Aprendizaje automático

Práctica 2: Modelos lineales

Antonio Coín Castro

Curso 2019-20

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

Este ejercicio se desarrolla en el script `p2_complejidad.py`. En él perseguimos estudiar las dificultades que introduce la aparición de ruido a la hora de elegir la clase de funciones más adecuada.

En primer lugar definimos un par de funciones para simular puntos. En la primera generamos un número determinado de puntos de forma uniforme en un hipercubo, es decir, en cada componente se extrae una muestra de una distribución uniforme. Recordamos que una distribución uniforme en un intervalo $[a, b]$ es aquella cuya función de densidad viene dada por

$$f_{\mathcal{U}}(x) = \frac{1}{b-a}, \quad a < x < b.$$

La probamos en dimensión 2 para poder visualizar la nube de puntos resultante. En este caso, generamos 50 puntos en el cuadrado $[-50, 50] \times [-50, 50]$, usando para ello la función `uniform_sample`.

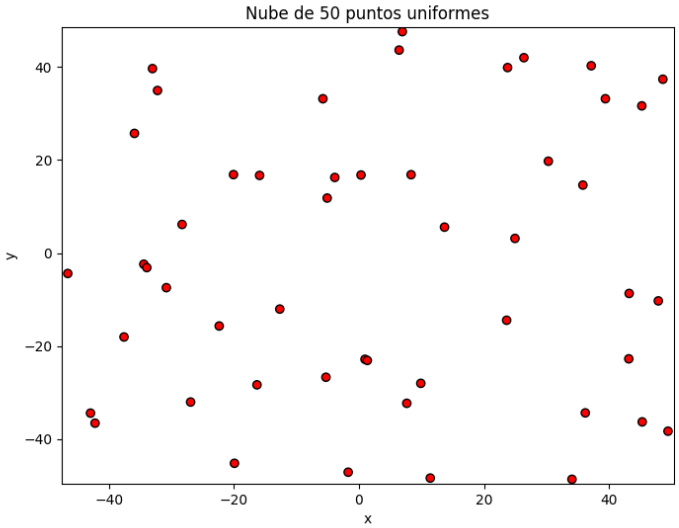


Figura 1: Nube de 50 puntos generados de forma uniforme en $[-50, 50]^2$.

La segunda función de simulación nos proporciona una forma de generar puntos que en cada componente provienen de una distribución normal *distinta*, cuya función de densidad es

$$f_{\mathcal{N}}(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right).$$

Como en el caso anterior, visualizamos el resultado en dimensión 2, generando 50 puntos a partir de dos normales de media 0 y desviaciones típicas 5 y 7, respectivamente. Cabe esperar que los puntos se encuentren en la región $[-3\sigma_x, 3\sigma_x] \times [-3\sigma_y, 3\sigma_y]$, pues sabemos que es ahí donde se concentran casi el 100 % de los valores (Regla 68-95-99.7). Empleamos la función `gaussian_sample`.

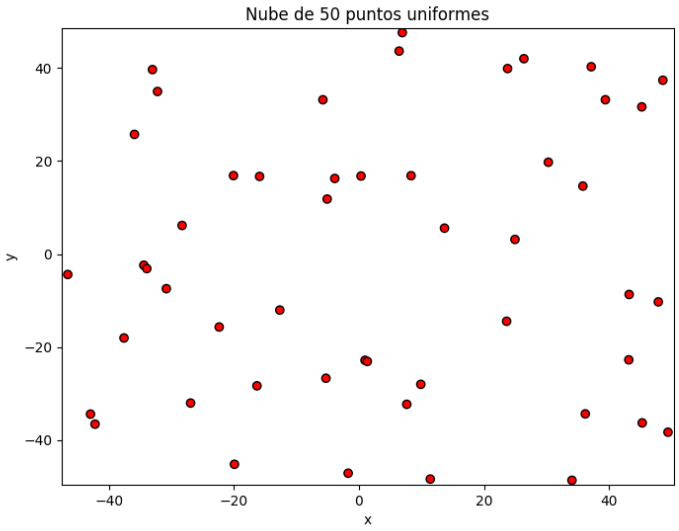


Figura 2: Nube de 50 puntos generados a partir de dos Gaussianas de media 0 y desviaciones típicas $\sigma_x = 5$ y $\sigma_y = 7$.

Procedemos ahora a hacer un pequeño experimento. Generamos 100 puntos de manera uniforme en el cuadrado $[-50, 50] \times [-50, 50]$, y empleando la función `line_sample` simulamos también de manera uniforme una recta que corte a dicho cuadrado. Esta recta queda determinada por su pendiente (a) y su punto de corte con el eje Y (b). Vamos a etiquetar los puntos generados en dos clases, utilizando la recta como frontera de clasificación. En particular, cada punto (x, y) se etiqueta a partir de la función

$$f_{a,b}(x, y) = \text{signo}(y - ax - b),$$

que no es más que el signo de la distancia del punto a la recta. Para evitar problemas suponemos que signo(0) = 1 (es decir, los puntos que están sobre la recta pertenecen a la clase del 1). Podemos ver en la Figura 4 cómo quedan los puntos generados y clasificados en función de la recta elegida. Es evidente que están todos bien clasificados, pues los hemos generado así.

Vamos ahora a introducir ruido en las etiquetas. Mediante la función `simulate_noise` perturbamos el signo de un 10 % de etiquetas positivas y otro 10 % de etiquetas negativas.

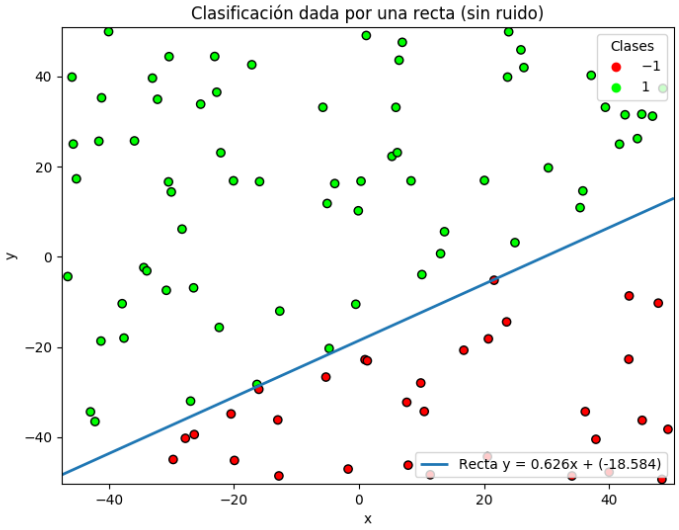


Figura 3: 100 puntos generados uniformemente etiquetados respecto de una recta.

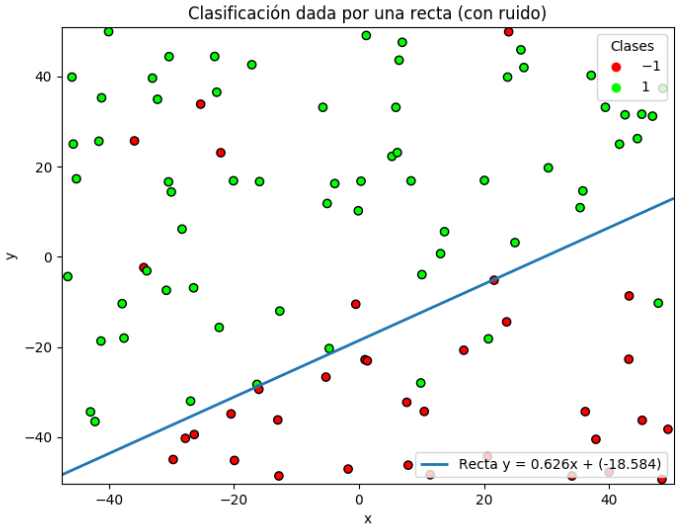


Figura 4: 100 puntos generados uniformemente etiquetados respecto de una recta y con ruido.

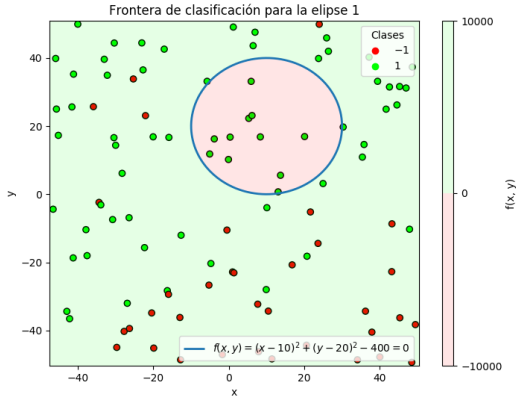
Si volvemos a dibujar los puntos con sus nuevas etiquetas, vemos en la Figura **??** cómo en torno a un 10 % de ellos están mal clasificados con respecto a la recta.

Queremos estudiar cómo se comportarían otros clasificadores más complejos a la hora de clasificar estos puntos con ruido, suponiendo que la función original que genera las etiqueta es $f_{a,b}$. Es decir, queremos ver si aumentando la complejidad de la clase de funciones $\mathcal{H}$ podemos obtener una mejor clasificación. Los clasificadores (cuadráticos) que consideramos son:

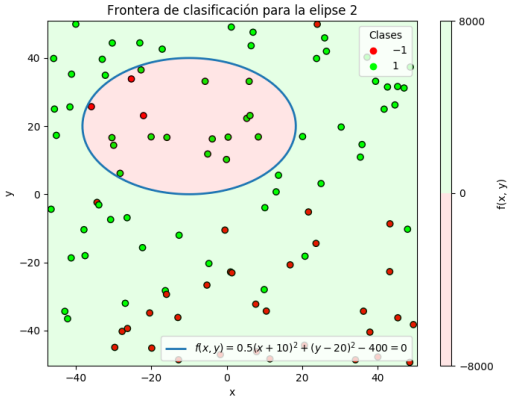
- (Elipse 1) $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$.

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

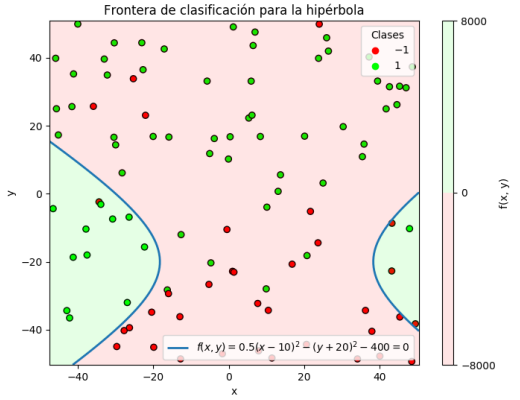
Mostramos en la Figura 5 las regiones positiva (en verde) y negativa (en rojo) de estos clasificadores binarios, notando que la frontera de todos ellos coincide con la curva de nivel $f(x, y) = 0$.



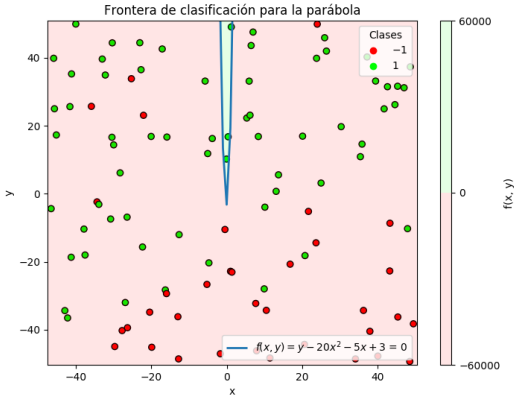
(a) Regiones de clasificación para la elipse 1.



(b) Regiones de clasificación para la elipse 2.



(c) Regiones de clasificación para la hipérbola.



(d) Regiones de clasificación para la parábola.

Figura 5: Regiones positiva y negativa para los distintos clasificadores.

Aunque a simple vista puede verse que ninguna de las cuatro funciones consigue clasificar siquiera de forma decente los datos, estudiamos un par de métricas que nos permitan justificar esto. Vamos a medir el *accuracy* de cada clasificador, es decir, el porcentaje de datos bien clasificados sobre el total. Además, como es posible que la distribución de etiquetas no esté balanceada, medimos también lo que se conoce como *balanced accuracy*, que corrige la medida de *accuracy* teniendo en cuenta la distribución de clases. Si P indica el número de ejemplos positivos, N el número de ejemplos negativos, TP el número de ejemplos positivos correctamente clasificados (*true positives*) y TN el número de ejemplos negativos correctamente clasificados (*true negatives*), tenemos que

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{P + N}, \quad \text{BACC} = \frac{TP/P}{2} + \frac{TN/N}{2}.$$

Si calculamos estados dos métricas sobre todos nuestros clasificadores, obtenemos lo siguiente:

```
Clasificador: recta
    Accuracy = 91.000%
    Balanced accuracy = 89.121%
Clasificador: elipse 1
    Accuracy = 54.000%
    Balanced accuracy = 41.538%
Clasificador: elipse 2
    Accuracy = 53.000%
    Balanced accuracy = 42.747%
Clasificador: hipérbola
    Accuracy = 43.000%
    Balanced accuracy = 53.516%
Clasificador: parábola
    Accuracy = 38.000%
    Balanced accuracy = 52.308%
```

Vemos cómo de los cuatro clasificadores cuadráticos las elipses son las que más *accuracy* consiguen, mientras que la hipérbola y la parábola obtienen una puntuación muy baja (como ya podíamos adivinar a partir de las gráficas). En nuestro caso la proporción de clases es del 65 % para la clase del 1 y del 35 % para la clase del -1, por lo que en la métrica balanceada los dos últimos clasificadores mejoran un poco al tener una región negativa bastante mayor. Sin embargo, ninguno de estos clasificadores cuadráticos consigue siquiera destacar ante un *clasificador aleatorio*, que obtendría de media un *accuracy* del 50 %, y no se acerca ni por asomo a la precisión obtenida con el clasificador lineal.

Podemos concluir que en este caso el aumento en la complejidad del modelo no conduce a una mejora de resultados, pues aún habiendo introducido ruido en las etiquetas, el mejor clasificador sigue siendo el más sencillo: la función lineal.

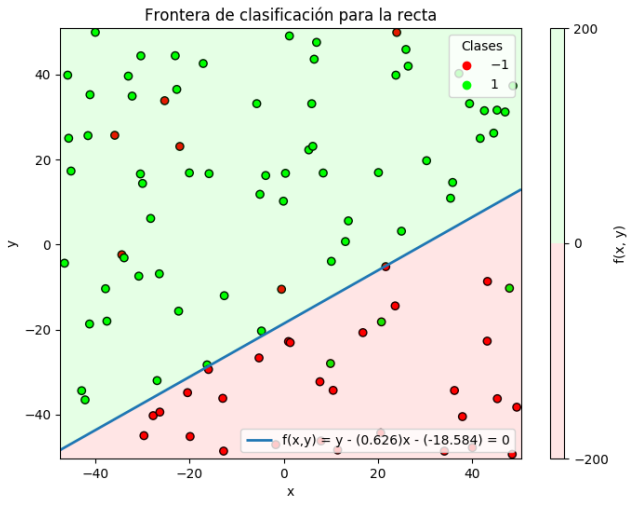


Figura 6: Regiones de clasificación para la recta.

El hecho de permitir que aumente la complejidad de la clase de funciones a elegir no siempre hace que aumente el rendimiento del modelo. De hecho, si intentásemos aprender un clasificador muy complejo (por ejemplo, un polinomio de grado elevado) que se ajustase a estos datos mejor que una recta, lo más probable es que cayéramos en el *sobreajuste* y dicho clasificador no consiguiera generalizar sobre nuevos puntos. Es por esto que en general debemos evitar que la presencia de ruido en los datos nos induzca a pensar que el modelo subyacente es mucho más complejo de lo que es en realidad.

Ejercicio sobre modelos lineales

PLA

- tomamos una iteración en PLA como una pasada por el conjunto de datos completo.
- si los datos son linealmente separables, hay infinitas rectas que los separan.
- se usa la evolución para hacer las gráficas
- las rectas son w^Tx = 0.

```
--- Apartado a): etiquetas sin ruido
Recta original
    Accuracy = 100.000%
    Balanced accuracy = 100.000%

Recta encontrada por PLA con vector inicial 0
    Iteraciones: 99
    Accuracy = 100.000%
    Balanced accuracy = 100.000%

Recta encontrada por PLA con vector inicial aleatorio (de media)
    Iteraciones: 96.7
    Accuracy = 100.000%
    Balanced accuracy = 100.000%


--- Apartado b): etiquetas con ruido
Recta original
    Accuracy = 91.000%
    Balanced accuracy = 89.121%

Recta encontrada por PLA con vector inicial 0
    Iteraciones: 1000
    Accuracy = 84.000%
    Balanced accuracy = 84.396%

Recta encontrada por PLA con vector inicial aleatorio (de media)
    Iteraciones: 1000.0
    Accuracy = 83.700%
    Balanced accuracy = 83.967%
```

RL

- SGD con bs=1, si no no converge (se ha probado) Instead of reading everything and then correct yourself at the end, you correct yourself on the way, making the next reads more useful since you correct yourself from a better guess.

```
--- EJERCICIO 2: REGRESIÓN LOGÍSTICA ---
Iteraciones: 501
E_in = 0.128
E_out (en 1000 nuevos puntos) = 0.119
Accuracy en training = 98.000%
Balanced accuracy en training = 96.667%
Accuracy en test = 99.500%
Balanced accuracy en test = 99.284%
```

Bonus

-1 –> 4 1 –> 8

- sabemos que los datos no son linealmente separables (por eso usamos pocket, que es más lento que solo PLA), así que el número de iters es fijo.
- Se usa como ein el error de clasificación
- Cotas: usando VC y usando hoeffding. Los puntos de test < puntos de training.
- mostrar gráfica con la evolución de pla para contrastar

```
---- Pseudoinversa
Vector de pesos = [-0.507 8.251 0.445]
Errores:
    E_in = 0.22781
    E_test = 0.25137
Cotas para E_out:
    Cota usando E_in (VC) = 0.65874
    Cota usando E_in (Hoeffding) = 0.46713
    Cota usando E_test (Hoeffding) = 0.32236

---- PLA-Pocket (aleatorio)
Vector de pesos = [-8.014 140.224 8.222]
Errores:
    E_in = 0.22864
    E_test = 0.24863
Cotas para E_out:
    Cota usando E_in (VC) = 0.65958
    Cota usando E_in (Hoeffding) = 0.46797
    Cota usando E_test (Hoeffding) = 0.31962

---- PLA-Pocket (pseudoinversa)
Vector de pesos = [-6.507 94.333 4.884]
```

```
Errores:
    E_in = 0.22529
    E_test = 0.25410
Cotas para E_out:
    Cota usando E_in (VC) = 0.65623
    Cota usando E_in (Hoeffding) = 0.46462
    Cota usando E_test (Hoeffding) = 0.32509
```