UNIVERSIDAD DE GRANADA

AA

Aprendizaje Automático

Práctica 2

Autores:
Mario Carmona Segovia

Profesor: Pablo Mesejo Santiago

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice

Índice de figuras

1. Ejercicio 1: La complejidad de H y el ruido

1.1. Ejercicio 1.2

1.1.1. Influencia de la complejidad en los resultados de la clasificación

En este ejercicio se han usado unas cuantas funciones más complejas, que la usada para clasificar los datos en principio, para comparar los resultados. Las funciones utilizadas son las siguientes:

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

La gráfica de la función $f(x, y) = y - ax - b$, que es la función utilizada para clasificar los datos de la muestra, y con la que se va a comparar el resto de funciones es la siguiente:

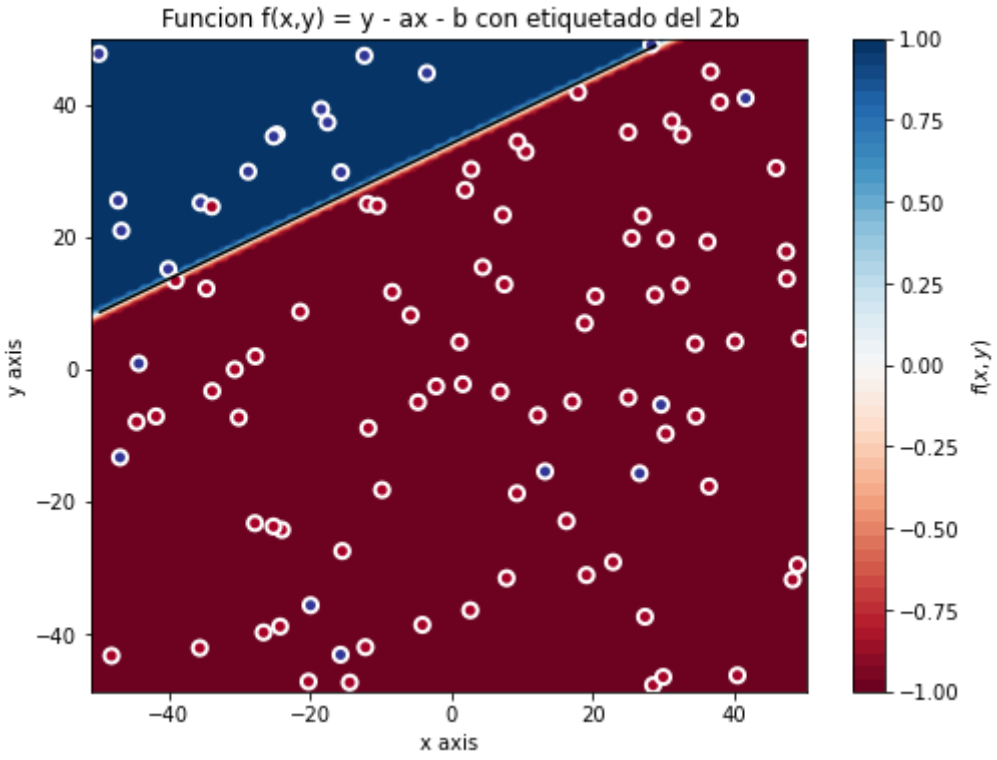
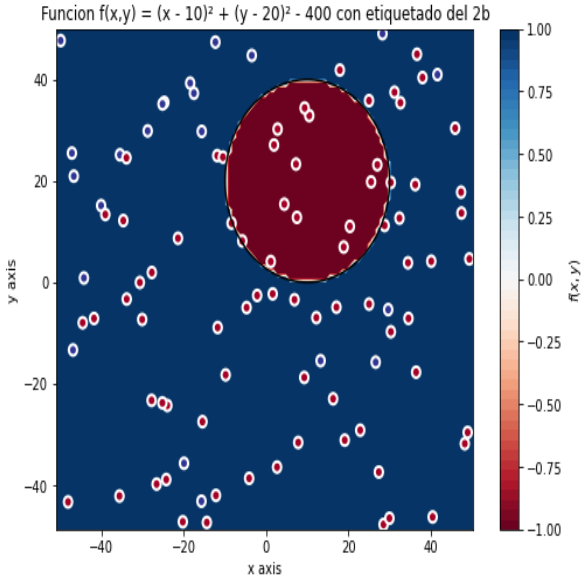
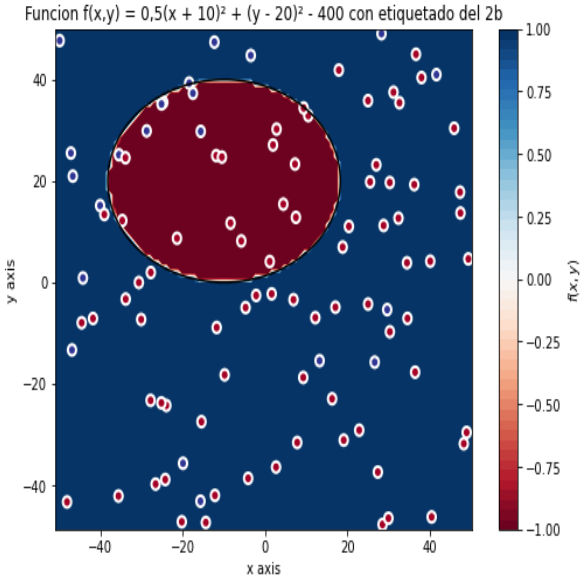


Figura 1: Ejercicio 1.2: Función f(x,y) = y - ax - b

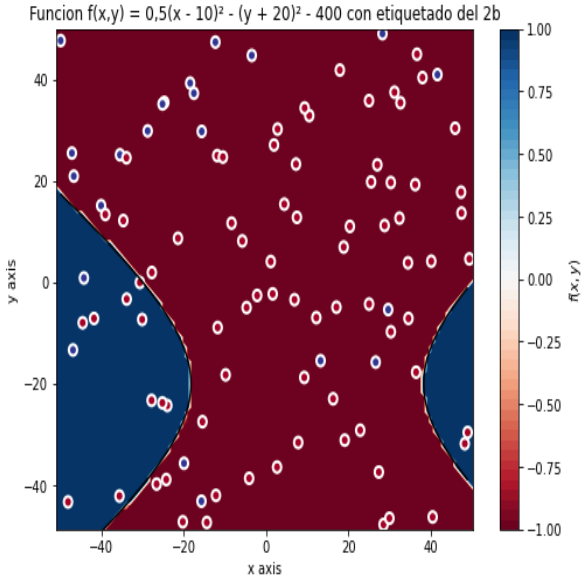
Las gráficas obtenidas con estas funciones utilizando los datos clasificados con la función $f(x, y) = y - ax - b$, son las siguientes:



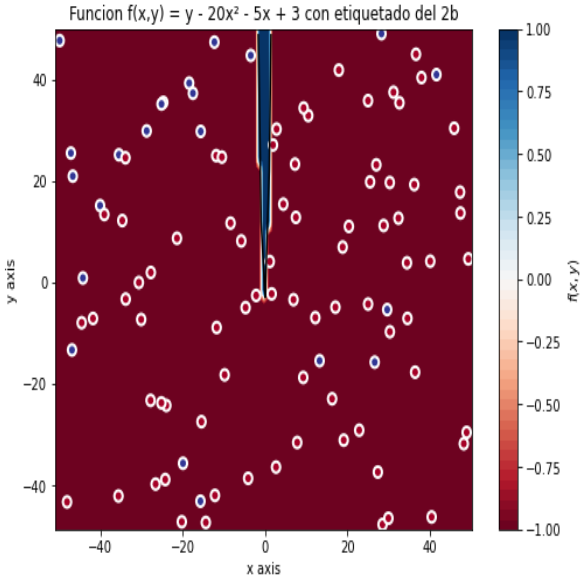
(a) Ejercicio 1.2: Función $f(x, y) = (x-10)^2 + (y-20)^2 - 400$



(b) Ejercicio 1.2: Función $f(x, y) = 0,5(x+10)^2 + (y-20)^2 - 400$



(c) Ejercicio 1.2: Función $f(x, y) = 0,5(x-10)^2 - (y+20)^2 - 400$



(d) Ejercicio 1.2: Función $f(x, y) = y - 20x^2 - 5x + 3$

Figura 2: Ejercicio 1.2: Resto de funciones más complejas

Como se puede apreciar ninguna de las funciones reduce el error obtenido con la función inicial. Esto principalmente se debe a que no se ha realizado un proceso de aprendizaje con las funciones, sino que se han asignado unos valores fijos a las funciones, por lo que no se ajustan a los datos; y por lo tanto es imposible que mejoren los resultados de la función inicial, ya que los datos han sido etiquetas usando esa función como frontera, por lo que se trata de la recta perfecta que separa perfectamente los datos de la muestra.

Pero en el caso de que no se hubiese etiquetado los datos de esa manera y hubiese la posibilidad de obtener menor error dentro de la muestra con alguna de las nuevas funciones, seguiría siendo un error, ya que al ser más complejas es posible que se ajusten mejor a los datos de la muestra, pero tanto ajuste en la muestra hace que pierda generalidad el modelo y obtenga un peor error fuera de la muestra, que es realmente donde importa el error. El ejemplo más extremo que se puede ver, es el de la función $f(x, y) = y - 20x^2 - 5x + 3$, que se justa en exceso a los datos.

De estos resultados se puede deducir que el aumento de la complejidad no implica una mejora en los resultados. Se debe intentar encontrar la función que mejor se ajuste a los datos de la muestra, pero sin perder la generalidad necesaria para dar buenos resultados fuera de esa muestra.

1.1.2. Influencia del ruido en la selección de la complejidad de la clase de funciones

El ruido es un factor que siempre se encuentra al clasificar datos en problemas reales. El error que se obtiene con la función $f(x, y) = y - ax - b$, es del 10 %, que es exactamente el error provocado por el ruido, por lo que el error que se obtiene no se puede reducir más. Así que es absurdo usar una función de mayor complejidad.

Además en el caso de no tener esta recta perfecta para los datos usados, si utilizase una función de mucha complejidad y los datos con ruido sin etiquetar, aunque se consiguiese ajustar al máximo la función a los datos, siempre se obtendría como mínimo un 10 % de error derivado del ruido. Por lo que una función de mayor complejidad no es mejor clasificador de los datos con ruido.

2. Ejercicio 2: Modelos lineales

2.1. Ejercicio 2.1

En este ejercicio se hace uso del algoritmo del perceptron (PLA) para ajustar un modelo a los datos.

El algoritmo del perceptron (PLA) consiste en los siguientes pasos:

1. Mientras no se mejore en toda una época y no se llegue al máximo de iteraciones
2. Iterar por todos los datos de la muestra realizando lo siguiente:
   - Obtener la predicción del modelo
   - Si no se acierta en la predicción, se modifican los pesos

Una vez visto en que consiste el algoritmo PLA, vamos a analizar los resultados obtenidos en el ejercicio.

Con los datos sin ruido se han obtenido los siguientes resultados:

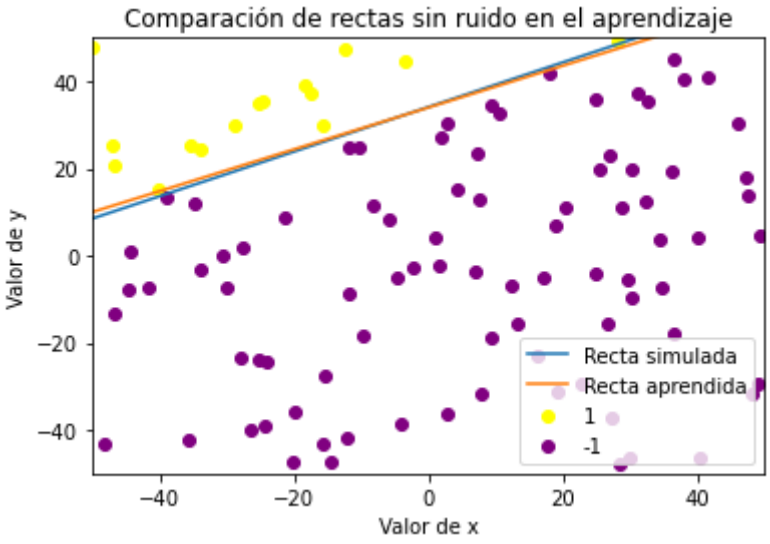


Figura 3: Ejercicio 2.1: Datos sin ruido

Número medio de iteraciones necesarias = 6920 iteraciones

Con los datos con ruido se han obtenido los siguientes resultados:

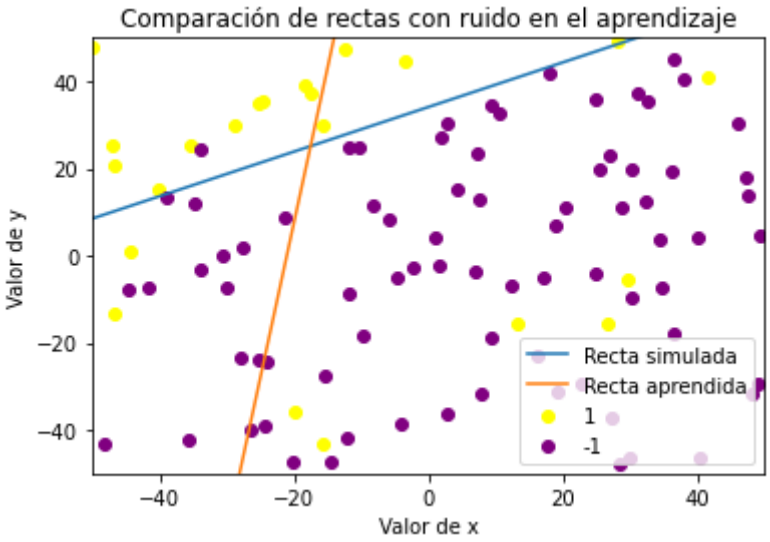


Figura 4: Ejercicio 2.1: Datos con ruido

Número medio de iteraciones necesarias = 10000 iteraciones

La gran diferencia entre las gráficas, es que en la gráfica [3], la recta del modelo se ajusta casi perfecto a la recta simulada; mientras que en la gráfica [4], la recta del modelo es muy distinta de la recta simulada. Esto se debe al ruido que hay en los datos de la gráfica [4].

Este ruido provoca que los datos no sean linealmente separables, por lo que será imposible hallar un modelo que clasifique perfectamente los datos. Debido a esto, para crear el modelo de la gráfica [3] no hace falta utilizar el máximo de iteraciones, ya que llega un momento en que no hace falta mejorar los pesos, ya que no habrá predicciones fallidas, y el modelo converge. Mientras que para obtener el modelo de la gráfica [4] se utiliza el máximo de iteraciones, ya que en todas la iteraciones alguna predicción fallará al no ser los datos linealmente separables. Por lo que en esta gráfica da igual el número máximo de iteraciones que nunca convergerá el modelo.

2.2. Ejercicio 2.2

Para este ejercicio se ha usado el algoritmo de la regresión logística (LGR) para ajustar los pesos. El algoritmo LGR es una variante del gradiente descendente estocástico (SGD), en la que se utiliza otra condición de parada y un gradiente distinto, el resto es totalmente igual que el algoritmo SGD utilizado en la primera práctica.

El LGR para de iterar en el momento en que converge, es decir, que la distancia de error entre los pesos actuales y los pesos de la anterior iteración sea menor que cierta cantidad de error, ya que en el momento en que la distancia es muy pequeña quiere decir que el algoritmo está muy cerca de una buena solución.

El gradiente en este algoritmo se calcula con la siguiente fórmula:

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

Pero al colocar N=1 de forma fija, la fórmula se simplifica, y se obtiene la siguiente fórmula:

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

En esta fórmula se puede apreciar porque es importante asignar a los datos etiquetas que sean contrarias, porque en la fórmula se tiene en cuenta el valor de la etiqueta (y_n), por lo que el valor de una etiquetas deberá ser equivalente pero de signo contrario al de la otra etiqueta, para que sean proporcionales los cálculos.

Y otra parte importante del ejercicio es la fórmula para obtener el error fuera de la muestra. En mi caso obtengo este error en forma de accuracy, que es el número de elementos bien clasificados.

Para calcular el accuracy sólo hace falta obtener la proporción de elementos bien clasificados.

Ahora toca ver los resultados del ejercicio. El ejercicio consiste en obtener el error medio fuera de la muestra que se obtiene con un modelo que aprende a partir de una muestra de 100 elementos.

Un ejemplo de recta aprendida con este algoritmo es la siguiente:

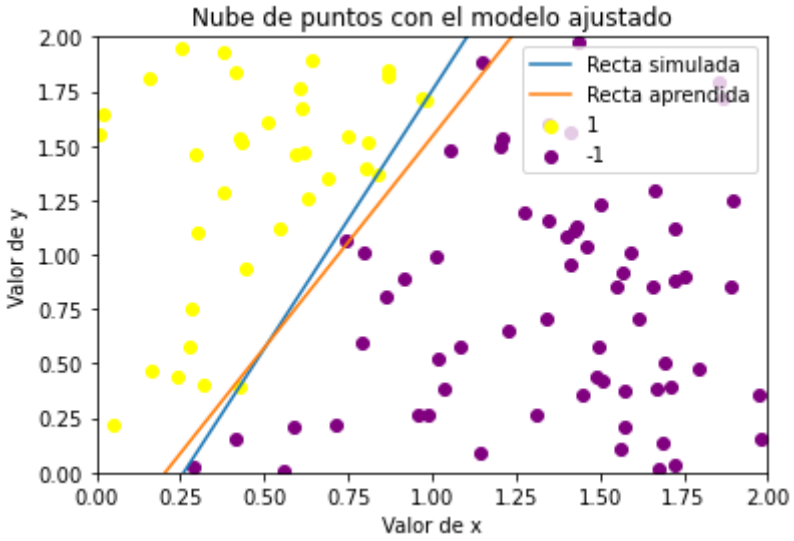


Figura 5: Ejercicio 2.2: Ejemplo de ajuste con LGR

El valor del error medio fuera de la muestra es el siguiente:

- Accuracy: 98.269 %

Como se puede ver, el accuracy no llega a ser del 100 %, esto es debido a la mala clasificación de los puntos que se encuentran en el límite entre la región positiva y la región negativa.

Y conforme más densa es la nube de puntos mayor probabilidad hay de que algún punto se encuentre en esta zona difícil de clasificar.

En la siguiente gráfica se puede apreciar esa zona de límite, que se crea entre la recta simulada, que es la que etiqueta los datos; y el modelo aprendido, que intenta ajustarse para obtener el menor error.

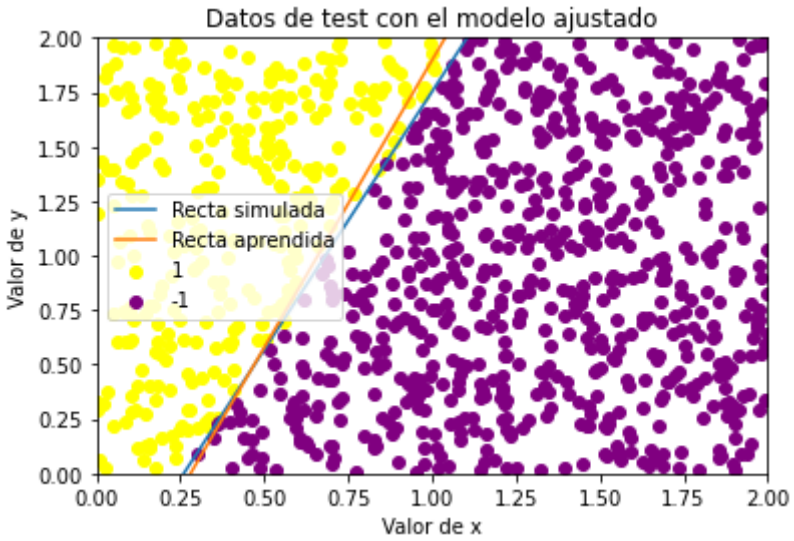


Figura 6: Ejercicio 2.2: Ejemplo de datos de test con la recta aprendida

3. Ejercicio Bonus: Clasificación de Dígitos

3.1. Ejercicio 1

El problema planteado es un problema de clasificación binaria, es decir, el problema consiste en decidir que elementos son de cierta clase y cuáles de la clase contraria. En nuestro caso tenemos que decidir si los dígitos de la muestra se parecen al dígito cuatro, ó al dígito ocho. Para conseguir esta clasificación se debe ajustar un modelo a los datos d entrada y obtener una función g que sea un buen clasificador fuera de la muestra.

Los datos de entrada son el conjunto de datos de los dígito manuscritos. Las características de cada dígito que son tenidas en cuenta a la hora de la clasificación, son la intensidad promedio y la simetría del dígito manuscrito, estos son los elementos del vector de características usado en el aprendizaje.

Elementos del problema:

- $\mathcal{X}$ = intensidad promedio y la simetría de los dígitos manuscritos
- $\mathcal{Y}$ = clase del dígito 4 y clase del dígito 8

Los datos de entrada para el training son los siguientes:

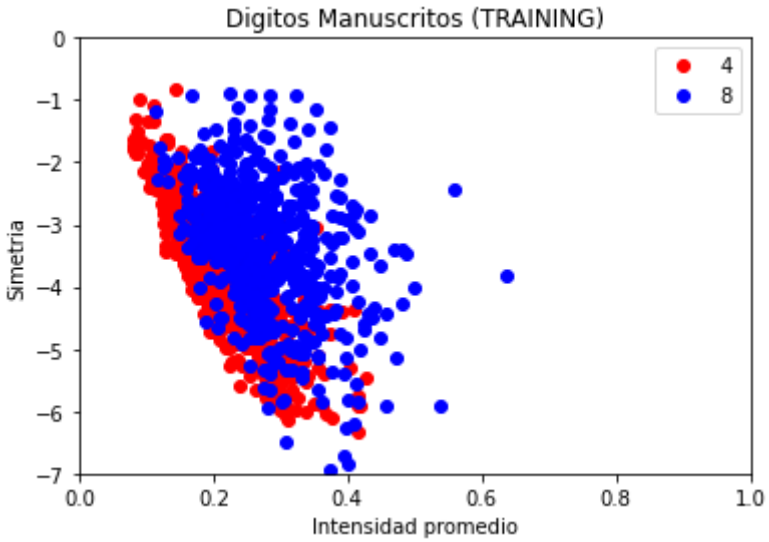


Figura 7: Ejercicio Bonus: Datos de training

Los datos de entrada para el test son los siguientes:

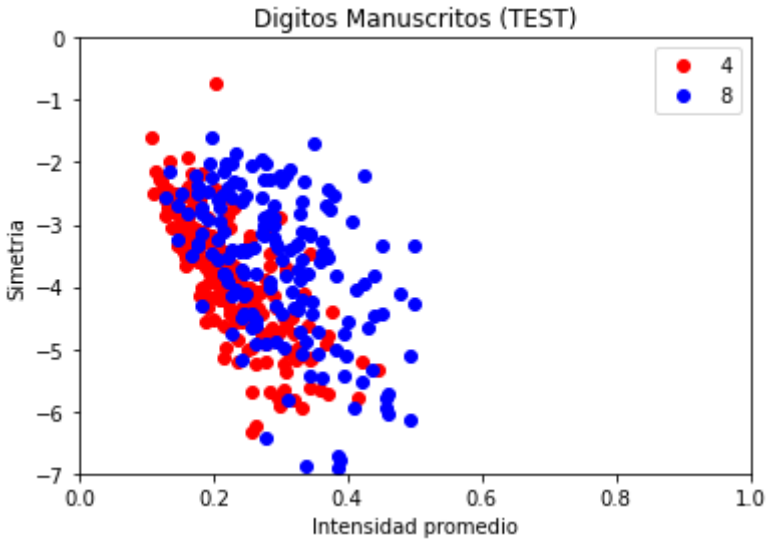


Figura 8: Ejercicio Bonus: Datos de test

3.2. Ejercicio 2

Para este ejercicio el modelo se ha obtenido de diferentes maneras. Se han utilizado los siguientes algoritmos:

- Regresión lineal (SGD)
- Algoritmo Pocket
- Algoritmo SGD + Pocket

3.2.1. SGD

Este algoritmo ya fue usado en la Práctica 1, y para este ejercicio se ha utilizado la misma implementación. El gradiente que se utiliza es el error cuadrático.

3.2.2. Pocket

Esta algoritmo es una variante del algoritmo PLA. En este caso no se devuelven los pesos calculados en la última iteración realizada, sino que se devuelven los mejores pesos encontrados hasta el momento.

Para obtener los mejores pesos, en cada iteración en que se modifican los pesos, debido a una mal predicción, se comprueba si los nuevos pesos son mejores a los que ya se tienen, y si son mejores se actualizan los mejores pesos y su accuracy, que es la medida con la que se comparan los pesos. El resto del algoritmo funciona exactamente igual que el PLA.

Lo bueno de esta variante es que da igual el momento en que se pare el algoritmo, ya que siempre devolverá la mejor solución encontrada, cosa que no pasa con PLA; pero esta ventaja conlleva un sobre coste en cada iteración, con la comprobación de los mejores pesos. También se podría realizar una época entera y en ese momento comprobar si alguno de los pesos obtenido en esa época mejoran a los actuales mejores pesos.

Para la comparación se ha tomado como pesos iniciales un vector de ceros en ambos algoritmos.

En las siguientes gráficas se puede ver una comparación de la evolución del accuracy durante el algoritmo, con el algoritmo PLA y el algoritmo Pocket.

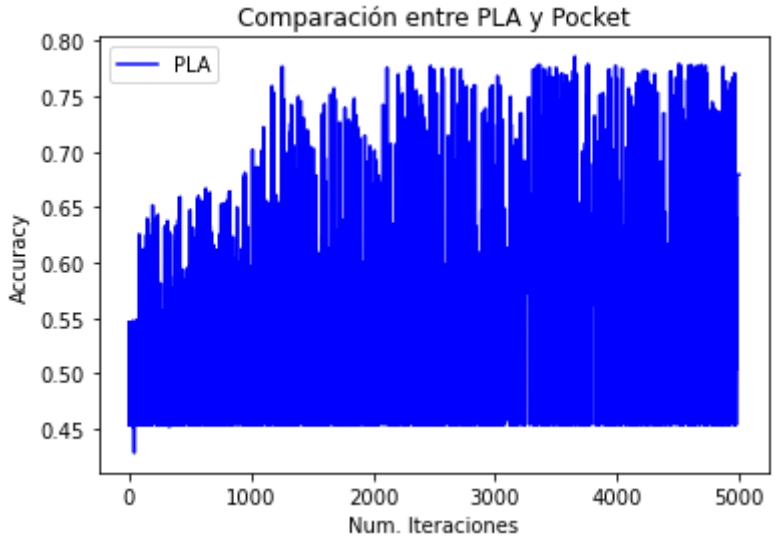


Figura 9: Ejercicio Bonus: Comparación, Modelo con PLA

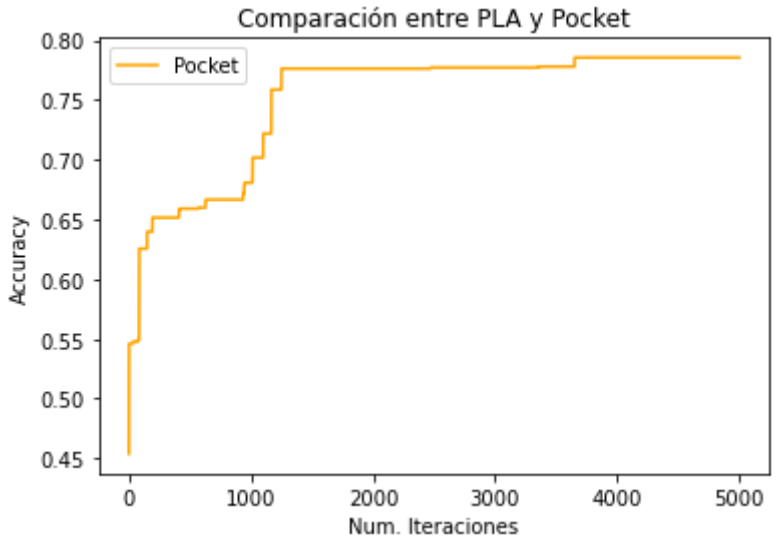


Figura 10: Ejercicio Bonus: Comparación, Modelo con Pocket

Como se puede ver en la gráfica [10] el crecimiento del accuracy es monótono, mientras que en la gráfica [9] el crecimiento sufre oscilaciones, debido a posibles cambios en los pesos que deshacen cambios positivos realizados en las anteriores iteraciones.

Además se puede ver que con el algoritmo Pocket a partir de más o menos la iteración 1250, la mejora en el accuracy es muy pequeña, por lo que si el máximo de iteraciones hubiera sido 1300, se hubiera obtenido un accuracy muy cercano al obtenido con 5000, esta es una ventaja del Pocket.

3.2.3. SGD + Pocket

Un posible añadido al algoritmo Pocket, podría ser obtener el valor inicial de los pesos con el algoritmo SGD, que es muy rápido. Esto no permite empezar en una mejor posición que si lo hiciésemos con un valor al azar.