Aprendizaje Automático

Práctica 2

Luis Antonio Ortega Andrés

26 de abril de 2020

1. Ejercicio sobre la complejidad de $\mathcal{H}$ y el ruido

En este ejercicio vamos a aprender la dificultad que introduce la aparición de ruido en las etiquetas a la hora de elegir la clase de funciones más adecuada para nuestro problema.

Para ello comenzamos utilizando las tres funciones auxiliares que se nos dan programadas, a saber:

- `simula_unif(N, dim, rango)`
- `simula_gaus(N, dim, sigma)`
- `simula_recta(intervalo)`

para generar conjuntos (o nubes) de puntos sobre los que realizar nuestro estudio.

Vemos entonces dos ejemplos de ejecución de estas funciones, concretamente `simula_unif` y `simula_gaus`. Ambas nos devuelven un conjunto de puntos generado de forma aleatoria, sin embargo, la primera de estas lo hace siguiendo una distribución uniforme y la segunda distribuciones normales centrales de varianza la indicada en el argumento.

Concretamente generaremos 50 puntos en $[-50, 50] \times [-50, 50]$ en el caso de la uniforme y en el caso de la normal, la primera coordenada generada por una normal $N(0, 5)$ y la segunda por una $N(0, 7)$. Veamos los puntos obtenidos.

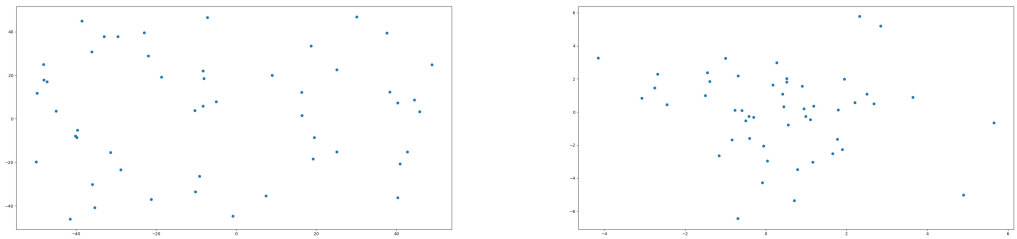

Figura 1: `simula_unif(50, 2, [-50,50])` y `simula_gaus(50, 2, [5,7])`

Como cabría esperar, los resultados obtenidos con la distribución normal son mas centrales que los obtenidos con la distribución uniforme.

Ahora vamos a valorar la influencia del ruido en la selección de la complejidad de la clase de funciones, para ello, utilizamos la función `simula_unif` para generar una muestra de puntos en $[-50, 50] \times [-50, 50]$, tras ello utilizaremos la función `simula_recta` para generar los parámetros a, b reales de una recta que pase por dicho cuadrante. En concreto será la recta del plano

$$y = ax + b$$

La utilizaremos para definir una función de etiquetado

$$\begin{aligned} f : [-50, 50]^2 &\to \{0, 1\} \\ (x, y) &\mapsto \text{signo}(y - ax - b) \end{aligned}$$

En la figura 2 podemos ver la nube de puntos junto con la recta que los separa.

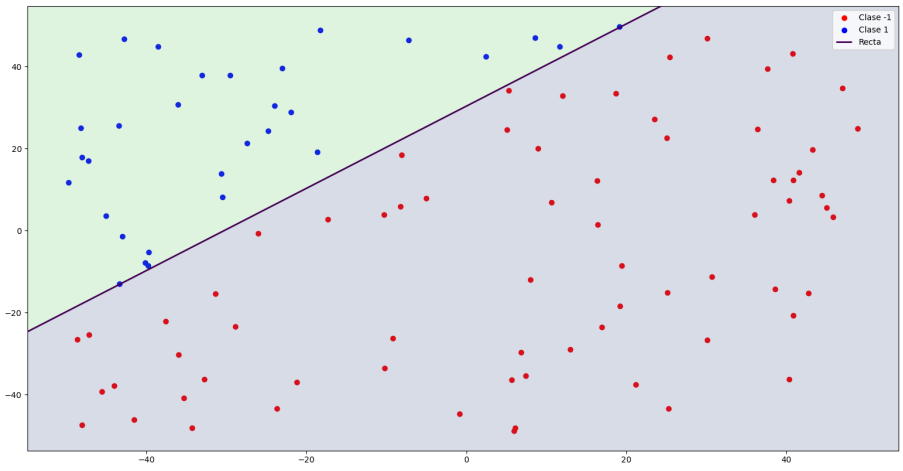

Figura 2: Nube de puntos etiquetada con la función `f(x,y)` y recta correspondiente.

Modificamos entonces un $10\,\%$ del conjunto de etiquetas de cada clase y mostramos de nuevo la correspondiente nube de puntos. Podemos ver los resultados en la figura 3.

Notamos que la proporción de elementos de cada clase no esta equilibrada, existen más elementos pertenecientes a la clase -1, esto será importante a la hora de comparar clasificadores, en particular el número de puntos de la clase -1 es 69, siendo los 31 restantes de la clase 1.

Ahora buscamos comprobar como se comportan modelos de clasificación mas complejos ante nuestros datos con ruido. Para ello tenemos en cuanta los siguientes podemos.

- **Elipse 1** $\to f_1(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
- **Elipse 2** $\to f_2(x, y) = 0{,}5(x + 10)^2 + (y - 20)^2 - 400$
- **Hipérbola** $\to f_3(x, y) = 0{,}5(x - 10)^2 - (y + 20)^2 - 400$
- **Parábola** $\to f_4(x, y) = y - 20x^2 - 5x + 3$

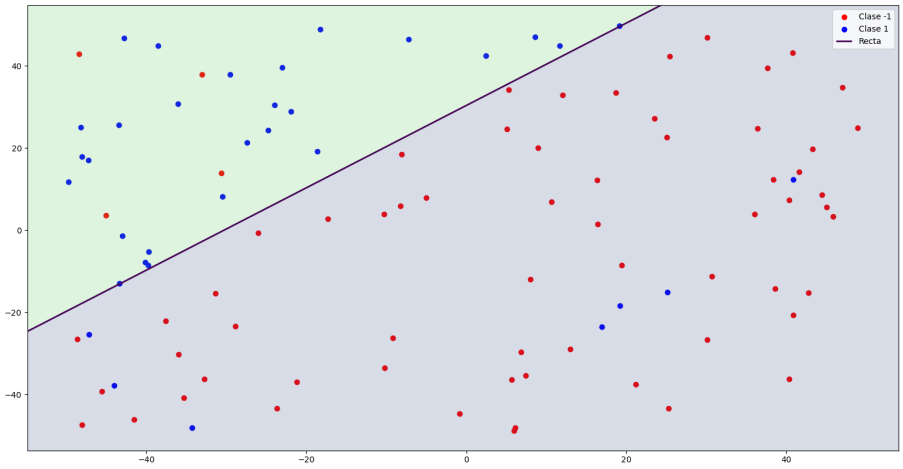

Figura 3: Nube de puntos con 10 % de ruido añadido.

Si utilizamos dichas funciones para etiquetar los puntos obtenemos los siguientes porcentajes de clasificación.

Clasificador	Aciertos/Puntos
Recta original	0.89
Elipse 1	0.46
Elipse 2	0.38
Hipérbola	0.61
Parábola	0.66

Llegados a este punto, las conclusiones serían que los modelos elípticos no son capaces de obtener una tasa de clasificación lo suficientemente alta (un clasificador aleatorio obtendría una puntuación de $0{,}5$). Por otro lado la hipérbola y la parábola han obtenido resultados mejores al clasificador aleatorio sin llegar a ser tan buenos como el lineal.

Podemos concluir que estos clasificadores son peores que el modelo más simple lineal para clasificar este tipo de datos, lo cual tiene sentido pues los hemos generado usando un modelo lineal, sin embargo, por los resultados obtenidos podríamos pensar que no funcionan tan mal como clasificadores, si miramos las gráficas veremos que esto no es así.

Como se puede observar en la figura 4, la hipérbola y la parábola obtienen buenas puntuaciones sin embargo, no es debido a una buena separación de los puntos, si no que clasifican la mayoría del espacio en la clase correspondiente -1, por ello clasifica correctamente toda esta clase e incorrectamente la otra. Esto nos sugiere la utilización de otra medida para valorar los resultados de los clasificadores, por ejemplo, una que tenga en cuenta los *falsos positivos* obtenidos por el mismo.

En el caso de las elipses ocurre justo lo contrario, el interior de las elipses corresponde a la región clasificada como negativa, dejando entonces la mayoría del espacio en la región positiva, motivo por el que su tasa de clasificación es tan baja.

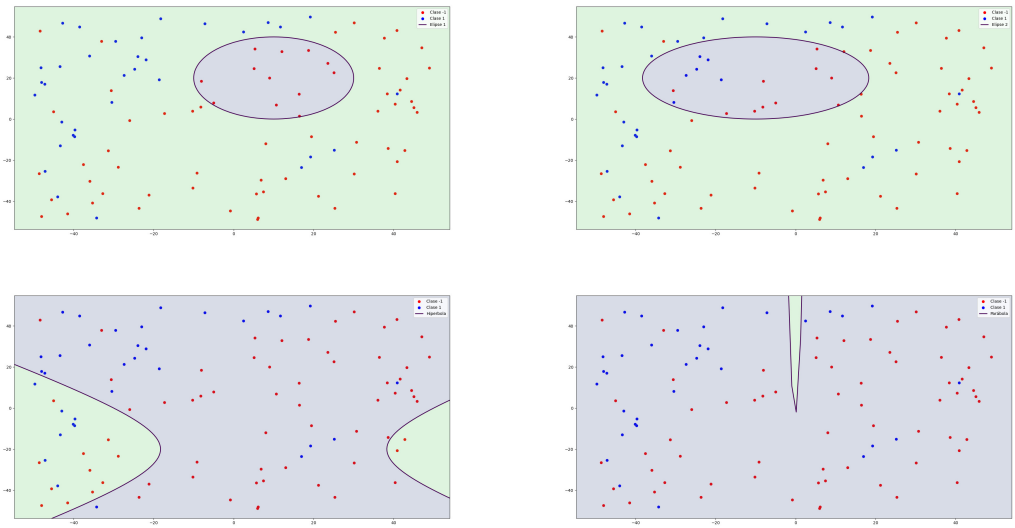

Figura 4: Gráficas obtenidas con los clasificadores elipse 1, elipse 2, hipérbola y parábola.

2. Modelos lineales

2.1. Algoritmo Perceptrón

Implementamos el Algoritmo Perceptrón en la función `ajusta_PLA`, que acepta los siguientes parámetros:

- `datos`. Matriz de datos.
- `labels`. Vector de etiquetas de los datos.
- `max_iter`. Número máximo de iteraciones a realizar por el algoritmo.
- `vini`. Vector de pesos inicial.

El algoritmo devuelve el vector de pesos final y el número de iteraciones que han sido necesarias hasta converger.

El funcionamiento del algoritmo es el siguiente, para cada iteración, recorremos todo el conjunto de datos, para cada dato $x(t)$ con etiqueta $y(t)$, comprobamos si el clasificador obtiene el resultado correcto (signo$(w(t)^T x(t) = y(t))$), en caso contrario, actualizamos los pesos mediante

$$w(t+1) = w(t) + y(t)x(t)$$

De forma que si el vector de pesos clasifica incorrectamente un punto, actualizamos los mismos de forma que la recta que simboliza se «mueve» en la dirección de clasificar dicho punto correctamente.

Como criterio de parada utilizamos tanto el número máximo de iteraciones como la convergencia del vector de pesos (no cambia en una pasada completa por los datos).

Se nos pide utilizar el mismo conjunto de datos (con y sin ruido) que en el ejercicio anterior, como debemos entregar cada ejercicio en un archivo distinto, generamos un conjunto de datos nuevo con los mismos parámetros y semilla.

Buscamos realizar el siguiente experimento con ambos conjuntos de datos: ejecutar el algoritmo 10 veces con un vector de pesos inicial nulo y con vectores iniciales aleatorios y mostrar los resultados medios en cada caso. Al ser un algoritmo determinista, ejecutarlo múltiples veces sobre un mismo vector inicial da el mismo resultado.

Mostramos los resultados en la siguiente tabla:

Cuadro 1: Tabla de resultados experimento PLA

Conjunto de datos	**Vector inicial**	**Iteraciones**	**Porcentaje Correctos**
Sin ruido	Nulo	$\mu = 34{,}0\ \sigma = 0$	$\mu = 100\,\%\ \sigma = 0$
	Aleatorio	$\mu = 93{,}6\ \sigma = 58{,}62$	$\mu = 100\,\%\ \sigma = 0$
Con ruido	Nulo	$\mu = 1000\ \sigma = 0$	$\mu = 79{,}0\,\%\ \sigma = 0$
	Aleatorio	$\mu = 1000\ \sigma = 0$	$\mu = 78{,}9\,\%\ \sigma = 5{,}84$

2.1.1. Datos sin ruido

Comenzamos analizando los resultados obtenidos sin introducir ruido en la muestra, podemos ver el conjunto de datos junto con la recta aproximada por el algoritmo (vector inicial nulo) en la siguiente imagen.

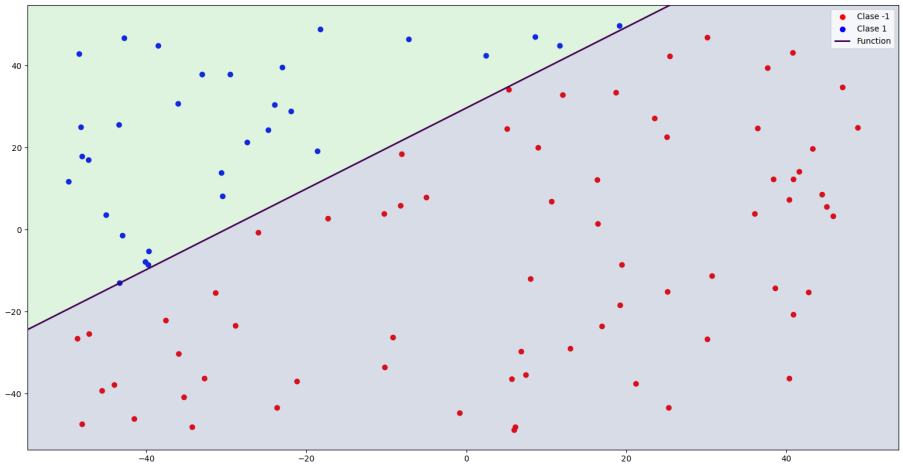

Figura 5: Resultado PLA vector de pesos inicial nulo

Al utilizar un conjunto de datos sin ruido, en todos los casos se ha alcanzado el 100 % de clasificación (desviación típica nula), en cuanto al número de épocas necesarias, a primera vista parece que en caso de utilizar un vector de pesos aleatorio este número aumenta sustancialmente, sin embargo, si nos fijamos en la desviación típica de los resultados, vemos que existe una gran variación en el número de épocas. Por esto extraer una conclusión sobre que vector inicial es mas

conveniente de estos resultados puede ser precipitado.

En la siguiente imagen podemos ver como ha sido la evolución del porcentaje de clasificación a lo largo de las épocas, al utilizar el vector de pesos nulo.

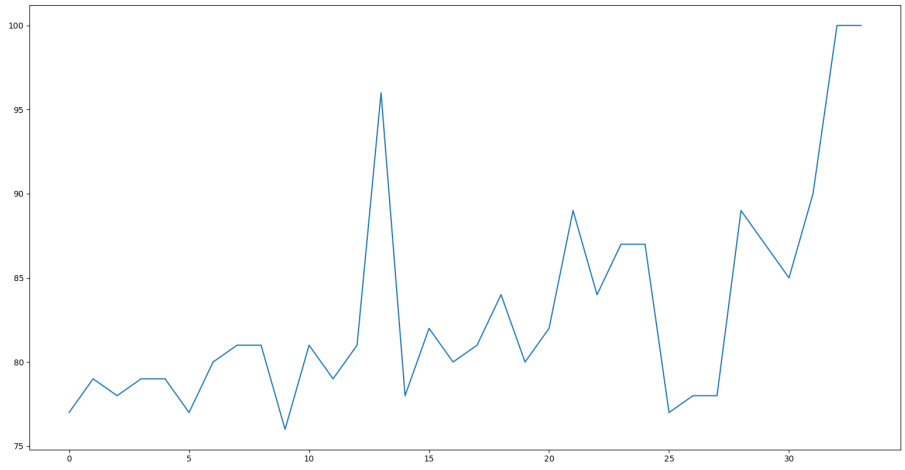

Figura 6: Evolución del clasificador comenzando con un vector nulo.

2.1.2. Datos con ruido

Si utilizamos el conjunto de datos con ruido introducido, podemos ver que sea cual sea el vector de pesos se utilizan todas las iteraciones que se le pongan al algoritmo, esto es debido a que al no ser linealmente separables, que se produzca una convergencia del método es menos probable, pues siempre existirán puntos mal clasificados.

Los resultados obtenidos en la tasa de clasificación siguen siendo similares independientemente del vector de pesos utilizado, incluso existiendo una baja desviación típica al utilizar un inicial aleatorio.

En la siguiente imagen podemos ver la evolución del clasificador utilizando el vector inicial nulo, como se puede ver, se produce una oscilación continua en la tasa de clasificación que produce la no convergencia.

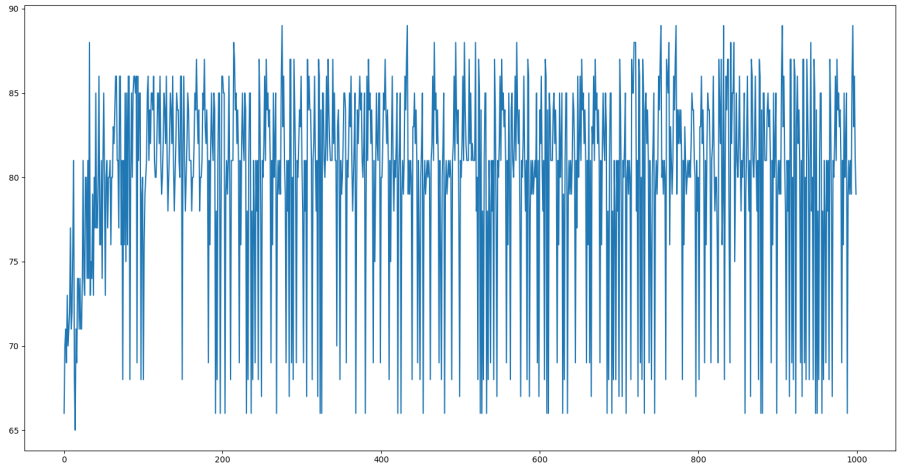

Figura 7: Resultado PLA vector de pesos inicial nulo

2.2. Regresión Logística

En este apartado queremos comprobar como se desenvuelve la regresión logística en un modelo simple donde f es una probabilidad con valores $0/1$. Para ello consideramos como espacio de características $[0, 2] \times [0, 2]$ y como conjunto de etiquetas $\{-1, 1\}$.

Utilizamos una implementación del SGD con tamaño del batch 1, luego el algoritmo se reduce a los siguientes pasos:

- Realizar una permutación de los datos y etiquetas (representado mediante una permutación de los índices).
- Recorrer los datos y actualizar el vector de pesos en todas las iteraciones siguendo

  ```
  w += -eta * d_logistic_error(data, label, w)
  ```

  Donde `d_logistic_error` es la derivada de la función de error de regresión logística.

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

- Como criterio de parada utilizamos que la distancia entre el nuevo vector de pesos y el anterior sea mejor que `0.01`.

Entrenamos el algoritmo con un conjunto de datos de 100 puntos. Generamos ahora un conjunto de tamaño 1000 para estimar E_{out}. Los resultados obtenidos utilizando el vector de pesos generado antes son

```
% correctos: 94.6%
Error: 0.12860094539277195
```

Podemos visualizar los datos en la siguiente imagen.

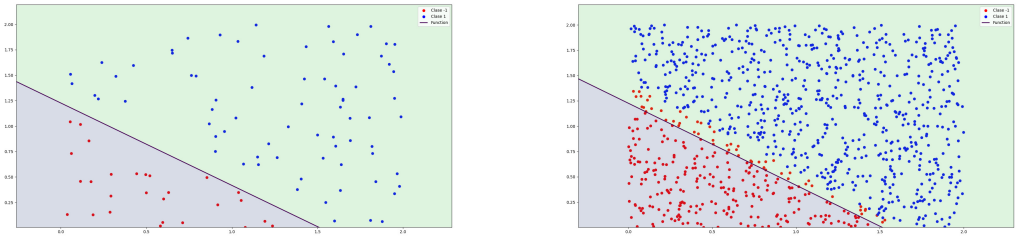

Figura 8: Resultado RL SGD en el conjunto de Entrenamiento y Test

Se puede observar como, pese a separar perfectamente las clases en el conjunto de entrenamiento, existe un pequeño margen que no estaba contemplado en este, que hace que la clasificación no sea perfecta en el conjunto de test.

3. Clasificación de Dígitos

3.1. Planteamiento del problema

Definimos el problema de clasificación binaria consistente en los siguientes elementos.

- Espacio de entrada $\mathcal{X} = \{1\} \times \mathbb{R}^2$
- Espacio de etiquetas $\mathcal{Y} = \{-1, 1\}$.
- Conjunto de muestra $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ muestra aleatoria simple de tamaño $N = 1194$ de la distribución desconocida $P(\mathcal{X}, \mathcal{Y})$ tal que $f(x) = P(y \mid x)$ es la nuestra función objetivo.
- Espacio de funciones candidatas $\mathcal{H} = \{g : \mathcal{X} \to \mathcal{Y}$ tales que $\exists w \in \mathbb{R}^3 \; g(x) = signo(w^T x)\}$.
- Criterio de aprendizaje (ERM): $g^* = argmin_g \frac{1}{N} \sum_{i=1}^{N} loss_g(x_i, y_i)$.
- Función de pérdida $loss_g(x, y) = \mathbb{I}[g(x) = y]$. Donde $\mathbb{I}$ denota la función indicadora.
- Como algoritmo utilizamos el método de la *pseudoinversa* y luego *PLA-POCKET*.

Notamos que buscar la función h es equivalente a buscar el vector de pesos w.

3.2. Regresión Lineal + PLA-Pocket

Como modelo de Regresión Lineal utilizamos el algoritmo de la pseudo-inversa, como comparativa ejecutaremos también el algoritmo sobre un conjunto de pesos inicial aleatorio.

En la siguiente imagen podemos ver las gráficas generadas sobre el conjunto de datos de entrenamiento.

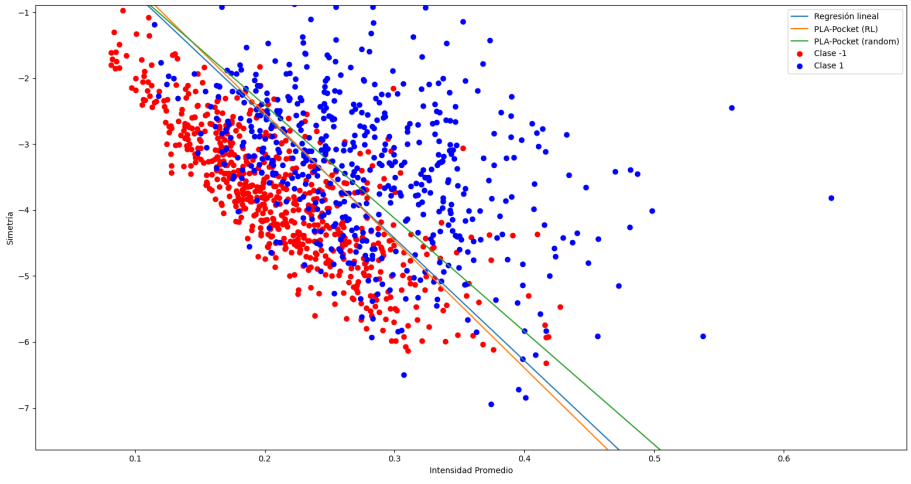

Figura 9: Modelos para clasificación de dígitos sobre el conjunto de entrenamiento.

Veamos ahora el conjunto de test.

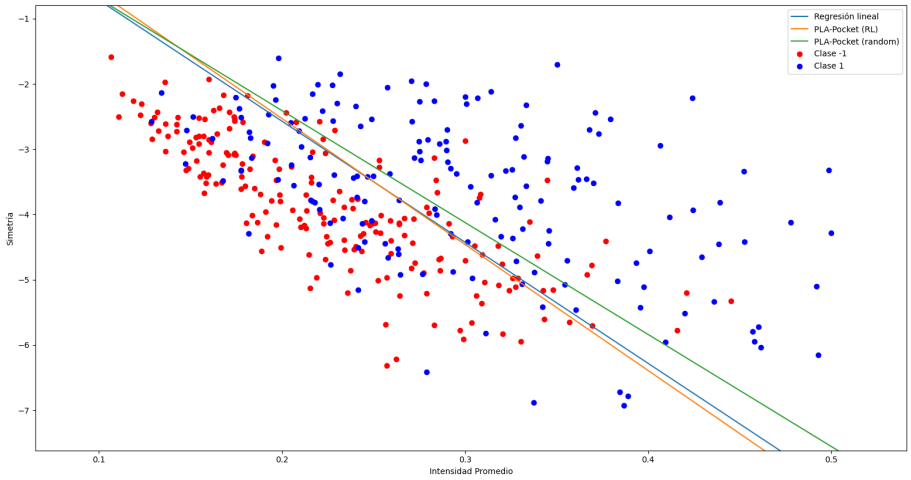

Figura 10: Modelos para clasificación de dígitos sobre el conjunto de test.

Calculamos los valores de E_{in} y E_{test} de las tres aproximaciones. Obtenemos los siguientes resultados.

Modelo	E_{in}	E_{out}
Regresión Lineal	0.22780569514237856	0.25136612021857924
PLA-Pocket (random)	0.22864321608040200	0.24863387978142076
PLA-Pocket (RL)	0.22529313232830822	0.25409836065573770

Como se puede observar, todos se encuentran muy próximos entre sí, siendo el mejor de ellos la mejora del modelo de regresión con PLA-Pocket. El motivo detrás de porque no se produce una mejora mayor puede ser simplemente que los pesos generados por la pseudo-inversa son ya lo suficientemente buenos y no admiten mucha mejoría.

Para calcular las cotas sobre el verdadero valor de E_{out} utilizamos la cota de *Hoeffding*:

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} \log \frac{2M}{\delta}}$$

Donde

- N es el tamaño de la muestra utilizado.
- M es el cardinal del espacio de clasificadores.
- δ es el nivel de tolerancia, es decir, aseguramos que la cota es cierta con probabilidad $1 - \delta$.

El cardinal del espacio al calcular la cota sobre E_{test} es 1 ya que la función candidata ya ha sido fijada. Sin embargo, en el caso del entrenamiento, el espacio tiene cardinal infinito, luego utilizamos que nuestros clasificadores se encuentran codificados como tres valores reales (64 bits). Por

ello, nuestro espacio tiene un total de $(2^{64})^3 = 2^{192}$ clasificadores. Utilizamos este valor en la desigualdad.

Los resultados obtenidos son

```
Cota superior de Eout (con Ein)   ->  0.4646154745114327
Cota superior de Eout (con Etest) ->  0.3250874640883532
```

Donde podemos ver como la cota utilizando el conjunto E_{test} es menor que utilizando E_{in}.