Memoria Práctica 2

Aprendizaje Automático

Christian Vigil Zamora
3º - Grupo 2
Abril,2019

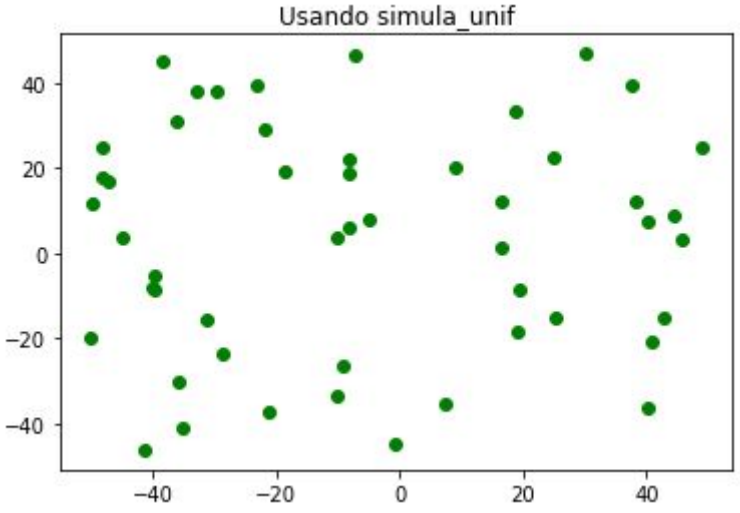
Índice

1. EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO

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

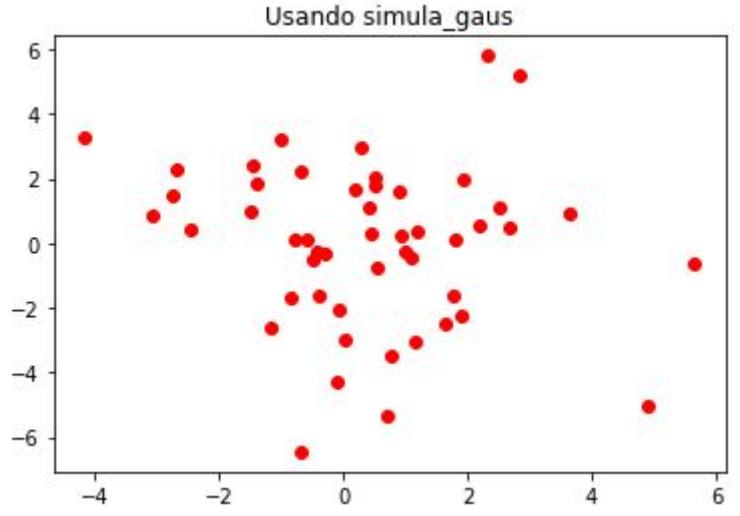
a) Considere N = 50, dim = 2, rango = [-50, +50] con simula_unif(N, dim, rango).

Nube de puntos obtenida con la función simula_unif:



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

Nube de puntos obtenida con la función simula_gaus:

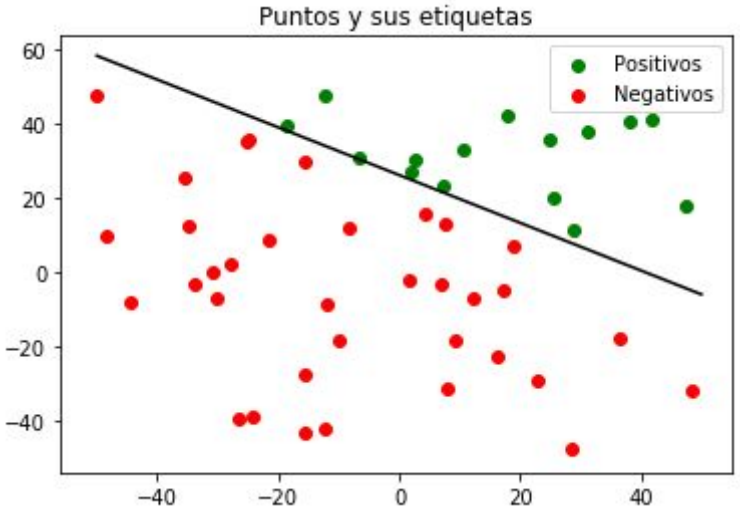


2. Con ayuda de la función simula_unif() generar una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función f(x, y) = y - ax - b, es decir el signo de la distancia de cada punto a la recta simulada con simula_recta().

En éste ejercicio, he generado una muestra de 50 datos definida en el intervalo [-50,50]. También genero un array de etiquetas inicializadas a 0, las cuales toman valor usando el signo de la función $f(x,y) = y - ax - b$. Además obtengo los términos a y b de la recta con la función 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).

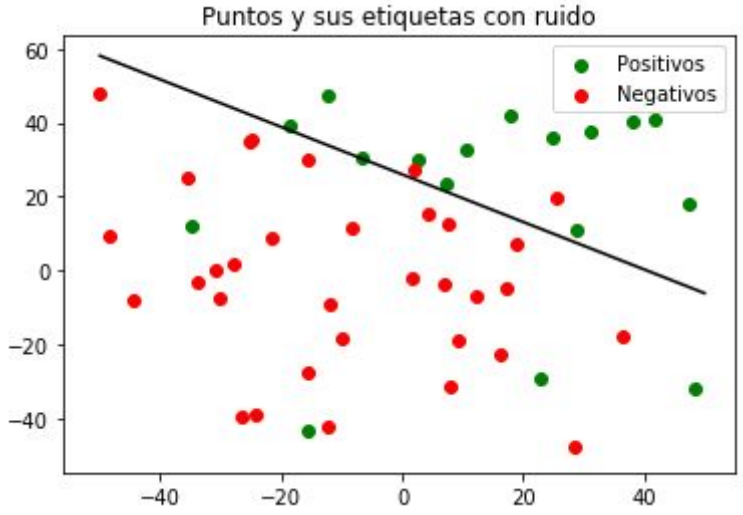
Una vez realizado todo lo anterior, divido los datos según su etiqueta, es decir, divido los datos en dos grupos: positivos y negativos, mediante la función **divide_data**. Para dibujar la gráfica, represento mediante un scatter los datos positivos y negativos separados previamente, y para la recta, considero los puntos x = -50 y x = 50, y obtengo el valor de su coordenada 'y' evaluando dichos puntos en la función f(x,y), siendo $y = ax + b$. Gráfica obtenida:



Como se puede observar en la gráfica, todos los puntos están bien clasificados, puesto que la recta clasifica perfectamente a su izquierda los datos negativos y a su derecha los datos positivos.

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)con las iteraciones.

Para modificar las etiquetas positivas y negativas, primero las divido en dichos subgrupos mediante la función **divide_label**. Una vez separadas, introduzco ruido en el 10% de ellas. Puesto que el 10% de las etiquetas tanto positivas como negativas no salía exacto, he decidido redondear hacia arriba, pues el 10% daba decimal 5. A continuación, concateno las etiquetas, que habían sido separadas previamente, y concateno los datos que habían sido separados en el apartado anterior. Puesto que ciertas etiquetas han sido modificadas, debo separar de nuevo los datos en positivos y negativos, ya que se han producido cambios. Para dibujar la gráfica, he seguido el mismo procedimiento que en el apartado anterior. Gráfica obtenida:



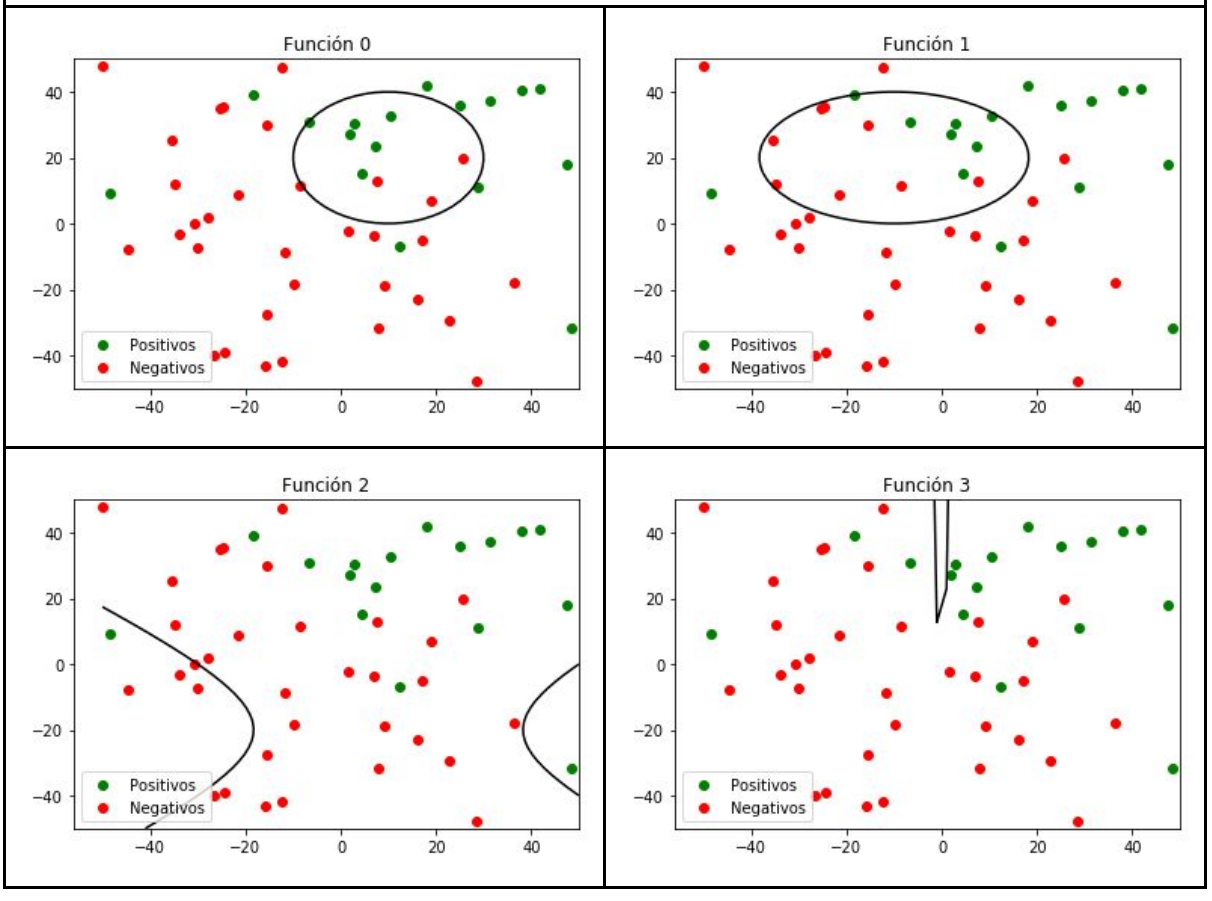
Ahora sí, se observa con facilidad que ciertos puntos no están clasificados, y ellos son los puntos que se corresponden con las etiquetas que hemos modificado anteriormente. En la región positiva, la izquierda, vemos como hay dos puntos negativos, correspondientes con el 10% de los datos positivos (1.5) redondeado a 2, y en la región negativa, vemos que hay cuatro puntos positivos, correspondientes con el 10% de los datos negativos (3.5) redondeado a 4.

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

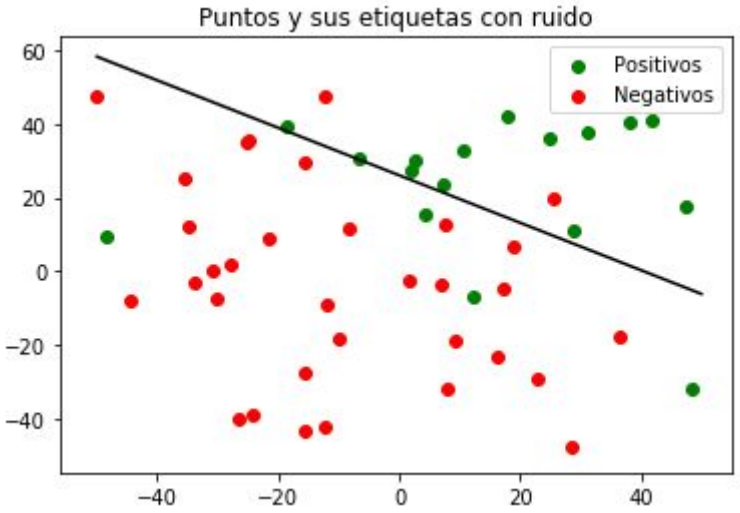
- $f(x,y) = (x-10)^2 + (y-20)^2 - 400$
- $f(x,y) = 0.5(x+10)^2 + (y-20)^2 - 400$
- $f(x,y) = 0.5(x-10)^2 - (y+20)^2 - 400$
- $f(x,y) = y - 20x^2 - 5x + 3$

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.

Frontera de clasificación a partir de las funciones



Incluyo de nuevo la gráfica obtenida anteriormente con la recta, puesto que la anterior no sirve para comparar, ya que he tomado las capturas en ejecuciones diferentes y la modificación de las etiquetas es diferente en cada ejecución:



A la vista de las gráficas, es más que evidente que dichas funciones no son mejores clasificadoras que la función lineal, por lo que el hecho de que sean más complejas no conlleva una mejora en la clasificación. En cuanto a las regiones, es prácticamente imposible distinguir cual es la región positiva y negativa en la clasificación de las funciones, a diferencia de la función lineal en la que sí se aprecian ambas regiones con facilidad. Si nos basamos puramente en las gráficas, concluímos que son peores que la función lineal, pero el motivo teórico está en el conjunto de muestras que se han tratado de clasificar. Éstas funciones ganarían a la función lineal si estuviéramos ante un conjunto de muestras que no fuera linealmente separable, de hecho la función lineal ante tal conjunto de datos daría un resultado de clasificación pésimo. Por lo tanto, para finalizar, concretamos que para éste conjunto de datos linealmente separable, las funciones evaluadas dan unos resultados mucho peores que la función lineal, pero sin olvidar que si el conjunto de datos no fuese linealmente separable, la conclusión sería lo contrario.

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.**

La función ajusta_PLA viene a ser la implementación del algoritmo Perceptrón. Su funcionamiento es el siguiente:
Partimos de un conjunto de datos y unas etiquetas asociadas a ellos. El algoritmo va a tratar de corregir el vector de pesos en busca de la mejor clasificación posible, es decir, converger, por ello, para cada elemento del conjunto de datos predice su etiqueta, si coincide la etiqueta con la predicción, el vector de pesos no se corrige, en cambio si la predicción falla, se corrige el vector de pesos sumándole al actual el producto vectorial del dato y la etiqueta que se acaba de predecir. El algoritmo va iterando indefinidamente hasta cumplir una de las dos condiciones de salida que tiene; o bien alcanza un número máximo de iteraciones, o bien alcanza la clasificación más óptima, la cual ocurre cuando no se produce una modificación del vector de pesos en una iteración con respecto al vector de pesos anterior, y por tanto significa que ha logrado converger.

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.

Éste ha sido el número medio de iteraciones obtenido tanto con inicio 0 como inicio aleatorio:

```
* APARTADO A

Valor medio de iteraciones necesario para converger con vector cero:  44.0
Valor medio de iteraciones necesario para converger con vector aleatorio:  54.6
```

La valoración que tengo tras ver los resultados es que partiendo de un vector de ceros, como es lógico, siempre tarda las mismas iteraciones en converger, en cambio, cuando partimos de un vector aleatorio, las iteraciones necesarias para converger son muy variantes, pese a que la media de iteraciones es mayor que cuando partimos de un vector de ceros, sí que es cierto que en ciertas ejecuciones se logra converger en un número menor de iteraciones, como muestro aquí:

ej1	ej2	ej3	ej4	ej5	ej6	ej7	ej8	ej9	ej10
43	69	79	47	67	35	46	35	60	65

Por lo tanto, la conclusión que obtengo es que tal y como vemos en la fotografía anterior, la diferencia de iteraciones entre ambas no es muy significativa, por lo tanto a la hora de decantarme por inicializar el vector a ceros o aleatorio, lo que me plantearía es en cuántas ejecuciones voy a usar el algoritmo, y en función de ello, decidiría, ya que si son pocas ejecuciones, quizá conviene probar con el vector aleatorio y si se da el caso, converger en menos iteraciones, o en el peor de los casos converger en algo más de iteraciones, mientras que si son muchas ejecuciones, iría a lo seguro y consideraría el vector de pesos inicial con ceros, pues garantiza un consumo de recursos menor y un número de iteraciones necesarias para converger menor con mucha probabilidad.

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.

Éste ha sido el número medio de iteraciones obtenido tanto con inicio 0 como inicio aleatorio:

```
* APARTADO B

Valor medio de iteraciones necesario para converger con vector cero:  500.0
Valor medio de iteraciones necesario para converger con vector aleatorio:  500.0
```

Se observa un comportamiento diferente, ya que en ambos casos el valor medio de iteraciones obtenidas ha sido el máximo, en éste caso 500. También probé con más iteraciones por si 500 no eran suficientes para converger, pero seguía saliendo igual. Las razón principal por la que ocurre ésto es porque el algoritmo Perceptrón funciona con datos linealmente

separables, y en éste apartado como el conjunto de datos y etiquetas que consideramos tiene ruido aplicado, tenemos unos datos y etiquetas que no son linealmente separables por lo tanto el Perceptrón por muchas iteraciones que le pongamos como tope, nunca va a converger.

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] [0, 2] con probabilidad uniforme de elegir cada x ε 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 $\{X_n\}$ de X y evaluar las respuestas $\{y_n\}$ de todos ellos respecto de la frontera elegida.**

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

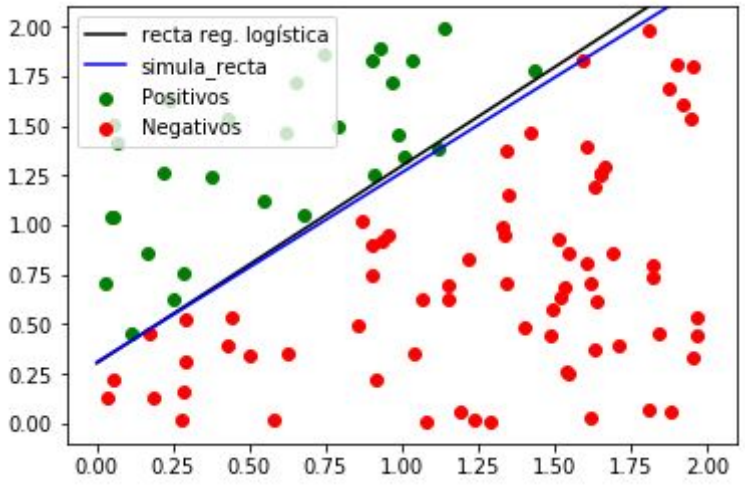
- **Inicializar el vector de pesos con valores 0.**
- **Parar el algoritmo cuando $\|w^{(t-1)} - w^{(t)}\|$ < 0.01, donde w(t) denota el vector de pesos al final de la época t. Una época es un pase completo a través de los N datos.**
- **Aplicar una permutación aleatoria, 1,2,….,N, en el orden de los datos antes de usarlos en cada época del algoritmo.**
- **Usar una tasa de aprendizaje de = 0.01**

Para implementar dicho algoritmo, he reutilizado mi algoritmo de Gradiente Descendiente Estocástico y le he añadido algunas modificaciones. El nuevo algoritmo parte de un vector de pesos inicializado a 0, en el que en cada época va a ir actualizando el vector de pesos en busca de la convergencia. Esa actualización se lleva a cabo mediante el cálculo del Gradiente. En cada época, se realiza una permutación aleatoria sobre los datos antes de ser usados en minibatches y tras ello se acumula el Gradiente de cada elemento del minibatch. Una vez finalizado, se actualiza el vector de pesos con la sumatoria de los Gradientes obtenidos para ese minibatch y el learning rate. Tras ésto, se comprueba si la norma vectorial entre el vector de pesos recién actualizado y el vector de pesos anterior a la actualización es menor que 0.01. Si se da el caso, nos encontramos ante una condición de parada en la que el algoritmo devuelve el vector de pesos y el número de iteraciones. La

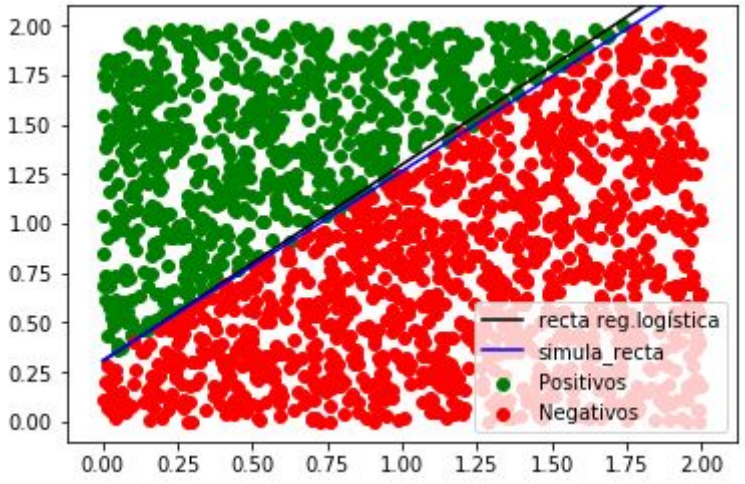
otra condición de parada sería finalizar por alcanzar un número máximo de iteraciones.

b) Usar la muestra de datos etiquetada para encontrar nuestra solución g y estimar Eout usando para ello un número suficientemente grande de nuevas muestras (>999).

En primer lugar para encontrar la solución g, genero una muestra de 100 elementos y realizo el proceso de etiquetado comentado anteriormente. Una vez hecho, llamo al algoritmo de Regresión Logística y obtengo la solución. A continuación muestro una gráfica con la representación de la solución clasificando las 100 muestras:



Como podemos ver la solución obtenida (recta negra) frente a la recta frontera (recta azul) no difiere apenas, lo que indica que la solución obtenida es bastante acertada pues clasifica los datos con un bajo error. Una vez obtenida la solución, genero 2000 nuevas muestras sobre las que aplico la solución anterior, obteniendo la gráfica:



Nuevamente, vemos que la solución clasifica de forma muy aproximada los datos, obteniendo una clasificación similar a la dada por la recta frontera. Los datos que no llega a clasificar con exactitud son debido a que se encuentran en el límite de la frontera y es excesivamente complejo determinar en qué lado de la recta quedan. Tras ver el amplio porcentaje de acierto obtenido para 2000 muestras, estimo el valor de Eout, aunque a la vista de la gráfica debe ser bajo.

```
Error estimado para 2000 muestras:  0.1376857115801278
```

Ese ha sido el valor de Eout estimado para 2000 muestras. Como ya decía anteriormente, debía ser bajo y así ha sido. Además si lo comparamos con el valor de Eout estimado para las 100 muestras iniciales:

```
Error estimado para 100 muestras:  0.12118573058973031
```

podemos llegar a la conclusión de que la solución obtenida por el algoritmo de Regresión Logística se comporta de forma excelente y muy similar ante nuevas muestras suficientemente grandes, ya que el incremento en el valor estimado de Eout ha sido mínimo, dando por hecho que la solución dada clasifica los datos de forma muy acertada.

3. BONUS

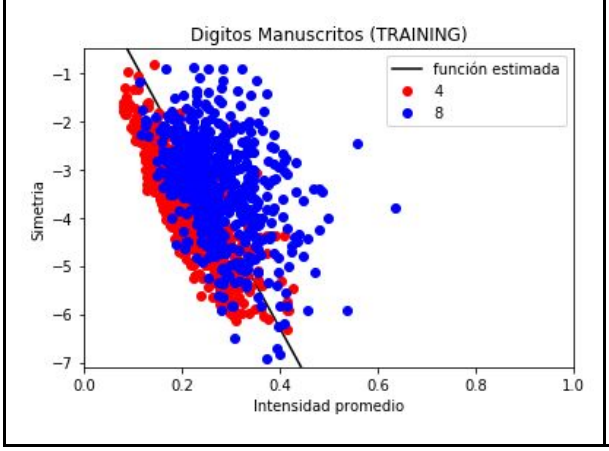
1. Clasificación de Dígitos. **Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 4 y 8. Usar los ficheros de entrenamiento (training) y test que se proporcionan. Extraer las características de intensidad promedio y simetría en la manera que se indicó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

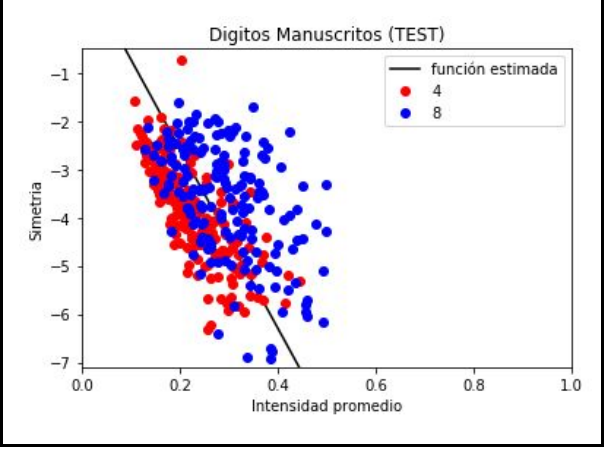
1. Plantear un problema de clasificación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g.
2. Usar un modelo de Regresión Lineal y aplicar PLA-Pocket como mejora. Responder a las siguientes cuestiones.

Para elegir el modelo de Regresión Lineal me he basado en la rapidez en obtener la solución, por lo que me he decantado por el algoritmo de la Pseudo-Inversa. La necesidad del algoritmo PLA_Pocket surge porque como mostraré a continuación en las gráficas, los datos que estamos considerando no son linealmente separables, por lo que el algoritmo PLA resultaría ineficaz. PLA_Pocket trata de aproximarse a una solución antes la dificultad de separar los datos, de la forma que guarda la mejor solución obtenida hasta el momento y va comparando las soluciones siguientes con la mejor almacenada. De esa forma, sabemos con certeza que el algoritmo nos va a devolver la solución más óptima pues siempre almacena la mejor y no la modifica hasta no encontrar otra mejor, en una iteración futura. Como estamos tratando de observar la mejora que produce el algoritmo PLA_Pocket, le paso como vector de pesos inicial, la solución obtenida con el algoritmo de la Pseudo-Inversa, para que parta de una solución aproximada y trate de mejorarla.

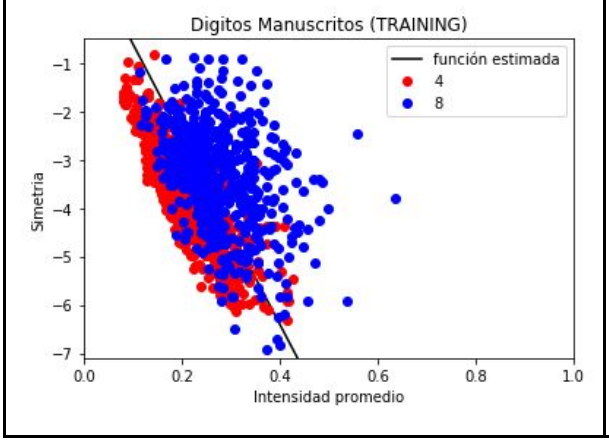
a) Generar gráficos separados (en color) de los datos de entrenamiento y test junto con la función estimada.

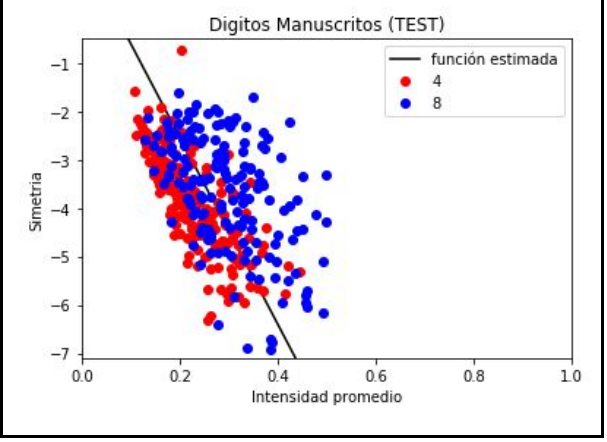
PSEUDO_INVERSA





PLA_POCKET





Como se puede apreciar, la mejora es prácticamente irrelevante puesto que nos encontramos ante unos datos que presentan bastante dificultad para ser clasificados. La escasa mejoría se puede apreciar comparando los extremos de los datos por donde la recta divide, viendo que ciertos puntos que con el algoritmo de la Pseudo-Inversa quedaban en la recta, ahora quedan clasificados a un lado de ella.

b) Calcular Ein y Etest (error sobre los datos de test).

A continuación muestro los errores obtenidos tanto con el algoritmo de la Pseudo-Inversa como con el algoritmo de PLA-Pocket:

```
* APARTADO B

Error obtenido para el Train con Pseudo-Inversa:  0.22780569514237856

Error obtenido para el Test con Pseudo-Inversa:  0.25136612021857924

Error obtenido para el Train con Pocket:  0.22529313232830822

Error obtenido para el Test con Pocket:  0.2540983606557377
```

Si bien es cierto que para el conjunto de datos Train, el algoritmo PLA-Pocket reduce ligeramente el error, para el conjunto de datos Test no. Hay que tener en cuenta que los datos con los que estamos trabajando presentan cierta dificultad, además el tamaño del conjunto de datos Test es bastante inferior al conjunto de datos Train. En definitiva, para nuestro caso, la mejora no es muy relevante.

c) Obtener cotas sobre el verdadero valor de Eout. Pueden calcularse dos cotas una basada en Ein y otra basada en Etest. Usar una tolerancia = 0.05. ¿Que cota es mejor?

```
* APARTADO C


Cota obtenida para el Train con Pocket:
0.6562296377985837

Cota obtenida para el Test con Pocket:
0.9809354817457594
```

Obviamente, a la vista de los resultados, la cota obtenida para el conjunto Train, es decir basada en Ein es mejor que la cota basada en Etest, ya que es menor. El motivo de ello está en el tamaño de ambos conjuntos de datos, ya que si los elementos que componen la fórmula de la cota son los mismos a excepción del tamaño del conjunto de datos. En ambos la

dimensión VC es 3 y la tolerancia es 0.05, pero como ya he dicho, el tamaño no. En nuestro caso, el tamaño del conjunto de datos Train es 3-4 veces mayor que el conjunto de datos Test, lo que provoca que sí o sí, la cota obtenida para el conjunto Test sea superior pues el hecho de que el denominador (tamaño del conjunto) sea menor, hace que el resultado de la división sea mayor. Por lo tanto, para nuestro caso, la cota para el conjunto Train es mejor, pero si ambas cotas fueran calculadas sobre conjuntos con tamaños similares, no habría una cota notablemente mejor que otra, dando lugar a que la tasa de error fuera de la muestra tendría un valor similar a la tasa de error dentro de ella.