Práctica 2: Complejidad de conjuntos hipótesis, ruido y modelos lineales

Alejandro Alonso Membrilla

Contents

Introducción

La primera parte de esta práctica consistirá en la comparación de diversos tipos de modelos pertenecientes a clases de hipótesis distintas. En ella realizaremos un experimento sencillo generando un conjunto de datos y mediremos el error obtenido por cada una de estas hipótesis al clasificar dichos datos.

La segunda parte tratará sobre dos tipos distintos de modelos lineales: el algoritmo del perceptrón (o PLA) y la regresión lineal. Implementaremos ambos algoritmos y realizaremos diversos experimentos a partir de los mismos.

En ambas partes haremos también un estudio superficial del problema del ruido. Veremos cómo puede perjudicar la presencia de ruido a las soluciones obtenidas, o incluso evitar la convergencia de algunos algoritmos.

El código implementado para los experimentos ha sido escrito en Python, haciendo uso de la biblioteca `numpy` para álgebra lineal y generación de números pseudo-aleatorios, y `matplotlib` para la visualización de los resultados. Los cálculos y los gráficos obtenidos para muchos experimentos en esta práctica han sido realizados en el ordenador personal del autor de la misma, Alejandro Alonso Membrilla, que cuenta con las siguientes características:

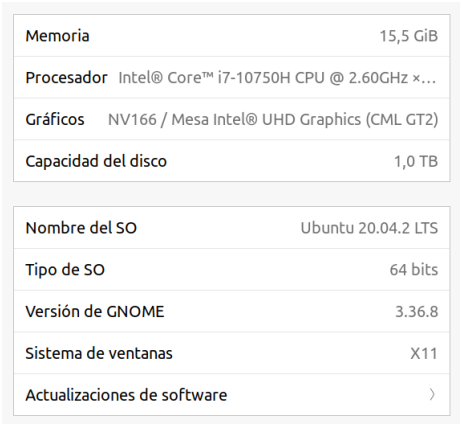


Figure 1: Especificaciones del hardware utilizado

Ejercicio sobre la complejidad de H y el ruido

Generación aleatoria de datasets de prueba

En este apartado hemos usado las funciones suministradas *simula_unif* y *simula_gaus*, que generan puntos en una distribución uniforme y gaussiana, respectivamente, según los parámetros especificados. El resultado puede verse en la figura 2.

Hemos generado 50 elementos para cada nube de puntos. Para la primera los hemos muestreado uniformemente del rectángulo $[-50, 50] \times [-50, 50]$. Para la segunda, la distribución normal utilizada tiene media 0 y varianza dada por la matriz de covarianzas

$$\begin{pmatrix} 5 & 0 \\ 0 & 7 \end{pmatrix}$$

que representa una desviación típica de $\sqrt{5}$ en el eje x y de $\sqrt{7}$ en el eje y.

Complejidad de H y ruido

A continuación, hemos generado otro conjunto de datos muestreando 100 puntos de una distribución uniforme en el rectángulo $[-50, 50] \times [-50, 50]$, junto con una recta creada mediante la función *simula_recta* suministrada.

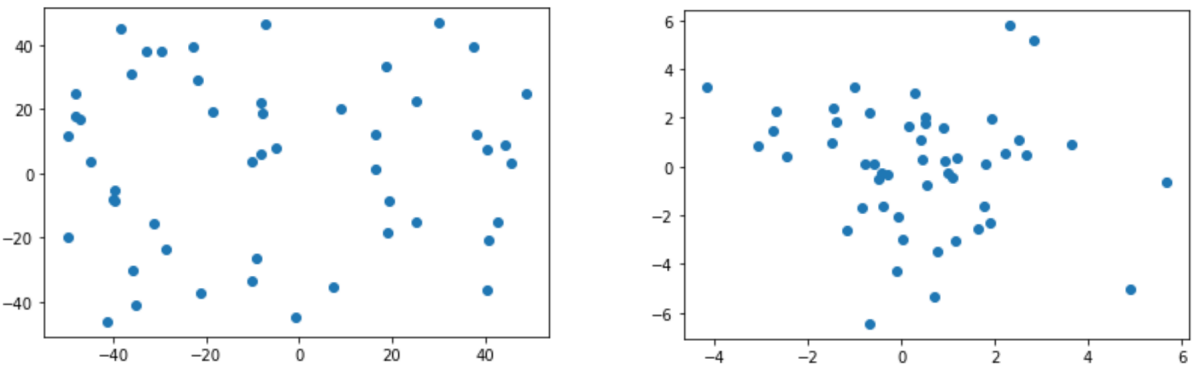


Figure 2: Nubes de puntos usando *simula_unif* y *simula_gaus*

Dicha recta pasa por el rectángulo $[-50, 50] \times [-50, 50]$, separando los puntos generados anteriormente. Tanto la nube de puntos como la recta usada para clasificarlos pueden observarse en la siguiente gráfica:

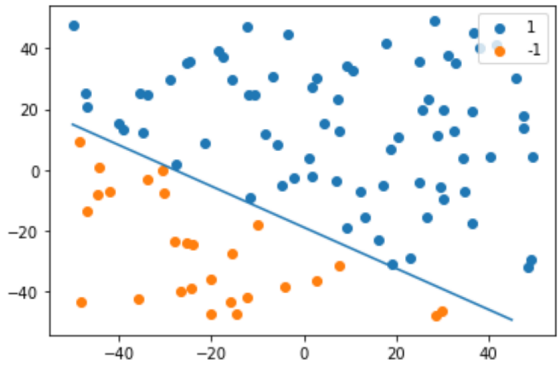


Figure 3: Clasificación de la nube de puntos con la recta generada

Para la clasificación anterior hemos usado el signo de la función $f(x) = y - ax - b$, donde a y b son los coeficientes de la recta dados por la función *simula_recta*.

Posteriormente, queremos cambiar la etiqueta al 10% de los puntos de cada clase. Realmente esto no es necesariamente posible, puesto que no hay forma de garantizar que el número de puntos de cada clase sea un múltiplo de 10, y no podemos cambiar la etiqueta de una cantidad "no entera" de puntos. Por tanto, hemos resultado cambiar la etiqueta de $\lfloor k \cdot 0.1 \rfloor$ puntos para cada clase, donde k representa el número de puntos total de la misma.

La clasificación final tras insertar el ruido aleatorio puede verse en la figura 4. Puede comprobarse "a ojo" que solo hay 9 puntos mal clasificados, lo cual es consecuencia de la aproximación explicada anteriormente.

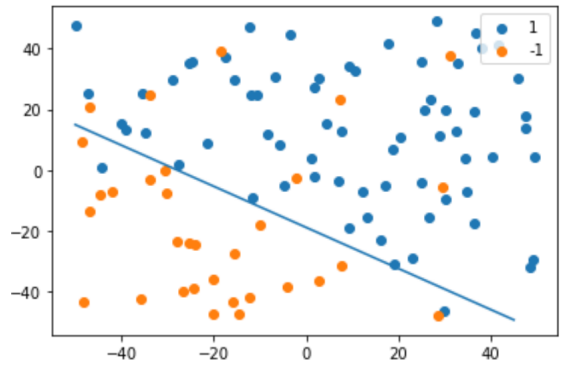
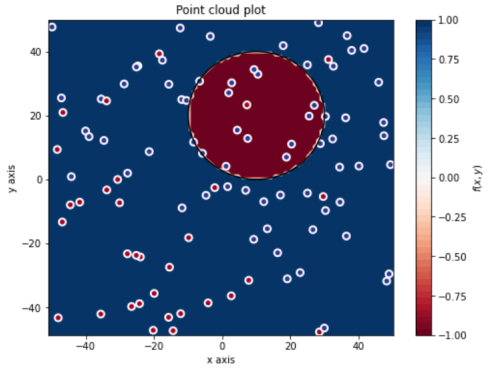


Figure 4: Clasificación de la nube de puntos añadiendo ruido aleatorio

Finalmente, tomaremos las cuatro funciones dadas y compararemos la separación del dominio realizada por estas y su clasificación de la nube de puntos con las originales.

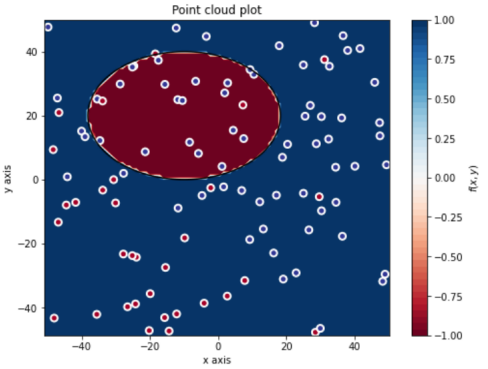
- $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$

  El error obtenido por esta función al clasificar los puntos de la gráfica anterior es del 44%. La siguiente gráfica muestra las zonas en las que la función tiene signo positivo (en azul) y en las que tiene signo negativo (en rojo).



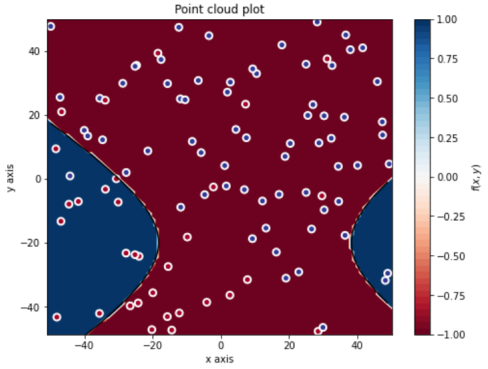
- $f(x, y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$

  El error obtenido por esta función es del 50%. Separa la nube de puntos de la siguiente forma:



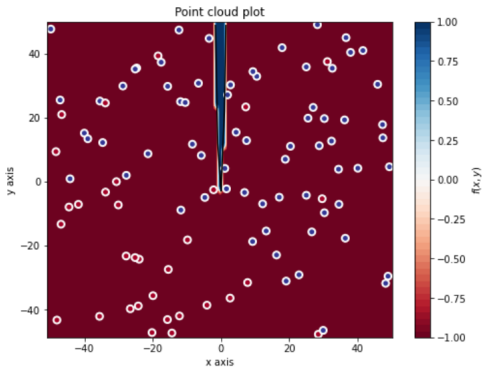
- $f(x, y) = 0,5(x - 10)^2 - (y + 20)^2 - 400$

  El error obtenido por esta función es del 77%. Separa la nube de puntos de la siguiente forma:



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

  El error obtenido por esta función es del 68%. Separa la nube de puntos de la siguiente forma:



Un rápido vistazo es suficiente para observar que la forma de las funciones consideradas no se parece en absoluto a la separación de la nube de puntos original y, por tanto, intentar usar cualquiera de estas funciones para clasificar el conjunto inicial está fuera de lugar.

Por un lado, cabe destacar que al usar estas funciones para clasificar los puntos originales no ha habido ningún tipo de aprendizaje, por lo que sería absurdo esperar *a priori* que alguna de estas funciones, escogida al azar, pudiese ser útil para clasificar puntos que siguiesen distribuciones incluso de su misma clase (por ejemplo, elípticas en el caso de las dos primeras).

Por otro lado, esto también indica que usar funciones muy complejas no garantiza por sí mismo una mejor clasificación. En este caso nuestro conjunto de datos sigue un modelo relativamente sencillo, y al usar funciones más complejas nuestras clasificaciones empeoran al no ajustarse a la estructura real del dataset.

Además, en lo que respecta la selección de un modelo en presencia de ruido, puede ser interesante resaltar el hecho de que la solución más adecuada para este problema, la propia recta que hemos usado para etiquetar los datos, no es capaz de deshacerse del 10% de error ocasionado por el ruido aleatorio. Este problema puede darse igualmente independientemente de la complejidad de la función que usemos. Una función relativamente compleja, como las cuatro anteriores, no será capaz de reducir el error más allá de lo que permita la "limpieza" del conjunto de datos incluso aunque aproxime perfectamente la estructura interna del mismo.

Modelos Lineales

Algoritmo del Perceptrón

En esta sección implementaremos el algoritmo del perceptrón o PLA. Este método progresa tomando elementos del conjunto de datos y, en caso de que se encuentren mal clasificados, "avanzando" la frontera de clasificación en la dirección correcta para etiquetarla correctamente. La regla de actualización del perceptrón puede representarse de la siguiente forma (*Learning From Data*, Y.S.Abu-Mostafa *et al*):

$$w(t+1) = w(t) + y(t)x \tag{1}$$

Donde $w(t)$ representa el valor de los pesos en la iteración t, y x e $y(t)$ representan respectivamente un dato mal clasificado y su etiqueta en esa iteración.

A continuación mostraremos los resultados obtenido por el PLA para clasificar los puntos del ejercicio 1, comparando los resultados de usar pesos iniciados aleatoriamente con los de usar un vector de 0s como pesos iniciales.

En nuestra implementación del PLA procedemos recorriendo (una permutación aleatoria de) el conjunto de datos, y comprobando si el elemento observado se encuentra bien etiquetado. Esto será una iteración. En el caso de que no esté bien etiquetado, aplicaremos la fórmula (1), actualizando los pesos. Si se recorren todos los elementos del dataset sin encontrarse ejemplos mal clasificados la ejecución se detiene (convergencia).

Sin ruido

En el caso de la nube de puntos sin ruido el dataset es linealmente separable. Es conocido que, en esta situación, el PLA siempre convergerá a la separación perfecta, así que la única diferencia la encontraremos en el número de iteraciones necesarias.

Cuando inicializamos los pesos como un vector de ceros, el número de iteraciones necesarias para la convergencia es de 7501 (recordamos que, como mínimo, las 100 últimas son necesarias para comprobar la convergencia). Cuando inicializamos los pesos de forma aleatoria, nos encontramos que el número de iteraciones promedio es de 15471, con una desviación típica relativamente alta de 8190.5 iteraciones. Esto implica que la inicialización de pesos a 0 parece ser adecuada para el PLA, y que este método es altamente dependiente de los pesos iniciales.

Con ruido

En el caso de los datos con ruido, hemos visto en la figura 4 que el conjunto de datos no es linealmente separable. Esto implica que el PLA no será capaz de converger, como hemos podido comprobar empíricamente en el hecho de que el algoritmo ha agotado el número máximo de iteraciones indicado (10^5 iteraciones).

El error obtenido inicializando los pesos a 0 es del 18%, aunque no podemos concluir que este resultado sea estadísticamente significativo sin un experimento más exhaustivo. Inicializando los pesos aleatoriamente, el error medio alcanzado es del 22.9%. Estos resultados son relativamente malos, conociendo la solución óptima y la cantidad de ruido en el conjunto de datos. Conociendo el comportamiento del algoritmo perceptrón, es probable que las soluciones oscilen al no ser capaz de converger. Que el error de clasificación al alcanzar el máximo de iteraciones disponibles sea alto no quiere decir que no se hayan alcanzado soluciones mejores a lo largo del proceso de ajuste. En base a esto, concluimos que una mejora sobre el PLA que almacene las mejores soluciones encontradas, el PLA-Pocket, puede ser mucho más efectiva en casos como este.

Regresión Logística

La regresión logística genera un modelo lineal que representa la probabilidad de cada elemento de pertenecer a una clase determinada. Dicha probabilidad viene dada en la forma $h(x) = \sigma(w^T x)$, donde

$$\sigma(x) = \frac{e^x}{1 + e^x}$$

es la función logística, que toma valores entre 0 y 1. Nuestro objetivo, por tanto, será el de encontrar los pesos w que maximicen la verosimilitud de asignar a cada elemento x la etiqueta que le corresponde.

Aplicando Descenso de Gradiente Estocástico con un tamaño de minibatch de 1, y obviando algunos detalles de la teoría relativos al cálculo de la expresión del gradiente del error logístico, obtenemos la siguiente regla de optimización estocástica: recorremos (una permutación aleatoria de) el conjunto de datos de entrenamiento y, por cada par dato-etiqueta (x, y), aplicamos

$$w(t+1) = w(t) + \eta \frac{yx}{1 + e^{yw^T x}} \tag{2}$$

donde η es el ratio de aprendizaje (en este caso, fijado a 0.01). Siguiendo las indicaciones dadas en el enunciado, los pesos iniciales se han fijado a 0, y se ha impuesto como condición de parada que la diferencia entre norma entre los pesos obtenidos al final de una época difieran de los que había al principio de dicha época en menos de 0.01. Siguiendo la notación de la ecuación (2), esto equivale a que $||w(t-N) - w(t)|| < 0.01$, donde N es el tamaño del conjunto de entrenamiento. Dado que no se especificaba una norma particular en las indicaciones, se ha experimentado usando las normas 2 e infinito. Si no se cumple la condición de parada, empieza una nueva época: se vuelve a barajar el conjunto de datos y se aplica la regla (2) por cada elemento del mismo.

EXPERIMENTO

El experimento a realizar consiste en generar 100 puntos aleatorios uniformemente distribuidos en el cuadrado $[0,2]\times[0,2]$, y etiquetarlos usando una recta generada de forma análoga a la usada en el ejercicio 1. Estos puntos serán el conjunto de entrenamiento. Generamos y etiquetamos otros 1000 puntos usando la misma recta, que harán de conjunto de test y se usarán para calcular E_{out}. Entrenamos nuestros pesos para regresión logística con SGD tal como se ha explicado anteriormente, y calculamos el error en el conjunto de test con los pesos resultantes usando tanto la entropía cruzada (medida del error empírico en RL) como el error de clasificación.

Cabe destacar el hecho de que, dado el modelo entrenado en regresión logística, cada elemento se etiqueta con $+1$ si la probabilidad de que pertenezca a esta clase es mayor a $\frac{1}{2}$, y se etiqueta con -1 en caso contrario. Dado que, como hemos dicho, esta probabilidad viene dada por $h(x) = \sigma(w^T x)$, basta con ver que $h(x) > \frac{1}{2} \iff w^T x > 0$, por lo que podemos usar la misma función de etiquetado que en regresión lineal: $y = signo(w^T x)$.

Los resultados promedios de aplicar el experimento anterior 100 veces usando la norma infinito son:

- ERM medio: 0.131
- Error de clasificación medio: 0.0207
- ERM medio en test: 0.1398
- Error de clasificación medio en test: 0.03066
- Número medio de épocas: 263.61

Mientras que los resultados promedios usando la norma 2 son:

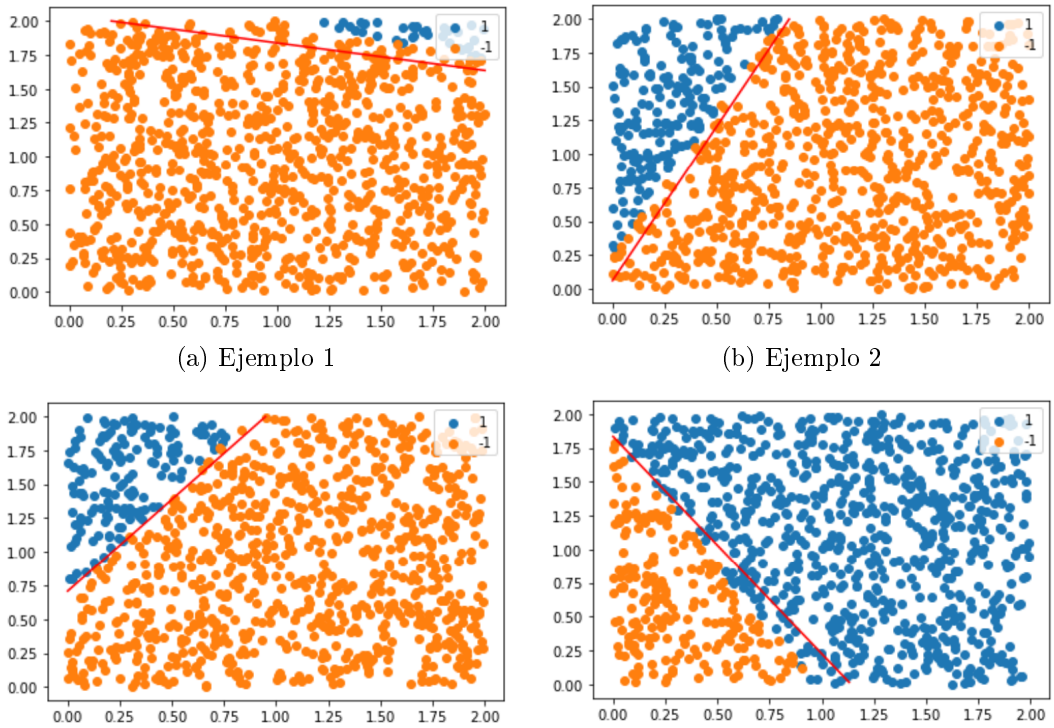
- ERM medio: 0.1077
- Error de clasificación medio: 0.0169
- ERM medio en test: 0.1184
- Error de clasificación medio en test: 0.02573
- Número medio de épocas: 412.85

Si comparamos los resultados de usar ambas normas, vemos estos mejoran en promedio usando la norma 2 (estrictamente mayor a la norma infinito y, por tanto, más exigente), pero el número de épocas medio también aumenta notablemente.

Por último, hemos repetido el experimento otras 4 veces usando la norma 2, que ha obtenido mejores soluciones, y hemos representado a modo de ejemplo el etiquetado resultante de regresión logística junto a la recta original que trataba de modelar (figura 5).

Observamos que los resultados obtenidos por regresión logística para este problema concreto son peores que los obtenidos por el PLA, al menos si los comparamos con los del apartado anterior en el que PLA lograba converger a la solución óptima (0% de error) en 7501 iteraciones, unas 75 épocas teniendo en cuenta que el dataset presentaba 100 elementos.

Es esperable, sin embargo, que regresión logística pueda obtener soluciones mucho mejores a las del algoritmo perceptrón en presencia de ruido, dado que asume que el conjunto de datos sigue una distribución de probabilidad. También cabe destacar el hecho de que hemos fijado el parámetro de la tolerancia usada en la condición de parada a 0.01 de forma arbitraria. Reducir este umbral podría hacer converger el algoritmo de forma más precisa a la solución óptima, a costa de aumentar todavía más el número de épocas necesarias.



(a) Ejemplo 1

(b) Ejemplo 2

(c) Ejemplo 3

(d) Ejemplo 4

Figure 5: Ejemplos de clasificación de datos linealmente separables mediante Regresión Logística

Referencias

- *Learning From Data*, Y.S.Abu-Mostafa *et al.*
- *How to block calls to print? (StackOverflow)*
- Referencia de NumPy