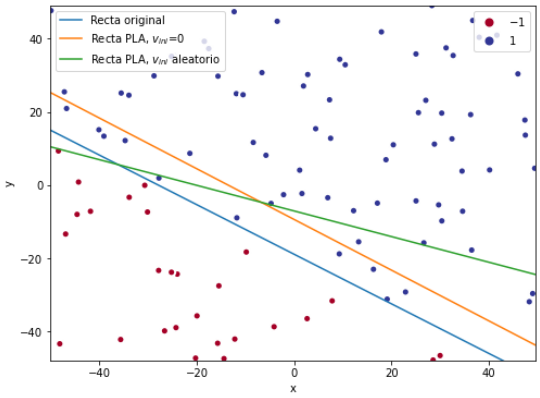


Gráfica 2.a).1

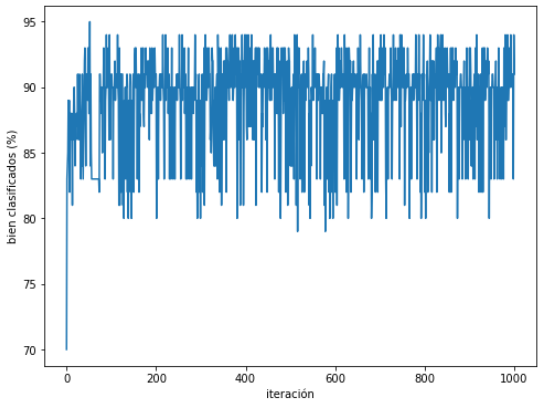


Gráfica 2.a).2

Etiquetas con ruido



Gráfica 2.a).3



Gráfica 2.a).4

Finalmente, valoremos qué ocurre con las etiquetas con ruido. En este caso, el algoritmo PLA no converge, y es por ello por lo que, en todos los casos, el algoritmo para por llegar al número máximo de iteraciones. Dado que los datos no son linealmente separables, no existe ninguna recta que pueda clasificarlos correctamente, y es por ello por lo que el porcentaje de bien clasificadas nunca es el total, a pesar de que logra acercarse. De hecho, es interesante ver que, incluso teniendo un vector inicial aleatorio, el porcentaje de datos que logra clasificar correctamente no varía excesivamente.

Valoremos ahora cómo de adecuadas son las clasificaciones obtenidas por el algoritmo. Como podemos ver en la *Gráfica 2.a).3*, el algoritmo PLA no consigue acercarse a la recta que usamos para generar los datos, algo que podíamos predecir. Además, es especialmente interesante cómo evoluciona el porcentaje de puntos bien clasificados con las iteraciones del algoritmo. Como podemos ver en la *Gráfica 2.a).4*, la tendencia es, al igual que en el caso de las etiquetas sin ruido, oscilante, pero en este caso de forma mucho más notable y sin crecer de forma regular. Esto también es esperable: dado que el algoritmo no tiene forma de conocer cómo evoluciona y, dado que los datos no son linealmente separables, las rectas originadas cambiarán notablemente las etiquetas de una iteración a otra.

Ejercicio 2.b) (Regresión Logística). En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos $\mathcal{D}$ 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 $\mathbf{x}$. Consideremos $d = 2$ para que los datos sean visualizables, y sea $X = [0, 2] \times [0, 2]$ con probabilidad uniforme de elegir cada $x \in \mathcal{X}$. Elegir una línea en el plano que pase por $\mathcal{X}$ como la frontera entre $f(\mathbf{x}) = 1$ (donde y toma valores +1) y $f(\mathbf{x}) = 0$ (donde y toma valores –1), para ello seleccionar dos puntos aleatorios de $\mathcal{X}$ y calcular la línea que pasa por ambos.

EXPERIMENTO: Seleccione $N = 100$ puntos aleatorios $\{\mathbf{x}_n\}$ de $\mathcal{X}$ y evalúe las respuestas $\{y_n\}$ de todos ellos respecto de la frontera elegida. Ejecute Regresión Logística (ver condiciones más abajo) para encontrar la función solución g y evalúe el error E_{out} usando para ello una nueva muestra grande de datos (> 999). Repita el experimento 100 veces, y

- Calcule el valor de E_{out} para el tamaño nuestral $N = 100$.
- Calcule cúantas épocas tarda en converger en promedio RL para $N = 100$ en las condiciones fijadas para su implementación.

(continúa en página siguiente)

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 $\|\mathbf{w}^{(t-1)} - \mathbf{w}^{(t)}\| < 0.01$, donde $\mathbf{w}^{(t)}$ 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 de $\{1, 2, ..N\}$ a los índices de los datos, antes de usarlos en cada época del algoritmo.
- Usar una tasa de aprendizaje $\eta = 0.01$.

Implementaremos la regresión logística con el algoritmo de descenso de gradiente estocástico, de acuerdo a lo descrito en el enunciado. Es importante notar que estamos aplicando regresión logística a un problema de clasificación binaria. La forma de traducir el *output* de la regresión logística a una clasificación es establecer un umbral de probabilidad, y tener que $y = 1$ cuando $\theta(w^T x) \geq 0.5$ y $y = -1$ en caso contrario (θ es la función logística, usaremos a continuación toda la notación explicada en clase de teoría). La condición $\theta(w^T x) \geq 0.5$ es equivalente a que $w^T x \geq 0$, así que podemos decir que $y = \text{sign}(w^T x)$, y utilizar todo lo hasta ahora implementado para los ejercicios y apartados anteriores.

El algoritmo de descenso de gradiente pretende minimizar el error *in-sample*, que en el caso de la regresión logística es:

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

Además, calculando el gradiente del error y usando un *minibatch* de N datos tenemos que:

$$\nabla E_{in}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}$$

En la función `lr_sgd` se encuentra la implementación del algoritmo SGD para el caso de regresión logística, teniendo en cuenta lo anteriormente expuesto, y las indicaciones del enunciado (condición de parada, tasa de aprendizaje, etc.). Además, sigue la recomendación de tamaño de batch $N = 1$. No he considerado necesario dar explicaciones en más profundidad al respecto, dado que este algoritmo fue abordado en la práctica anterior.

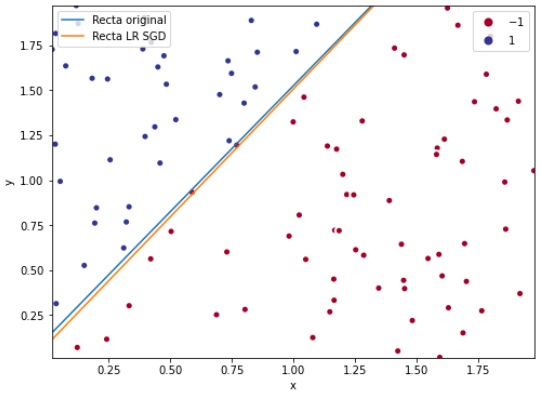
Vamos a realizar el experimento sugerido:

1. Generamos X_{train}, un conjunto de $N = 100$ datos de entrenamiento de dimensión $d = 2$ en $[0, 2] \times [0, 2]$, mediante `simula_unif`. Del mismo modo generamos X_{test} con $N = 1000$ datos.
2. Generamos f_{gen}, una recta que corte la región anterior (determinada por $\mathbf{w}_{gen}$), mediante `simula_recta`.
3. Calculamos y_{train} mediante f_{gen} con X_{train}. Análogamente calculamos y_{test} con X_{test}.
4. Aplicamos regresión logística para obtener $\mathbf{w}_{lr}$. Calculamos E_{in} a partir de X_{train}, y_{train} y $\mathbf{w}_{lr}$, y E_{out} a partir de X_{test} e y_{test}.

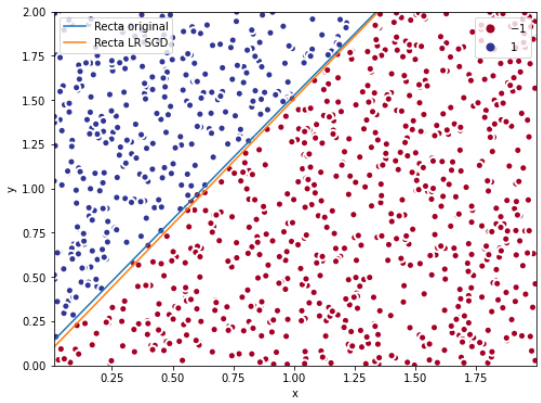
Repetiremos este experimento un total de 1000 veces. De este modo obtenemos los siguientes resultados medios (donde P es el porcentaje de datos bien clasificados):

$$\begin{aligned} E_{in} &= 0.0905 \pm 0.0257 \\ E_{out} &= 0.1023 \pm 0.0273 \\ P_{in} &= 98.0000 \\ P_{out} &= 99.4000 \\ \text{épocas} &= 360.5300 \pm 84.9121 \end{aligned}$$

Una vez hecho esto, extraigamos algunas conclusiones. En primer lugar, vemos cómo este algoritmo logra obtener unos resultados excelentes, con un porcentaje de datos bien clasificados muy alto. Además, la generalización es bastante buena, dado que E_{out} se incrementa (algo esperable dado que tenemos más datos en *testing*) muy poco respecto de E_{in}. En las gráficas siguientes podemos ver cómo la recta obtenida es bastante cercana a la que usamos para etiquetar los datos, y que aproxima bien los puntos tanto en los datos de *train* (*Gráfica 2.b).1*) como en los datos de *test* (*Gráfica 2.b).2*).



Gráfica 2.b).1



Gráfica 2.b).2

Por otra parte, analicemos el número de épocas requerido para converger a una solución, así como la convergencia. El resultado obtenido no logra clasificar la totalidad de los datos, pero esto es razonable del criterio de parada establecido –de todos modos, obtenemos resultados bastante buenos, como hemos dicho anteriormente–. Además, la parada del algoritmo depende de cómo se van recorriendo los distintos puntos, y dado que este recorrido es aleatorio, podemos ver justificada la alta desviación que tienen las épocas necesarias para parar de los 100 experimentos realizados.

En definitiva, los resultados nos conducen a pensar que el descenso de gradiente estocástico es un método adecuado para la clasificación binaria a partir de regresión logística.

Bonus. Clasificación de Dígitos

Ejercicio Bonus (Algoritmo PLA-Pocket). Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 4 y 8. Usar los ficheros de entrenamiento (*training*) y *test* que se proporcionan. Extraer las características de **intensidad promedio** y **simetría** en la manera que se indicó en el ejercicio 3 del trabajo 1.

1. Plantear un problema de clasificación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g.
2. Usar un modelo de Regresión Lineal y aplicar PLA-Pocket como mejora. Responder a las siguientes cuestiones.
   a) Generar gráficos separados (en color) de los datos de entrenamiento y test junto con la función estimada.
   b) Calcular E_{in} y E_{test} (error sobre los datos de test).
   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 $\delta = 0.05$. ¿Qué cota es mejor?

Apartado 1

Plantearemos en primer lugar el problema de clasificación. Dado que tenemos dos características, el espacio $\mathcal{X}$ será $\mathcal{X} = \{1\} \times \mathbb{R}^2$. Como estamos ante un problema de clasificación binaria, será $\mathcal{Y} = \{-1, +1\}$, donde –1 corresponde al dígito 4 y +1 corresponde al dígito 8.

El objetivo de nuestro problema es aproximar la función $f : \mathcal{X} \longrightarrow \mathcal{Y}$, completamente desconocida. Esta estimación la haremos a partir de un conjunto de datos de *training*, $\mathcal{D}$:

$$\mathcal{D} = \{(x_n, y_n) \subset \mathcal{X} \times \mathcal{Y} : n \in \{1, ..., N\}\}$$

con N el número de datos de entrenamiento. Suponiendo una distribución de probabilidad $\mathcal{P}$ en $\mathcal{X} \times \mathcal{Y}$, y que los datos de $\mathcal{D}$ son ocurrencias independientes e idénticamente distribuidas en $\mathcal{P}$, estimaremos una función $g \in \mathcal{H}$, siendo el conjunto de hipótesis posible:

$$\mathcal{H} = \{h : \mathbb{R}^3 \longrightarrow \mathbb{R} : h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}), \mathbf{w} \in \mathbb{R}^3\}$$

Nuestro problema usará la técnica ERM (minimización de riesgo empírico), teniendo en cuenta el error de clasificación binaria:

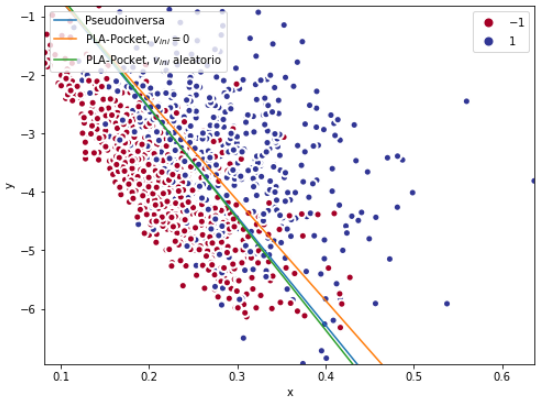
$$E_{in}(h) = \frac{1}{N}[\![h(\mathbf{x}_n) \neq y_n]\!] \quad \forall h \in \mathcal{H} \qquad\qquad \text{ERM}_{\mathcal{H}}(\mathcal{D}) = \text{argmin}_{h \in \mathcal{H}}\{E_{in}(h)\}$$

Como algoritmos de aprendizaje ($\mathcal{A}$) usaremos dos: la regresión lineal (usaremos el método de la pseudoinversa) y PLA-Pocket.

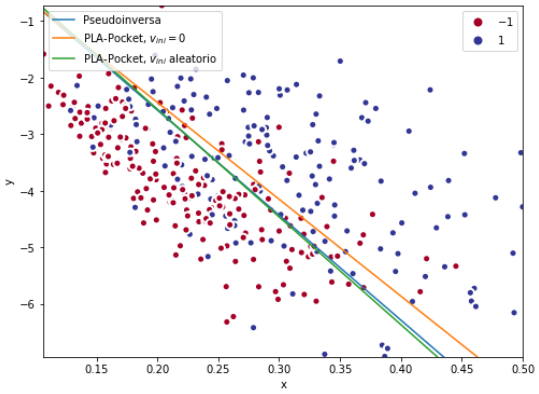
Apartado 2

Aplicaremos los algoritmos que hemos mencionado anteriormente. Además, dado que PLA-Pocket necesita de un vector inicial, haremos dos ejecuciones de este último: una que parta del vector cero y otra de un vector aleatorio. La implementación del algoritmo de pseudoinversa se encuentra en `ajusta_pseudoinverse` y la de PLA-Pocket en `ajusta_pla_pocket`. El algoritmo PLA-Pocket es muy parecido al de PLA, con la funcionalidad añadida de que va guardando el vector de pesos que tenga menor error, y lo "sobreescribiremos" sólo en caso de encontrar uno con error menor.

Ejecutamos los algoritmos y obtenemos los hiperplanos que podemos ver en las gráficas a continuación, con los datos de *training* (*Gráfica B.1*) y *testing* (*Gráfica B.2*).

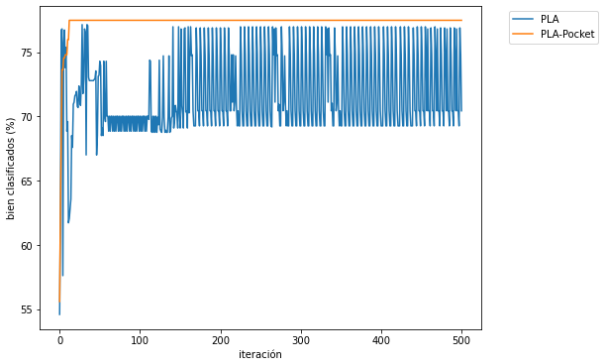


Gráfica B.1



Gráfica B.2

Vemos que los datos, claramente, no son linealmente separables, por lo que podemos esperar en el caso de PLA-Pocket un comportamiento similar a PLA con la mejora de que, a diferencia de PLA, en PLA-Pocket tenemos en cuenta el vector de pesos que ha proporcionado el mejor error, y lo vamos reemplazando en caso de que lo minimice. Es por ello por lo que es evidente que el porcentaje de datos bien clasificados sólo irá incrementándose (al igual que irá disminuyéndose el error *in-sample*). Este comportamiento es deseable en comparación al del algoritmo PLA, como podemos ver en la *Gráfica B.3*.



Gráfica B.3

A continuación, observemos la información obtenida de las ejecuciones.

$$\begin{aligned} \text{Pseudoinversa}: \quad & E_{in} = 0.2278 \\ & E_{test} = 0.2514 \\ \text{PLA-Pocket } (v_{ini} = 0): \quad & E_{in} = 0.2286 \\ & E_{test} = 0.2459 \\ \text{PLA-Pocket } (v_{ini} \text{ aleatorio}): \quad & E_{in} = 0.2253 \\ & E_{test} = 0.2541 \end{aligned}$$

Vemos que, al igual que en PLA, la elección del vector inicial es fundamental a la hora de encontrar una solución más adecuada de forma más rápida. Del mismo modo, vemos cómo PLA-Pocket consume todas las iteraciones, dado que los datos no son linealmente separables.

Dado que tenemos E_{test} y queremos conocer el valor de E_{out} con $\mathcal{X}$ desconocido, recurriremos a diversas cotas, con tolerancia $\delta = 0.05$. Estas no son otras que la proporcionada por la dimensión VC y por la desigualdad de Hoeffding.

Cota por dimensión VC

Para un $\delta > 0$ es

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} \log\left(\frac{4 \cdot ((2N)^{d_{VC}} + 1)}{\delta}\right)} \qquad \text{con probabilidad} \geq 1 - \delta$$

donde d_{VC} es la dimensión VC del clasificador y N es el tamaño del conjunto de *test*. En nuestro caso, $d_{VC} = 3$. Tomando la tolerancia especificada y la muestra usada anteriormente, tenemos que:

$$\begin{aligned} \text{Pseudoinversa} &: E_{out}(g) \leq 0.6587 \\ \text{PLA-Pocket } (v_{ini} = 0) &: E_{out}(g) \leq 0.6596 \\ \text{PLA-Pocket } (v_{ini} \text{ aleatorio}) &: E_{out}(g) \leq 0.6562 \end{aligned}$$

Cota por desigualdad de Hoeffding

Para un $\delta > 0$ es

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} \log\left(\frac{2\,|\,\mathcal{H}\,|}{\delta}\right)} \qquad \text{con probabilidad} \geq 1 - \delta$$

donde N es el tamaño del conjunto de *test* y $\mathcal{H}$ el espacio de funciones hipótesis.

Dado que en nuestro caso es $|\mathcal{H}| = \infty$, debemos discretizar nuestro conjunto para tener en cuenta la precisión de los ordenadores. Si cada flotante ocupa 64 bits y tenemos vectores de tres posiciones, tenemos $|\mathcal{H}| = 2^{64 \cdot 3}$. Por otra parte, podemos calcular la cota usando E_{test}, en cuyo caso $|\mathcal{H}| = 1$, dado que al calcular el error en *test* ya hemos fijado nuestro g, que es único.

Usando E_{in} obtenemos:

$$\begin{aligned} \text{Pseudoinversa} &: E_{out}(g) \leq 0.4671 \\ \text{PLA-Pocket } (v_{ini} = 0) &: E_{out}(g) \leq 0.4680 \\ \text{PLA-Pocket } (v_{ini} \text{ aleatorio}) &: E_{out}(g) \leq 0.4646 \end{aligned}$$

Mientras que usando E_{test}:

$$\begin{aligned} \text{Pseudoinversa} &: E_{out}(g) \leq 0.3224 \\ \text{PLA-Pocket } (v_{ini} = 0) &: E_{out}(g) \leq 0.3196 \\ \text{PLA-Pocket } (v_{ini} \text{ aleatorio}) &: E_{out}(g) \leq 0.3251 \end{aligned}$$

Vemos que las cotas de la desigualdad de Hoeffding son mejores que las de la dimensión VC. Por otra parte, vemos cómo las cotas proporcionadas por E_{test} son mejores, algo esperable al calcular el error sobre datos no vistos previamente (algo que es más fehaciente pues son datos de la distribución que no hemos visto, en lugar de datos con los que hemos entrenado al modelo).