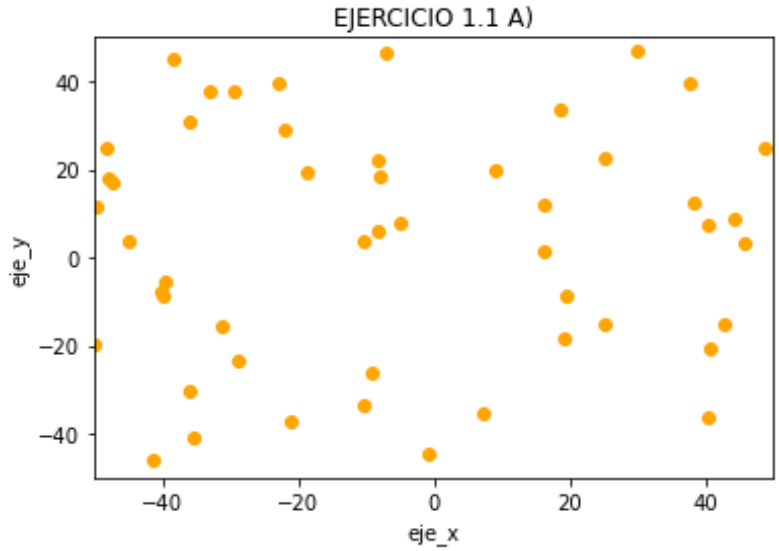
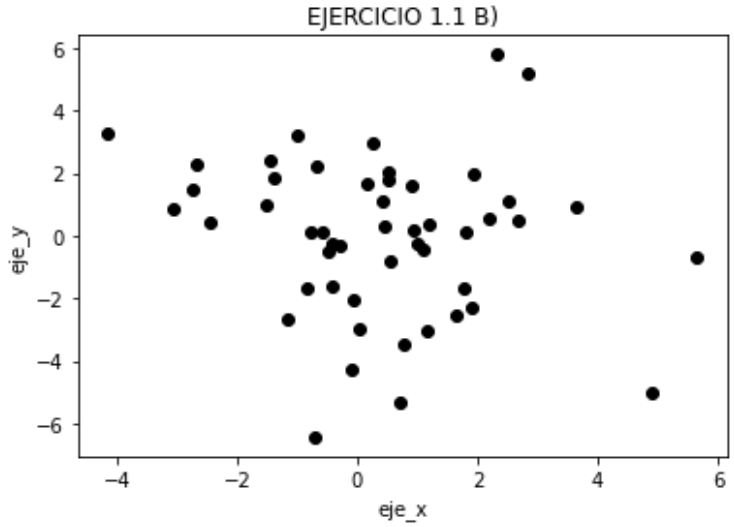
1. EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO

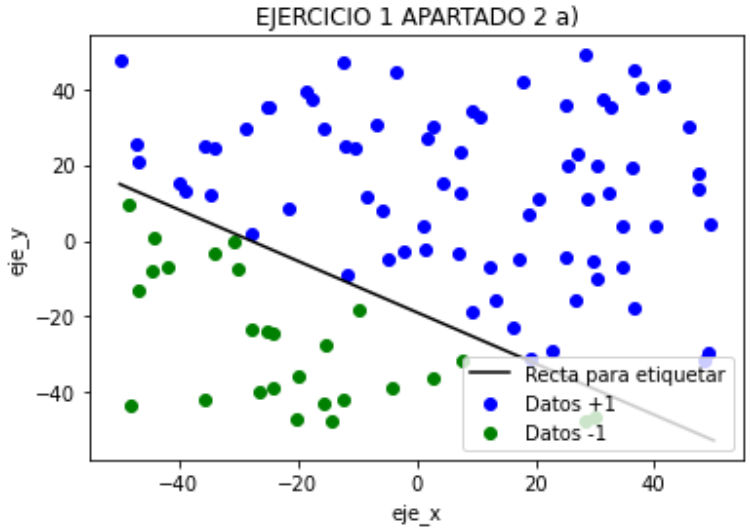
1. **Dibujar gráficas con las nubes de puntos simuladas con las siguientes condiciones:**
   a) **Considere N = 50, dim = 2, rango = [-50,+50] con simula_unif(N,dim,rango)**



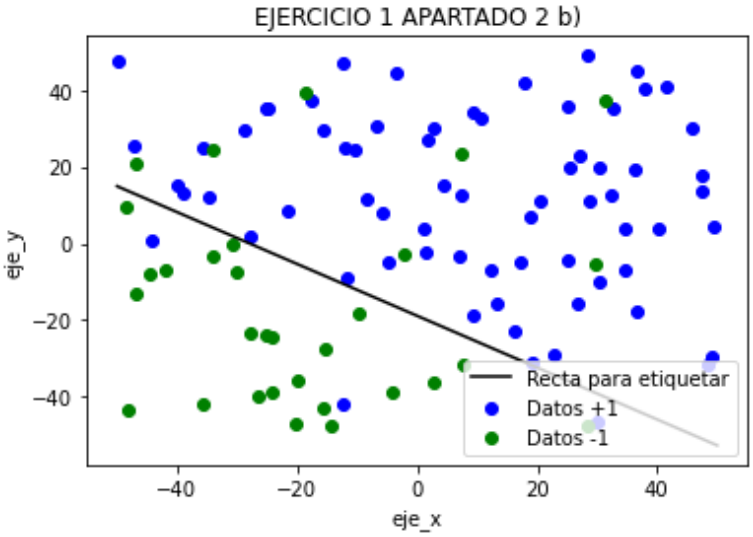
   b) **Considere N = 50, dim = 2, sigma = [5,7] con simula_gaus(N,dim,sigma)**



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 a 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 sumla_recta().**
   a. **Dibujar un gráfico 2D donde los puntos muestren el resultado de su etiqueta. Dibuje también la recta usada para etiquetar.**



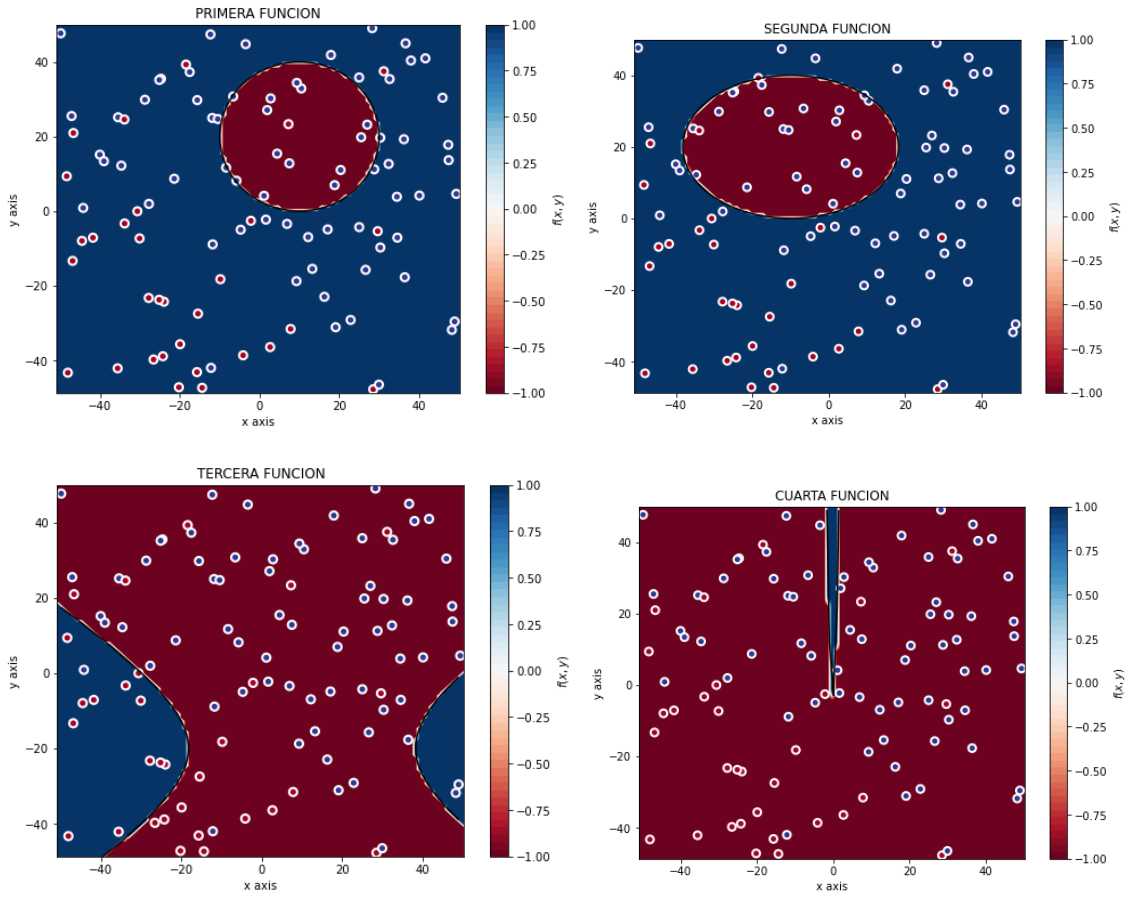
b. Modifique de forma aleatoria un 10% de las etiquetas positivas y otro 10% de las negativas y guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior.



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

- **$f(x,y) = (x - 10)^2 + (y - 20)^2 – 400$**
- **$f(x,y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$**
- **$f(x,y) = 0,5(x - 10)^2 + (y + 20)^2 - 400$**
- **$f(x,y) = y – 20x^2 – 5x + 3$**

Visualiza el etiquetado generado en 2b junto con cada una de las gráficas 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.

En el caso de la recta, las partes positiva y negativa obtenidas estaban separadas de forma inconfundible en dos partes que se acogían fielmente a la división de las etiquetas. Sí que es cierto que, a raíz del apartado b, la inclusión de ruido ha hecho que el ajuste de la recta no sea 100% correcto, pero al haber hecho la división de datos a raíz de esta, difícilmente encontremos una función no lineal que divida el conjunto con más exactitud que la primera. En el caso del resto de funciones, la parte negativa y positiva viene delineada por una curva que permite más flexibilidad a la hora de realizar divisiones de datos y facilitan la posibilidad de que el ajuste sea mejor. En dos de las cuatro funciones, la parte negativa es un área limitada por la parte positiva que la rodea, resultando en que la mayoría del espacio sea positivo; al contrario de lo que ocurre en las otras dos en las que, a pesar de que la parte positiva no está del todo rodeada por zona negativa, esta abarca mucho más espacio.

En cuanto a la calidad como clasificadoras, estas cuatro nuevas funciones no obtienen mucho mejores resultados. Como comentaba anteriormente, la recta se generaba antes que los datos y, a partir de esta, se asignaban etiquetas a

los puntos según el signo de la distancia de estos a la recta, por lo que el conjunto de datos quedará perfectamente separado con respecto de la recta. Es cierto que el hecho de introducir ruido en el 10% de cada conjunto de datos hace que no sea una división perfecta, pero a simple vista vemos que el resto de ajustes no consigue una separación precisa.
Ese detalle se hace visible en la primera función con ver la posición en la que aparece el área de datos negativos. Esta se encuentra más cercana a la esquina superior derecha, que es totalmente opuesta a la zona original en la que se encuentra el conjunto de datos negativos. Sí que es posible que con el cambio de etiqueta haya caído algún valor negativo en esta parte de la gráfica, pero realmente está bastante alejado de donde se prevén obtener los datos negativos.
En el caso de la segunda función, a pesar de tener esa zona negativa algo más desplazada a la izquierda, la cantidad de puntos negativos que recoge son 2 de los más de 30 que hay, por lo que el porcentaje de acierto no supera el 7%.
En la tercera función vemos como la parte negativa se expande por la gran mayoría del mapa cuando de entrada ya hay menos puntos negativos que positivos, por lo que, por probabilidad, es casi seguro que se estén abarcando puntos que no pertenecen a los correctos. A esto se le suma que la parte positiva generada se mantiene en la parte inferior de la tabla donde los únicos valores positivos que podían existir ahí eran debidos al ruido.
Para la cuarta función no hay mucho que decir. La parte positiva no recoge ni un solo punto positivo de forma unívoca por lo que no es solo mal ajuste, sino que es el peor de los 5 probados.

Si con la modificación de las etiquetas nos referimos la asignación original que dividía el conjunto de datos en dos grupos según su posición con respecto a una recta, diremos que este proceso es determinante a la hora de elegir qué función realizar el mejor ajuste. Si la asignación de etiquetas se lleva a cabo teniendo como hiperplano de referencia una recta que convierte en el conjunto de datos en separable, no existe mejor función que la de una recta para obtener el ajuste ideal, sin la necesidad de recurrir a funciones complejas.

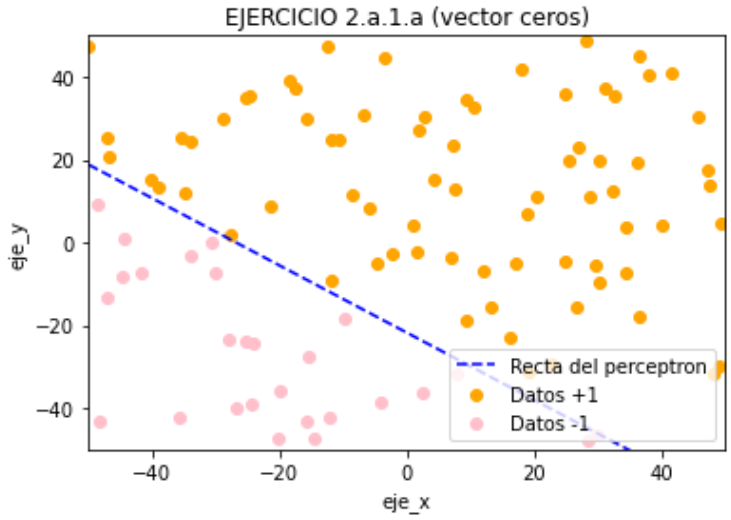
Sin embargo, si la modificación de las etiquetas hace alusión a la introducción de ruido, la influencia no es fácilmente determinable. Si es cierto que una consecuencia clara es la adición de dificultad al problema, pues una muestra separable se puede convertir en una que no lo es, pero si la alteración de las etiquetas se lleva a cabo de forma aleatoria no sabemos con certeza qué tan compleja puede ser la función que necesitemos para la división actual. A pesar de provocar resultados complejos, es una práctica necesaria para simular situaciones del mundo real en las que los datos no siempre son separables o la clasificación no es determinista, sino probabilística.

2. MODELOS LINEALES

1. Algoritmo Perceptrón

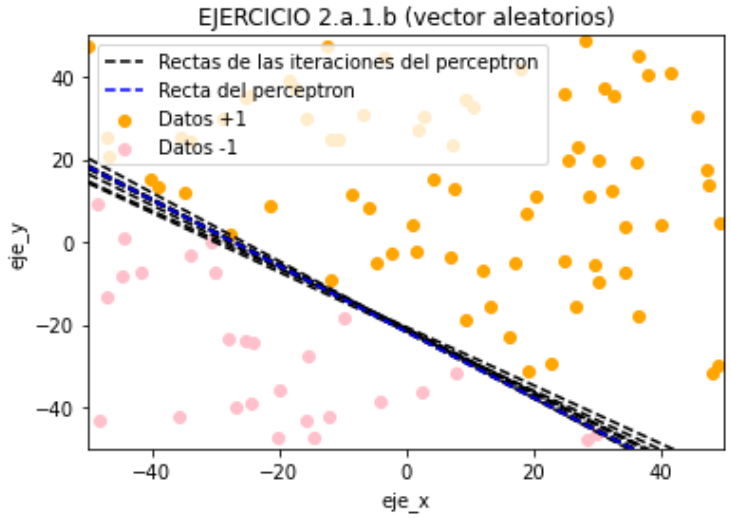
a. Ejecutar el algoritmo PLA con los datos simulados en los apartados 2a de la sección 1. Anotar el número medio de iteraciones necesarias inicializando el algoritmo con:

i. El vector cero




```
Numero 'medio' de iteraciones para el vector 0:  61
```

ii. Vectores de números aleatorios en [0,1] (10 veces)




```
Valor medio de iteraciones necesario para converger: 67.4
```

Valorar el resultado relacionando el punto de inicio con el número de iteraciones.

Como podíamos suponer, en ambos casos se llega a una solución correcta en no muchas iteraciones. Para el caso en el que se hace una sola vez (por eso pone 'media' entre comillas) obtenemos que en 61 iteraciones se encuentra una recta que divide el conjunto correctamente respetando las dos clasificaciones. Sin embargo, en el otro caso los puntos son

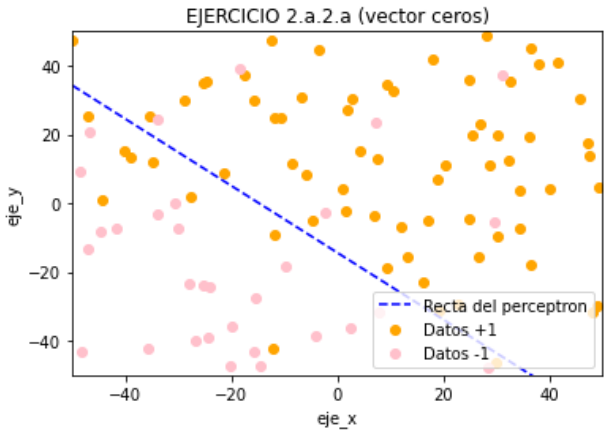
aleatorios, lo cual podría conducir a un aumento en el número de iteraciones necesarias, pero al ser un conjunto perfectamente separable termina convergiendo en una media de iteraciones bastante similar.

Podemos concluir en que al no haber ruido y ser un conjunto separable por un hiperplano formado por una recta, el perceptrón siempre encontrará solución y en un número de iteraciones similar. Si que es cierto que existen casos en los que el número de iteraciones puede dispararse debido a que la recta de la que se parte esté muy alejada de la correcta y tenga que hacer muchas correcciones intermedias, pero son casos aislados. Un ejemplo sería que en la tercera ejecución el número de iteraciones supera el centenar, con una w inicial con los valores [0.5737319, 0.63905897, 0.4896996], mientras que la ejecución que menos iteraciones necesita es la penúltima con 56, partiendo de w = [0.30557161, 0.48432156, 0.65005358], que está más cercano a [0, 0, 0].

Es más, para este ejercicio, la etiquetación se ha realizado a partir de la recta con ecuación y = -0.677x - 18.89, la cuál es la más acertada de todas. Partiendo del vector de pesos [0,0,0], la recta obtenida es y = -0.972x - 14.32; partiendo de [0.5737319, 0.63905897, 0.4896996] (ejecución con más iteraciones) obtenemos esta ecuación y = -0.756 - 21.29; y para el punto de partida [0.30557161, 0.48432156, 0.65005358] la recta a la que llegamos es y = -0.795 -21.67

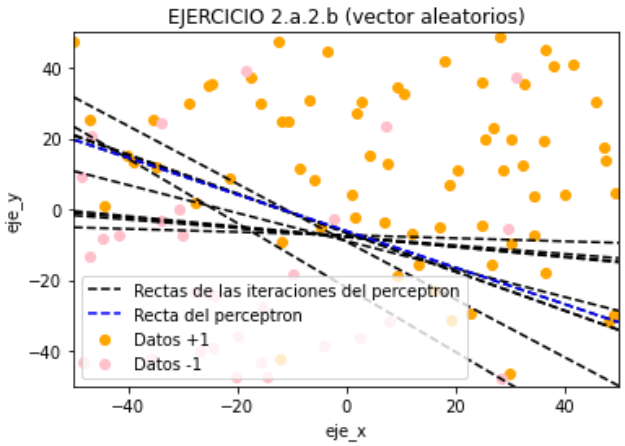
b. Hacer lo mismo que antes usando ahora los datos del apartado 2b de la sección 1.

i. El vector cero




```
Numero 'medio' de iterationes para el vector 0:  1000
```

ii. Vectores de números aleatorios en [0,1] (10 veces)




```
Valor medio de iteraciones necesario para converger (vector inicial aleatorio): 1000.0
```

¿Observa algún comportamiento diferente? En caso afirmativo diga cual y las razones para que ello ocurra.

En este caso, el conjunto de datos es el mismo, pero a un 10% de cada conjunto se le ha alterado la etiqueta para meter ruido. De esta forma, un modelo linear seguramente no pueda definir una recta que divida el conjunto sin error, ya que este podría no ser separable.

El comportamiento diferente es que tanto si se ejecuta una vez partiendo de [0,0,0], como si se ejecutan 10 empezando por w aleatorias, llegan al número máximo de iteraciones permitidas sin haber convergido. Esto se debe a que el perceptrón comprueba elemento a elemento si se encuentra en el lado correcto del hiperplano o no, y en caso negativo reajusta este para que abarque el dato que estamos evaluando en el momento, sin considerar los ya ordenados o los que pueda estar desubicando. De esta forma, al haber ruido y puntos entremezclados, al arreglar unos, estropea otros y viceversa, lo que le impide hacer una iteración completa sobre todo el conjunto de datos sin encontrar al menos un error (condición de parada).

Además, observando las gráficas se puede ver que, aun realizando todas las iteraciones posibles, no se llega a una solución perfecta, pues el ruido se mantiene al terminar de ajustar los datos durante 1000 vueltas al conjunto de datos.

En la representación de las gráficas referidas a las 10 ejecuciones con puntos de partida aleatorios podemos ver unas rectas negras que representan las rectas obtenidas en cada iteración que se ha realizado y una azul referida a aquella final por la que nos decantamos.

En el primer caso en el que no había ruido, se ve que el movimiento de la recta es muy leve hasta alcanzar la recta ideal, mientras que, para aquella con ruido, en cada iteración la recta parece cambiar lo suficiente como para intuir que no se está acercando hacia una correcta, sino que está oscilando. Esto último no se puede saber al 100% porque, a pesar de poder ver las rectas, no se ve el orden de estas y no sabemos si se está acercando a la final, pero teniendo en cuenta que:

- No alcanza una solución correcta

- Arreglar un punto puede dañar más de uno

podemos dar por hecho que el avance no va conducido hacia una mejora continua.

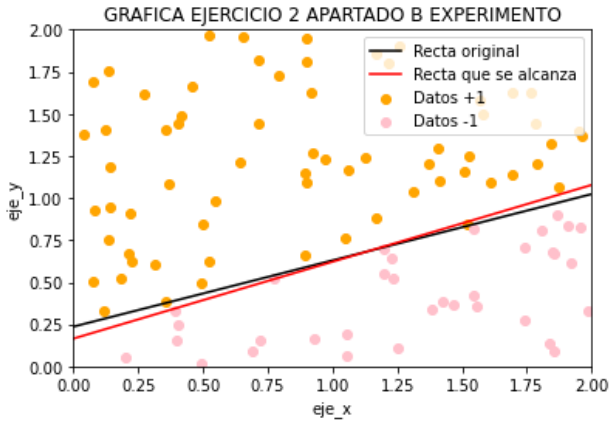
2. **Regresión Logística: Seleccionamos una muestra de datos aleatorios con N=100 y evaluamos las respuestas de todos ellos respecto de la frontera elegida, que es una recta simulada en este ejercicio en el rango [0,2]. Tras esto ejecutamos regresión logística para encontrar la función solución g y evaluamos el Eout usando para ello una muestra grande con más de 999 elementos. Todo esto se repite 100 veces.**

   En este ejercicio se ha realizado el experimento una primera vez para la obtención de gráficas y datos orientativos y, tras esto, se ha repetido 100 veces obteniendo de estos la media de Eout y las épocas necesarias para cada conjunto aleatorio de N=100 generado para la interpretación de resultados.

   El Eout obtenido para una primera muestra ha sido 0.1033

```
Eout:  0.10335505452619224
```

   Y esta es la gráfica obtenida utilizando Regresión Logística sobre ese conjunto de datos



   Tras 100 ejecuciones, la media del error y del número de épocas necesarias para cada muestra queda:

```
Error medio de las 100 ejecuciones: 0.1071640723502017
Epocas medias de las 100 ejecuciones: 355.37
```

   Como podemos ver, tanto para la primera ejecución como para la media tras 100 iteraciones, el error es ≈ 0.1, lo cual es un valor considerablemente bajo.

3. BONUS

Clasificación de dígitos. Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 4 y 8.

1. **Plantear un problema de clasificación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g.**

Para este problema de clasificación partimos de un conjunto de números manuscritos de los que tenemos que diferenciar los dígitos 4 y 8. Los datos son extraídos de los ficheros de entrenamiento y test proporcionados y las características a tener en cuenta para reconocer el número son la intensidad promedio y la simetría. La intensidad se refiere a la media de grises, es decir, la cantidad de píxeles blancos y negros que hay calculando su media y la simetría es con respecto del centro de la imagen en vertical.

Los valores que pueden recibir las etiquetas son -1 en el caso de leer un 4 y 1 en el caso de leer un 8, y se usará la Regresión Lineal y el PLA-Pocket para calcular la función, los errores y las cotas.

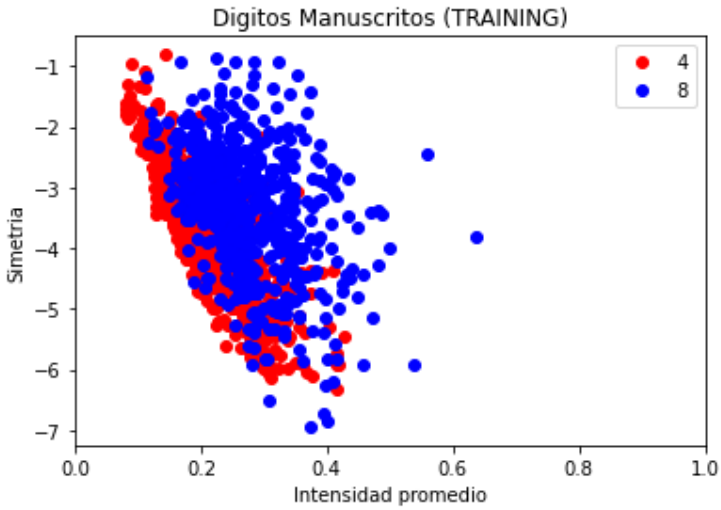
En el caso de Regresión Lineal el algoritmo seleccionado es el de pseudoinversa utilizado en la práctica anterior que recibe un conjunto de datos x y un conjunto de etiquetas y. Como veremos en los resultados, este algoritmo resulta ser lo suficientemente bueno como para que la diferencia con PLA-Pocket pueda ser insignificante.

Por otra parte, el algoritmo de PLA-Pocket es una variable del perceptrón, pero con memoria. Esto se debe a que cada vez que realiza una iteración completa por todo el conjunto de datos calcula el error y, en el caso de ser mejor que el que ya teníamos, actualiza este valor y guarda la w con la que se obtiene como solución temporal. Si al terminar no se encuentra uno mejor, se habrán ignorado aquellos cambios que empeoraban lo que teníamos hasta ahora.

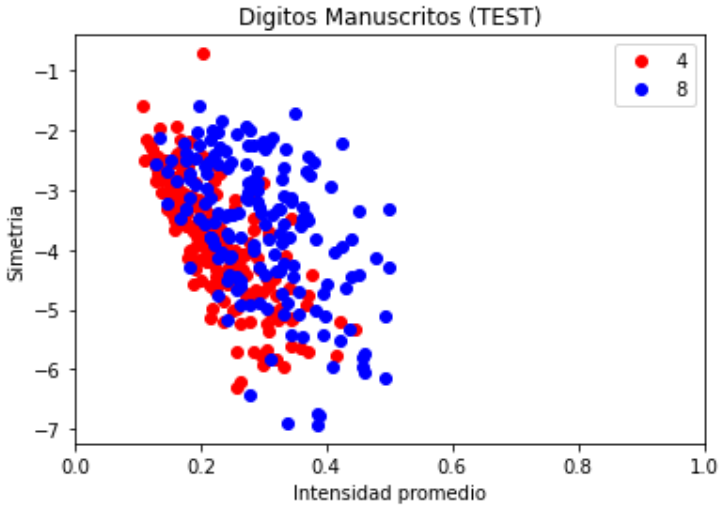
Las entradas de este algoritmo son un conjunto de datos con su conjunto de etiquetas respectivo, una w de partida, un número máximo de iteraciones y un umbral con el que parar si encuentro una solución lo suficientemente buena como para no interesarme el resto de búsqueda y, como salida, esta función devuelve el mejor w obtenido en el proceso.

El siguiente paso en el estudio sería calcular el error obtenido con cada algoritmo, la gráfica que muestre de forma gráfica el ajuste y unas cotas de los errores.

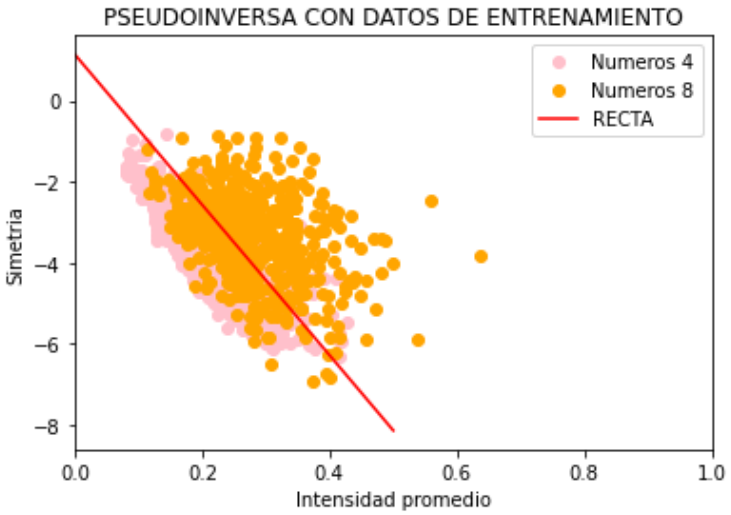
2. **Usar un modelo de Regresión Lineal y aplicar PLA-Pocket como mejora. Responder a las siguientes cuestiones.**
    a. **Generar gráficos separados de los datos de entrenamiento y test junto la función estimada**
        i. **Datos de entrenamiento separados por color**



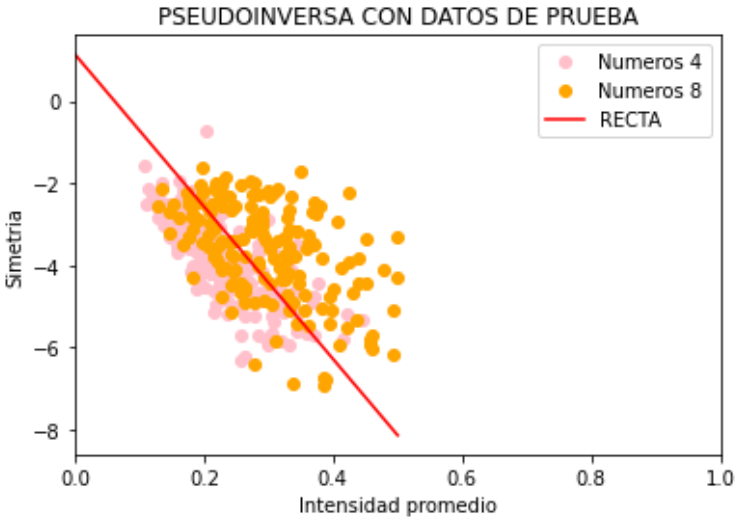
i. **Datos de test separados por color**



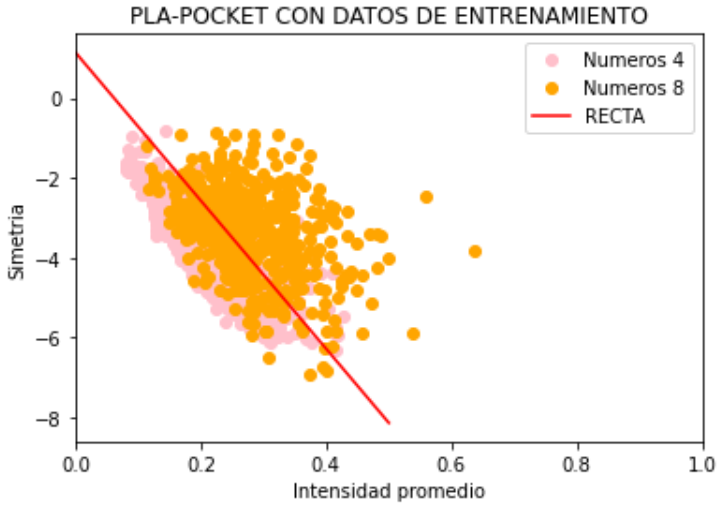
i. **Comportamiento de Regresión Lineal para los datos de train**



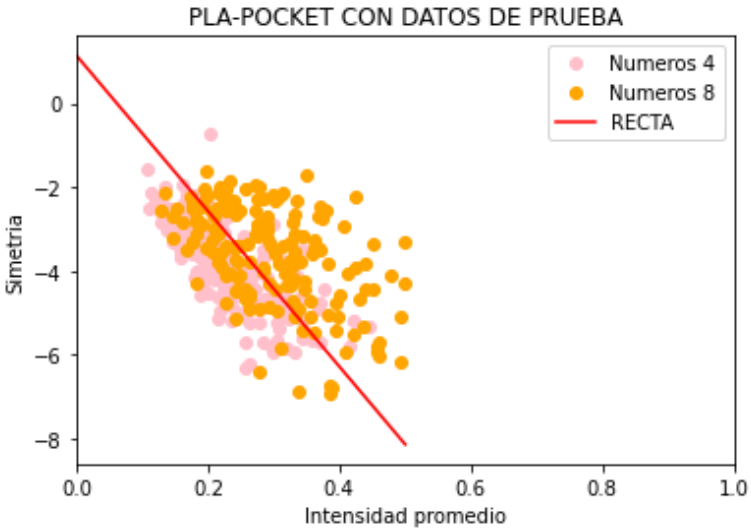
i. **Comportamiento de Regresión Lineal para los datos de test**



i. Comportamiento de PLA-Pocket para los datos de train



i. Comportamiento de PLA-Pocket para los datos de test



b. Calcular Ein y Eout

```
\PARA REGRESION LINEAL

Error dentro de la muestra con el modelo de Regresion Lineal pseudoinversa:  0.22780569514237856
Error fuera de la muestra con el modelo de Regresion Lineal pseudoinversa:  0.25136612021857924


--- Pulsar tecla para continuar al algoritmo PLA-Pocket---


\PARA PLA POCKET

Error dentro de la muestra con PLA-Pocket:  0.22529313232830822
Error fuera de la muestra con PLA-Pocket:  0.2540983606557377
```

Aunque existe alguna diferencia entre un algoritmo y otro, podemos ver que el error obtenido es realmente similar. Esto se puede deber a que el w inicial que recibe el algoritmo de PLA-Pocket es aquel al que llega pseudoinversa como mejor solución, por tanto, el primero de los dos se va a limitar a intentar mejorarlo y, al menos dentro de la muestra, es muy probable que obtenga un error menor. La solución de pseudoinversa está cercana a la correcta y por ello no se puede encontrar una que reduzca tanto el error.

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

```
OBTENCION DE COTAS PARA Ein y Etest

Cota del error dentro de la muestra con pseudoinversa:  0.6587422006130822
Cota del error fuera de la muestra con pseudoinversa:  0.9782032413373529
Cota del error dentro de la muestra con PLA-POCKET:  0.6562296377990118
Cota del error fuera de la muestra con PLA-POCKET:  0.9809354817745115
```

La cota se refiere a la longitud del intervalo en el que puede encontrarse el valor exacto del error. Cuanto mayor sea el valor, mayor será el intervalo del que obtener el valor exacto y menos preciso será. Es por ello que los mejores valores son las cotas dentro de la muestra para ambos algoritmos, mientras que para los errores fuera de esta, las cotas son peores.

Esto se puede deber a que el tamaño de las muestras en las que se calcula el Eout, el x_test e y_test, son mayores que las que se usan para el Ein, x_train e y_train, por lo que precisar un error es más compejo.