UNIVERSIDAD DE GRANADA

Aprendizaje Automático

CURSO 2018/2019

Trabajo 2: Programación

Laura Rabadán Ortega

1. EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO

1. Dibujar una gráfica con la nube de puntos de salida correspondiente.
   a. Considere $N = 50$, $dim = 2$, $rango = [-50, +50]$ con $simula\_unif(N, dim, rango)$.

Con ayuda de la función $simula\_unif$ se generan los 50 puntos aleatorios y uniformes de 2 dimensiones dentro del cuadrado $[-50, 50] \times [-50, 50]$. Esos puntos se representan en forma de nube de puntos, en la que el eje x representa la primera coordenada del punto y el eje y, la segunda coordenada del punto representado. Esa representación es la mostrada en la *Ilustración 1*.

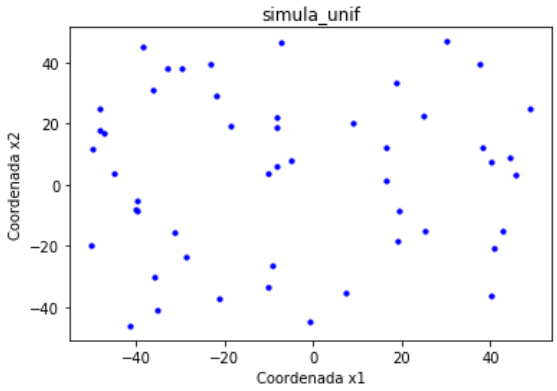

Ilustración 1 - simula_unif(50, 2, [-50, +50])

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

En este caso, se utiliza la función $simula\_gaus$ para conseguir los 50 puntos de 2 dimensiones. En este ejemplo, la función genera una lista de 50 valores de dimensión 2 de números aleatorios pertenecientes a una distribución Gaussiana de media 0 y de varianza dependiente de la posición del vector [5,7]. La representación de los puntos conseguidos es la representada en la *Ilustración 2.*

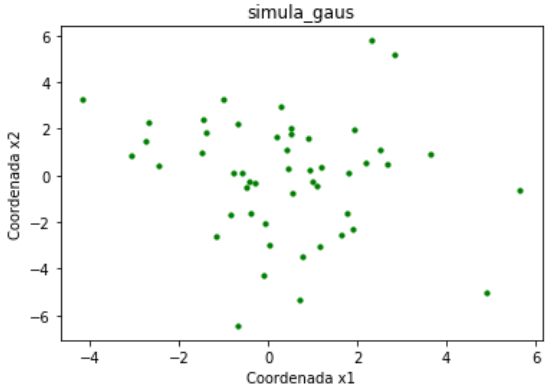

Ilustración 2 - simula_gaus(50, 2, [5,7])

Laura Rabadán Ortega

2. Con ayuda de la función $simula\_unif()$ 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)

Para poder etiquetar los datos conseguidos con $simula\_unif$, se simula una recta, con $simula\_recta$, que corta el cuadrado de valores, en este caso, el intervalo [-50, 50]. Esos valores serían:

$$a = -0.677158 \qquad b = -18.890228$$

Por lo que la recta que divide los datos sería de la forma:

$$y = -0.677x - 18.89$$

Esa recta puede verse representada en la *Ilustración 3*, y será la encargada de asignar la etiqueta a los datos en función al signo conseguido en la función, $f(x, y)$, especificada en el enunciado. Al ser una muestra sin ruido, los datos se presentan completamente clasificados respecto a la recta. Puede comprobarse en la *Ilustración 3* que todos los datos etiquetados como *1* se encuentran por encima de la recta simulada y que todos los datos etiquetados como *-1* se encuentran por debajo de la recta.

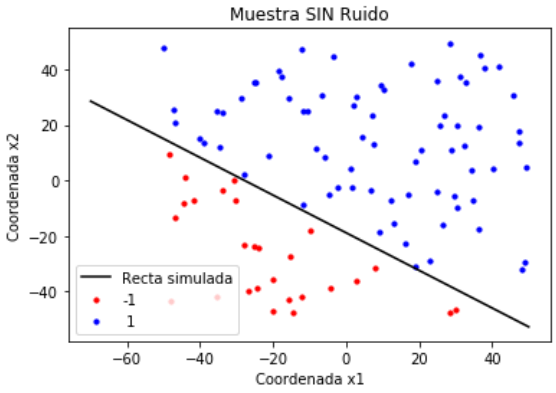

Ilustración 3 - Datos etiquetados sin ruido

    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)

Para este caso, se parte de los datos y etiquetas del apartado anterior, los mostrados en la *Ilustración 3*, por los que los valores de la recta simulada siguen siendo los mismos que en el apartado anterior.

En este apartado, se le añade un porcentaje de ruido, 10%, tanto a las etiquetas positivas como a las etiquetas negativas, por lo que un 20% de los datos estarán mal etiquetados. Al representar los puntos con sus nuevas etiquetas, *Ilustración 4,* se observa que la recta simulada no los separa de forma correcta y perfecta, como ocurría en el apartado anterior, ni se puede intuir ninguna recta similar que sea capaz de separarlos de manera exacta.

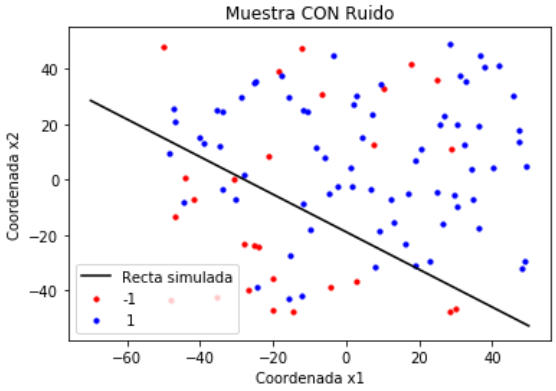

Ilustración 4 - Datos etiquetados con ruido

3. 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 formas de 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? ¿En que ganan a la función lineal? Explicar el razonamiento.

Las gráficas generadas por cada una de las funciones anteriores son las presentadas en la *Ilustración 5*. Como dice el ejercicio y comprobando en las gráficas, las funciones son más complejas que la recta simulada. Que la función sea más compleja no significa que clasifique mejor, como se observa en este caso, ni una función más sencilla lo hará peor. La clasificación de *"función mala"* o *"función buena"* no la determina la complejidad de la función, sino los datos. Por ejemplo, para el caso mostrado en la *Ilustración 3*, una de estas funciones complejas sería muy mala, clasificaría mal, en cambio, una función lineal consigue una clasificación sin error. Por otro lado, para unos datos cuyas etiquetas no siguieran una división lineal, como ocurría en uno de los ejemplos de la práctica anterior en el que una etiqueta "rodeaba" a la otra, la primera función conseguiría una mejor clasificación que una función lineal.

Por tanto, la función que se debe utilizar dependerá de los datos, aunque lo más conveniente sería buscar siempre la más sencilla. Es decir, si una función lineal clasifica de forma correcta los datos, no es necesario buscar funciones más complejas y complicar el estudio.

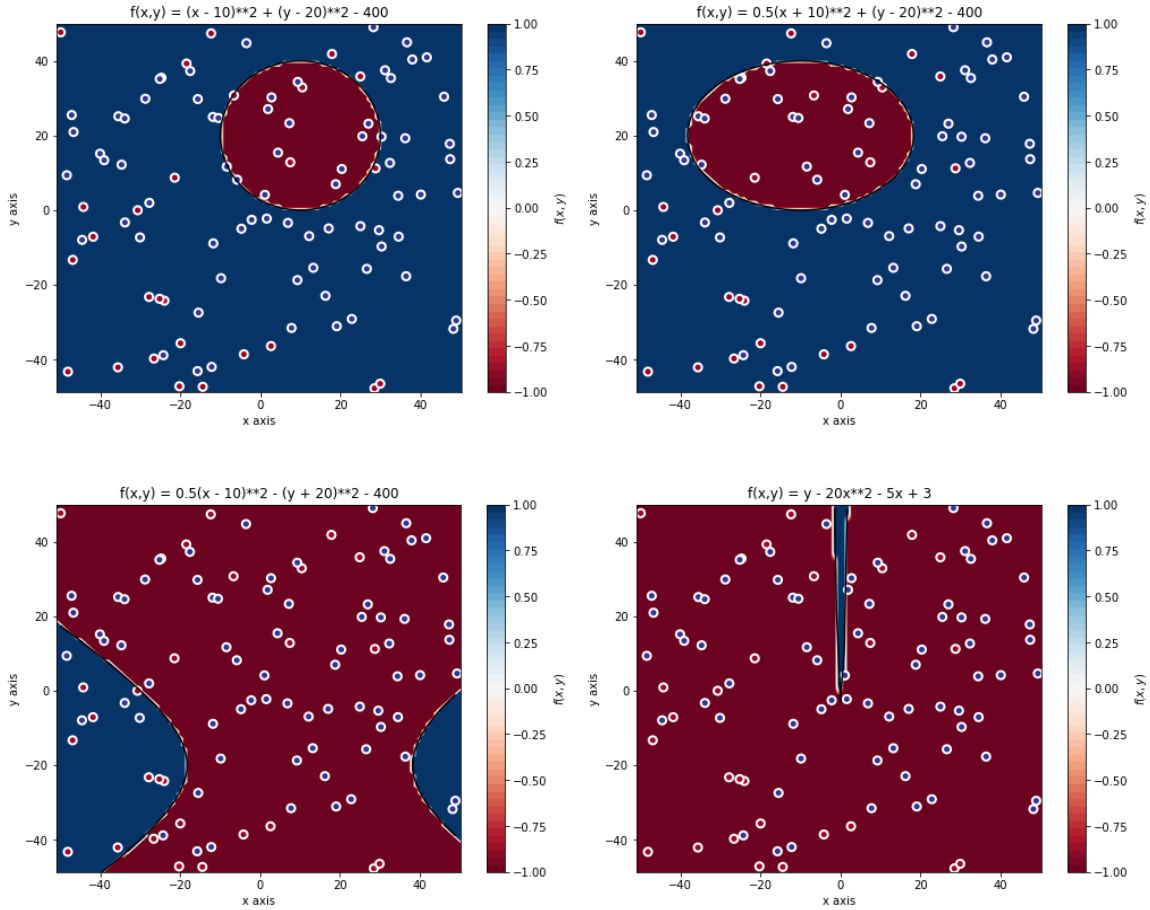

Ilustración 5 - Clasificación con funciones complejas

Estas funciones no consiguen clasificar mejor que la recta simulada en el apartado 2.

2. MODELOS LINEALES

1. **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.

    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.

Inicializando el vector de pesos al valor 0, se consigue converger en 75 iteraciones, consiguiendo, como se ve en la *Ilustración 6*, un ajuste perfecto. En cambio, tras 10 ejecuciones inicializando el vector de pesos a números aleatorios de entre [0,1], se consigue converger con una media de 130.8 iteraciones. Como se observa en la *Tabla 1*, el punto de inicio conlleva más o menos iteraciones, pero no se observa una tendencia de los valores con esta muestra de resultados. Según esos datos, el mejor punto de inicio parece ser un valor intermedio, 0.5, y el peor, un valor próximo a 0, sin ser 0.

Valor Inicial	Número de Iteraciones	Vector de Pesos			Valores de la recta	
					a	b
0	75	23.202	32.392	661.0	-0.716	-20.407
0.618514	43	15.343	23.868	464.619	-0.643	-19.467
0.010364	242	39.841	53.874	1098.010	-0.740	-20.381
0.538627	42	15.175	23.653	456.539	-0.642	-19.301
0.003018	265	40.325	61.053	1139.003	-0.660	-18.656
0.951194	78	21.418	29.606	635.951	-0.723	-21.481
0.905402	140	32.792	48.121	874.905	-0.681	-18.181
0.795967	240	39.497	53.483	1090.796	-0.739	-20.395
0.915274	140	32.802	48.131	874.915	-0.681	-18.178
0.145558	59	19.415	29.794	558.146	-0.652	-18.733
0.157730	59	19.428	29.806	558.158	-0.652	-18.726
	130.8	27.604	40.139	775.104	-0.681	-19.350

Tabla 1 - Estudio de la muestra sin ruido

Los valores del vector de pesos varían en función del punto de inicio, pero producen valores similares, como se ve en la columna de *Valores de la recta*.

El ajuste conseguido en todos los casos, con inicio en 0 y con inicio aleatorio, consigue clasificar de menera correcta todos los datos de la muestra, como se puede comprobar en las *Ilustraciones 6 y 7*.

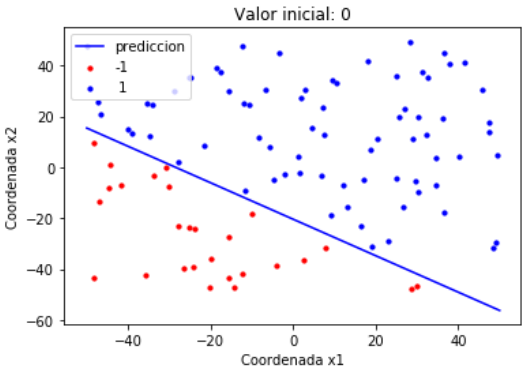

Ilustración 6 - Clasificación de la muestra sin ruido, inicio en 0

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.

En este caso, al algoritmo no logra converger, por lo que agota las iteraciones máximas permitidas de ejecución. Esto no es extraño, ya que, en comparación al apartado anterior, la muestra de este apartado presenta ruido, por lo que la clasificación de los datos no es perfecta. Aún asi, como se observa en la *Tabla 3*, los valores conseguidos en este ajuste son similiares a los conseguidos con la muestra sin ruido.

Con estos resultados, se puede pensar que, aún teniendo ruido en la muestra, el ajuste que se obtiene, en este tipo de casos, será similar al ajuste real, de tal manera que el error que se tendrá a la hora de clasificar nuevos datos será pequeño. El ajuste de la muestra con ruido tenderá al ajuste de la muestra sin ruido.

Valor Inicial	Número de Iteraciones	Vector de Pesos			Valores de la recta	
					a	b
0	1000	37.134	66.040	804	-0.562	-12.174
0.618514	1000	10.705	55.597	806.619	-0.193	-14.508
0.010364	1000	18.359	45.069	806.010	-0.407	-17.884
0.538627	1000	16.493	55.429	815.539	-0.298	-14.713
0.003018	1000	36.314	55.114	814.003	-0.659	-14.770
0.951194	1000	37.929	60.627	807.951	-0.626	-13.327
0.905402	1000	24.406	54.143	803.905	-0.451	-14.848
0.795967	1000	12.490	59.213	798.796	-0.211	-13.490
0.915274	1000	19.807	44.958	854.915	-0.418	-19.016
0.145558	1000	38.353	59.629	826.146	-0.643	-13.855
0.157730	1000	46.367	81.586	799.158	-0.568	-9.795
	1000	26.122	57.137	813.304	-0.447	-14.621

Tabla 2 - Estudio de la muestra con ruido

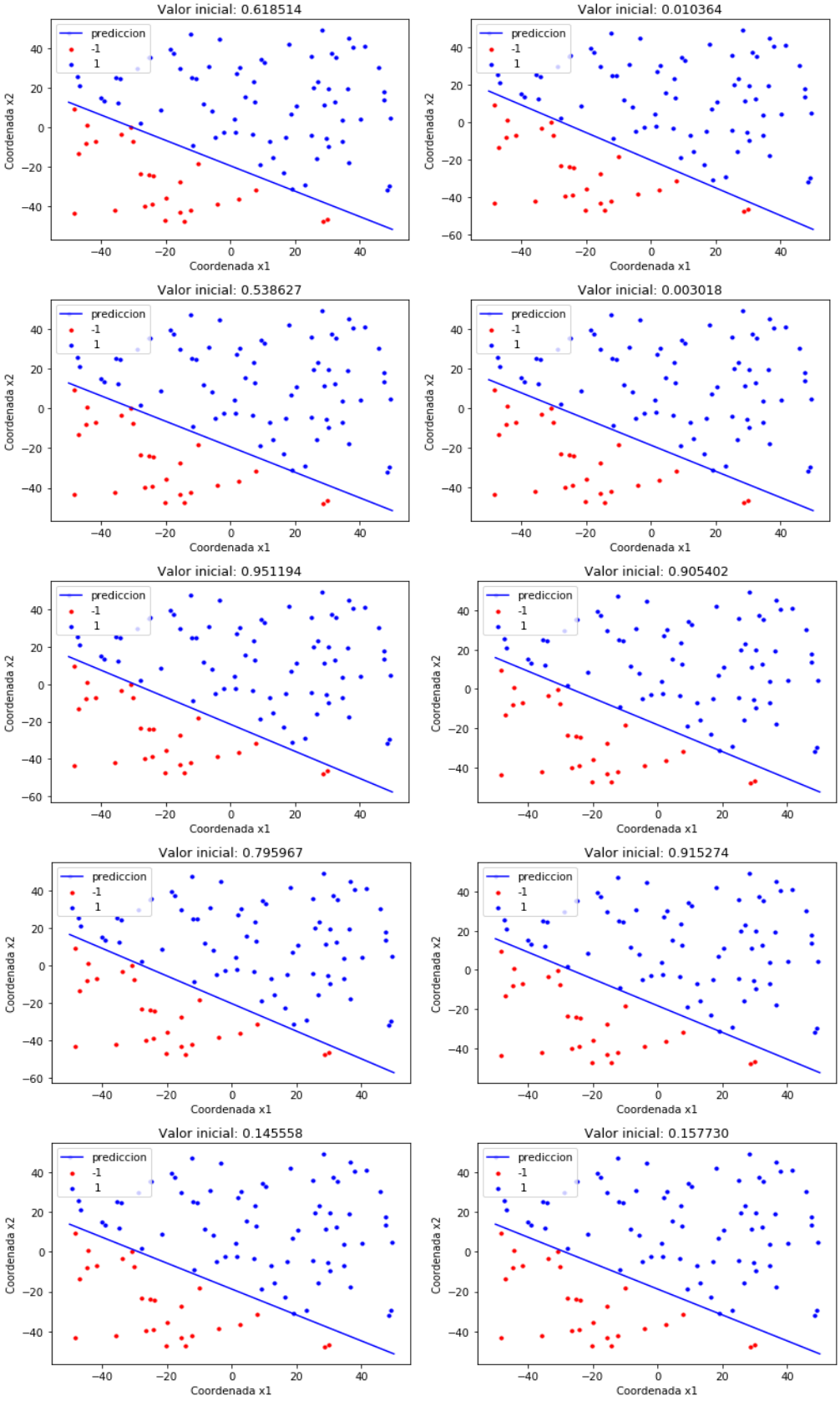

Ilustración 7 - Clasificación de la muestra sin ruido, inicio aleatorio

	Valor Inicial	Número de Iteraciones	Vector de Pesos			Valores de la recta	
						a	b
Sin Ruido	0	75	23.202	32.392	661.0	-0.716	-20.407
	Aleatorio	130.8	27.604	40.139	775.104	-0.681	-19.350
Con Ruido	0	1000	37.134	66.040	804	-0.562	-12.174
	Aleatorio	1000	26.122	57.137	813.304	-0.447	-14.621

Tabla 3 - Comparación de valores medios

En las gráficas de este apartado, *Ilustraciones 8 y 9*, puede observarse que, teniendo en cuenta el ruido que presenta la muestra, la recta simulada en función a la predicción es similar en todas y guarda cierto parecido con la recta conseguida en el apartado anterior.

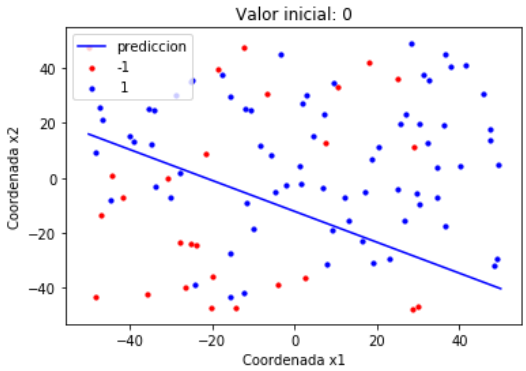

Ilustración 8 - Clasificación de la muestra con ruido, inicio en 0

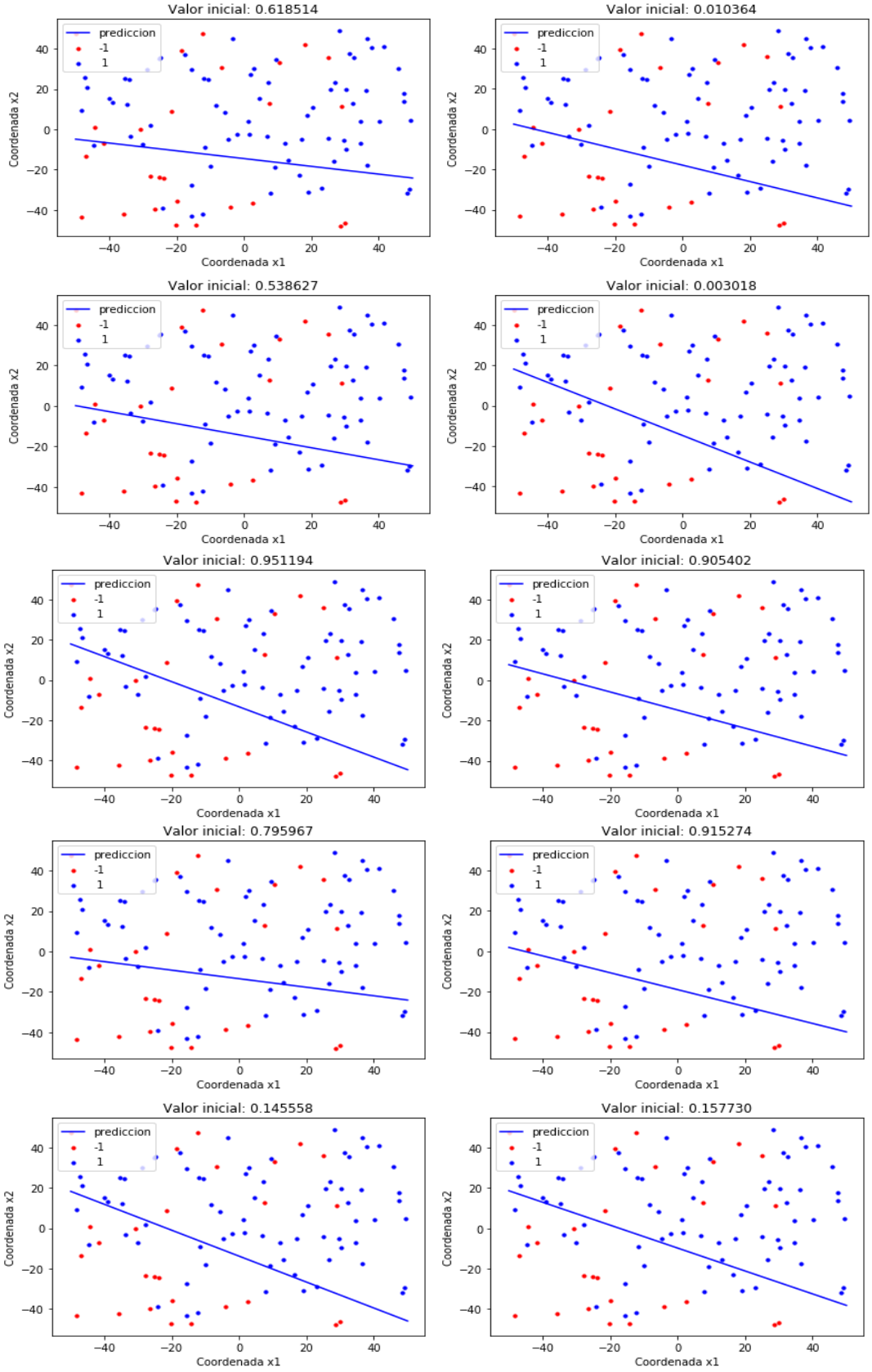

Ilustración 9 - Clasificación de la muestra con ruido, inicio aleatorio

2. **Regresión Logística**: En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos 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 x. Consideremos $d = 2$ para que los datos sean visualizables, y sea $X = [0,2] \times [0,2]$ con probabilidad uniforme de elegir cada $x \in X$. Elegir una línea en el plano que pase por X como la frontera entre $f(x) = 1$ (donde y toma valores $+1$) y $f(x) = 0$ (donde y toma valores -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos. Seleccionar $N = 100$ puntos aleatorios $\{xn\}$ de X y evaluar las respuestas $\{yn\}$ de todos ellos respecto de la frontera elegida.

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

        - 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, $1, 2, \ldots, N$, en el orden de los datos antes de usarlos en cada época del algoritmo.
        - Usar una tasa de aprendizaje de $\eta = 0{,}01$.

Siguiendo esos criterios, el ajuste conseguido es el mostrado en la *Ilustración 10*, tras 374 iteraciones. Una iteración representa una época de los datos, es decir, se recorren todos los datos de la muestra, cogiendo subconjuntos, *batches*, de un tamaño determinado. En este caso, el tamaño del *batch* es de 1. Teniendo en cuenta que la muestra es de tamaño 100 y que se hacen 374 iteraciones, se necesitan 37400 modificaciones del vector de pesos para conseguir un buen ajuste. Como se observa en la gráfica, el ajuste no es exactamente igual a la recta simulada, pero, para los datos de *train*, consigue adaptar perfectamente los datos. Esta perfección podría suponer un sobreajuste en los datos, lo cual podría afectar en el error a la hora de clasificar los datos de la muestra. Esto se estudia en el siguiente apartado.

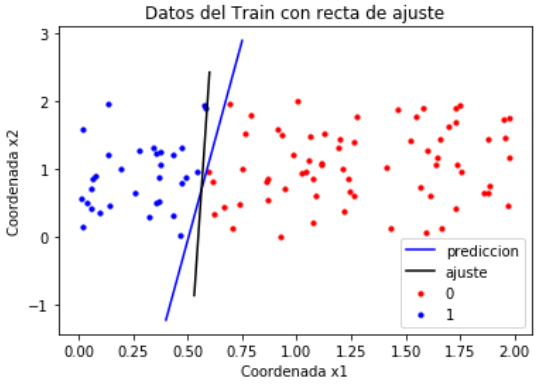

Ilustración 10 - Ajuste con datos de entrenamiento

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).

La solución conseguida es:

$$g = 11.800316x - 5.952521$$

Para este apartado, se usó un conjunto de tamaño 1000 sin etiquetar. Con ayuda de la función calculada en el apartado anterior, se clasificaron los nuevos datos y se calculó el error obtenido, comparando la etiqueta que se le asigna y la que debería asignársele en función a la recta simulada. El error que se consigue es de 0.013, es decir, un 1.3% de los datos están mal clasificados. Como especifiqué, la muestra tiene un tamaño de 1000 datos, lo que quiere decir que 13 datos, de los 1000 que hay, están mal clasificados. Esto supone un error muy pequeño, y en parte se consigue porque la muestra utilizada para predecir la solución g no presenta ruido. Este error se contempla en la gráfica generada, *Ilustración 11*, donde en color verde se pintan los datos que se clasifican de manera incorrecta.

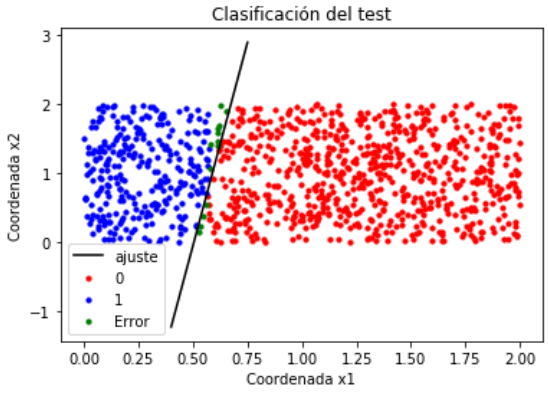

Ilustración 11 - Clasificación de los datos de estudio

3. BONUS

Clasificación de Dígitos. 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?

Para este ejercicio, como indica el enunciado, se utilizan los datos correspondientes a los dígitos 4 y 8, por lo que la clasificación de los datos será -1, que corresponde al dígito 4, y 1, que corresponde al dígito 8. A simple vista, los datos no presentan una clasificación abordable de manera correcta por una función lineal, como se muestra en las *Ilustración 12*.

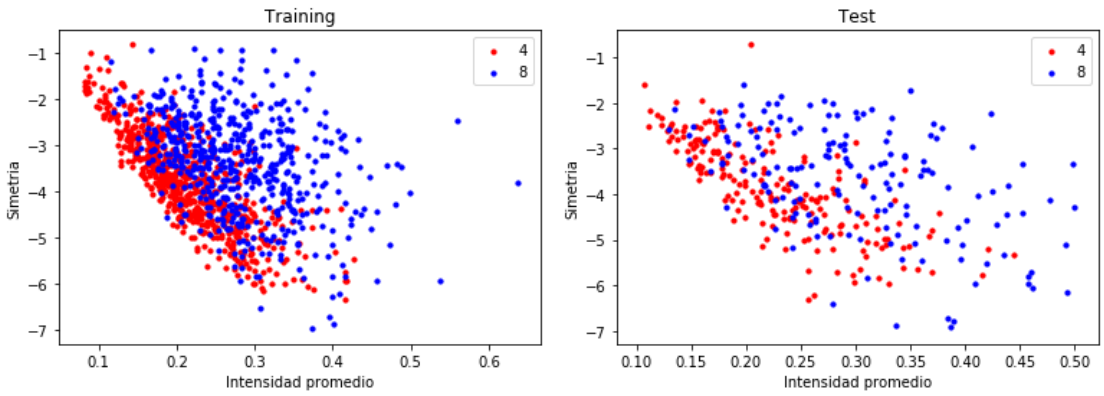

Ilustración 12 - Representación gráfica de los valores

Con ayuda del algoritmo PLA-Pocket, un algoritmo de bolsillo que utiliza PLA, se obtiene que la función g es la siguiente:

$$g = -16.893276x + 0.997867$$

Que gráficamente se puede observar en la *Ilustración 13*. Consigue dividir, visualmente, la mayoría de los datos, pero produce un error en el *Train,* E_{in}, de 0.248634, un 24.8634% de error,

y un error en el *Test*, E_{test}, de 0.231993, un 23.1993% de error. Los errores en ambos conjuntos de datos son similares y el error no es excesivamente alto.

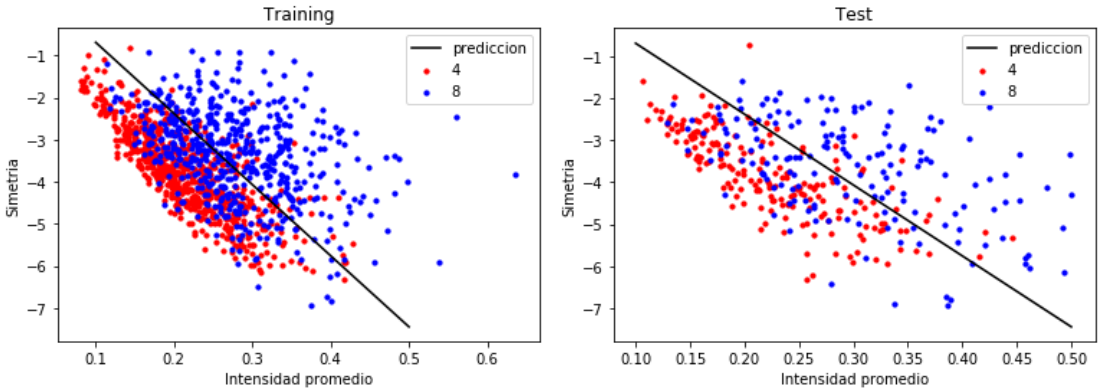

Ilustración 13 - Ajuste de los valores con la función g

Si se calculan las cotas del verdadero valor de E_{out}, en función de los errores E_{in} y E_{test}, usando una tolerancia de $\delta = 0{,}05$, se obtiene:

- Respecto a E_{in} : $E_{out} \leq 0.67957$
- Respecto a E_{test}: $E_{out} \leq 0.95883$

La cota preferible es la que se consigue respecto a E_{test}, ya que indica que se conseguirá un error menor o igual al 95.883, preferible frente % al 67.957% que se podría obtener según los datos del E_{in}.

Si se tiene en cuenta la cota del E_{in}, este método de obtener g para este problema no sería la más apropiada, pues el error es alto. Pero, estudiando la cota conseguida con E_{test}, podría utilizarse este modelo invirtiendo las etiquetas, es decir, cuando el algoritmo devolviera 4, se clasificaría como 8, y cuando devolviera 8, se clasificaría como 4. Con esto, el error se invertiría, por lo que el error se convertiría en 0.04117, es decir, un 4.117% de error.

Esta conversión puede hacerse porque E_{out} respecto a E_{test} es prácticamente 1, es decir, la mayoría de los datos se van a clasificar de manera incorrecta. Como es una clasificación binaria, si se están clasificando la mayoría mal, significa que el valor correcto es el contrario al que se devuelve, por lo que, invirtiendo el valor que se le asigna, se conseguiría una clasificación casi perfecta.

4. REFERENCIAS

Abu-Mostafa, Y. S., Magdon-Ismail, M., & Lin, H.-T. (2012). *Learning From Data A Short Course.* AMLbook.com.

Brownlee, J. (1 de Abril de 2016). *Logistic Regression for Machine Learning*. Obtenido de Machine Learning Mastery: https://machinelearningmastery.com/logistic-regression-for-machine-learning/

DataCamp. (s.f.). *Python For Data Science Cheat Sheet*. Obtenido de DataCamp: https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

Swaminathan, S. (15 de Marzo de 2018). *Logistic Regression — Detailed Overview*. Obtenido de Towards Data Science: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

The SciPy community. (24 de Julio de 2018). *numpy.array_equal*. Obtenido de SciPy.org: https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.array_equal.html

The SciPy community. (31 de Enero de 2019). *numpy.log*. Obtenido de SciPy.org: https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html