Momento de Retroalimentación

Módulo 2. Análisis y Reporte sobre el desempeño del modelo. (Portafolio Análisis)

Implementación de un Perceptrón

Profesor: Jorge Adolfo Ramírez Uresti

Materia: Inteligencia Avanzada para la ciencia de datos I

Alumno: Luis Eduardo González Quiroz

Matrícula: A01751188

Fecha: 13/Sept/2022

I. Introducción

Para este proyecto utilicé un algoritmo llamado *Perceptrón*. La idea detrás de un perceptrón se basa en un conjunto de entradas, pesos, una neurona, una función de activación y la idea del ajuste de pesos a través de un mecanismo de *backpropagation*. Primera alimentamos al algoritmo con nuestras entradas. A cada una de estas entradas se les asociará un *peso* (definido inicialmente de forma aleatoria) y de esta forma se creará una combinación lineal a la que al final se le sumará un bias igualmente definido inicialmente de forma aleatoria.

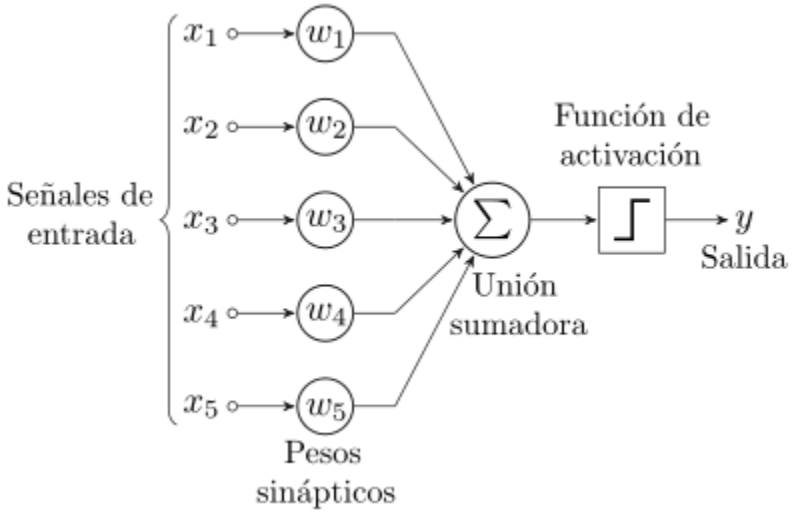

Fig.1 – Diagrama de arquitectura de un perceptrón.

Así, nuestra neurona será una función que tome nuestras entradas *x*, pesos *w* y un bias *b* y los transforme en una combinación lineal de la siguiente forma:

$$z = x_1 * w_1 + x_2 * w_2 + \cdots + x_n * w_n + b$$

Una vez calculada nuestra suma ponderada z, se manda este valor a nuestra función de activación. Como su nombre lo indica, ésta es una función que toma nuestra suma ponderara z, y la transforma en un valor de tal forma que nos permita crear discriminaciones más claras en nuestros valores para así poder llegar a una clasificación.

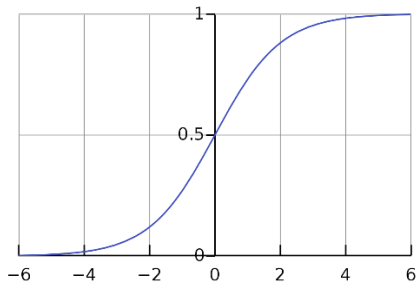

Fig. 2 – Función sigmoide

En este caso, el objetivo de nuestro algoritmo era crear un clasificador binario, por lo que buscamos una función de activación que pueda discriminar nuestras sumas ponderadas en dos categorías. En particular, dos ejemplos clásicos de este tipo de funciones de activación son las funciones escalonadas, y las funciones sigmoides. Para este proyecto se utilizó una función sigmoide pues se consideró que una función sigmoide representa más apropiadamente el tipo de respuesta que se busca obtener al poder más libremente dar resultados con distintos grados de confianza en vez de simplemente entregar un resultado binario con una función escalonada.

Finalmente, el paso más importante al comprender el algoritmo de clasificación de un perceptrón es la noción del ajuste de pesos. Como ya se mencionó, inicialmente tanto los pesos como el bias se definen de forma aleatoria, de esta forma es casi seguro que el resultado de nuestra suma ponderada y por tanto de nuestro perceptrón en general será bastante aleatorio y sin sentido. Por ello, es claro que es necesario hacer un ajuste de los pesos y el bias hasta de tal forma que logremos que la suma ponderada y nuestro resultado final sea consistente con nuestros valores esperados. Esto se hace a través del algoritmo de backpropagation. De forma general la idea es calcular los errores de pruebas anteriores y utilizar el gradiente para calcular la dirección y magnitud en la que debemos modificar nuestros pesos con el fin de minimizar dicho error.

Este proceso se repite a medida que se mejora el accuracy y se disminuye el error hasta llegar a un punto en donde consideremos que nuestro accuracy es lo suficientemente bueno.

II. Desarrollo

Para este desarrollo se utilizó un dataset generado inicialmente con la biblioteca de Scikit-Learn. Este fue el único elemento del código en el que se utilizó código externo. En particular, generamos un dataset con datos divididos en dos centroides que más tarde usaríamos como nuestras dos clases a clasificar. El dataset tiene 250 elementos, y cada elemento está definido por dos componentes o features, por lo que nuestro Perceptrón tomará dos valores x como entradas. Además, los elementos de nuestro dataset están distribuidos con una desviación estándar de 1.05. El dataset tiene dos elementos principales, nuestras X's (entradas) y su respectiva Y (salidas).

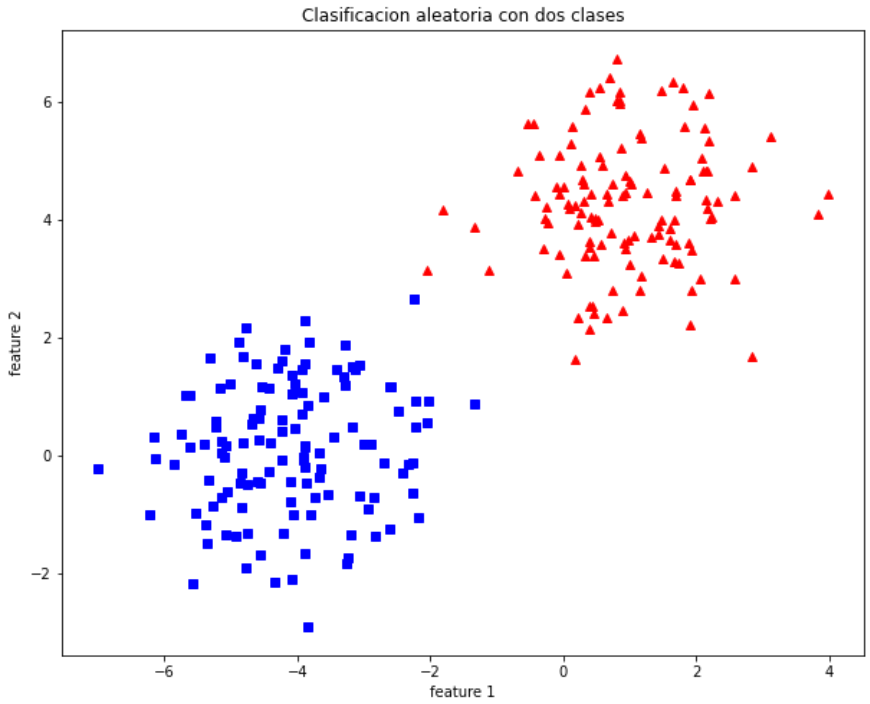

Fig. 3 – Distribución de los elementos del dataset.

Una vez que definimos el dataset, es importante dividirlo en un conjunto de entrenamiento, y otro de testing. Si bien en una primera fase de desarrollo se limitó a tomar el ultimo decil de los datos para testing y el resto para entrenamiento, en la versión final se utilizó una técnica de validación cruzada, en la que se divide el dataste en n número de subdatasets para después proceder a iterar por cada uno utilizando en cada caso el subdataset a utilizar como conjunto de testing. En este proyecto se utilizó esta técnica de validación cruzada con 10 subdivisiones del dataset y finalmente se obtuvo el promedio del accuracy de los 10 tests.

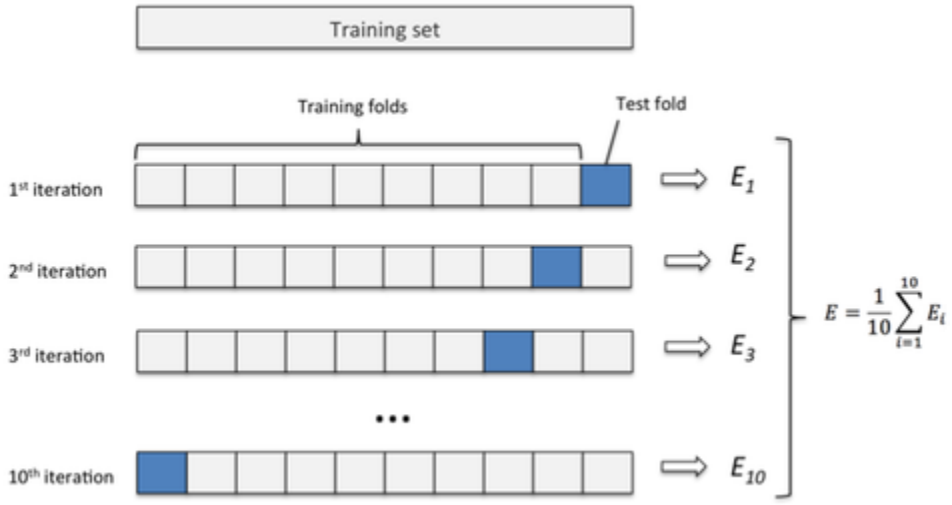

Fig. 4 – Diagrama de técnica de validación cruzada

En adición a la técnica anteriormente mencionada de validación cruzada, también se entrenó el modelo en diferentes etapas con diferentes valores de *epochs.* Esto, con el objetivo de identificar los efectos del overfitting u underfitting. Así, se entrenó el modelo con 1 y hasta 100 epochs (variando de 10 en 10).

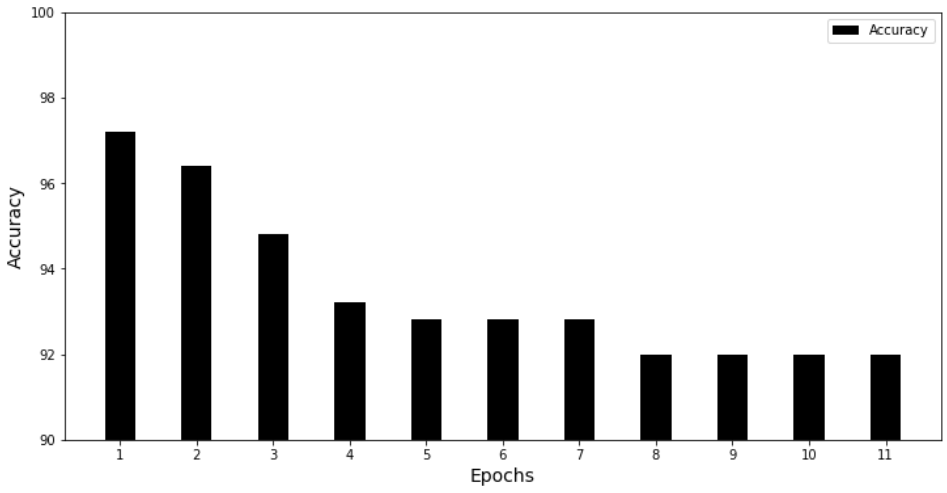

Fig. 5 – Variación del accuracy con diferentes apochs(x10).

Como se puede apreciar en la Fig.5., desde el primer epoch, el algoritmo logra un accuracy de 97%, sin embargo, conforme aumentamos el número de epochs, disminuye el accuracy. Esto se debe precisamente al anteriormente mencionado overfitting, pues conforme aumentamos el numero de epochs, el algoritmo empieza a "aprender de memoria" las características especificas del dataset de training, eclipsando así los aprendizajes y generalizaciones que había logrado extraer previamente por lo que el accuracy en el dataset de testing disminuye. Por tanto, a

partir de dicho análisis se concluyó que el hiperparametro de epoch = 1 sería el mas apropiado para continuar el desarrollo del clasificador.

Finalmente, tomando el hiperparametro de epoch = 1, se corrió nuevamente el algoritmo para estimar la desviación estándar y el error entre cada uno de nuestros K-Folds durante la validación cruzada. Así obtuvimos los siguientes resultados:

```
Accuracy promedio: 97.2%
Desviación estandar: 3.7947
Varianza: 14.4
```

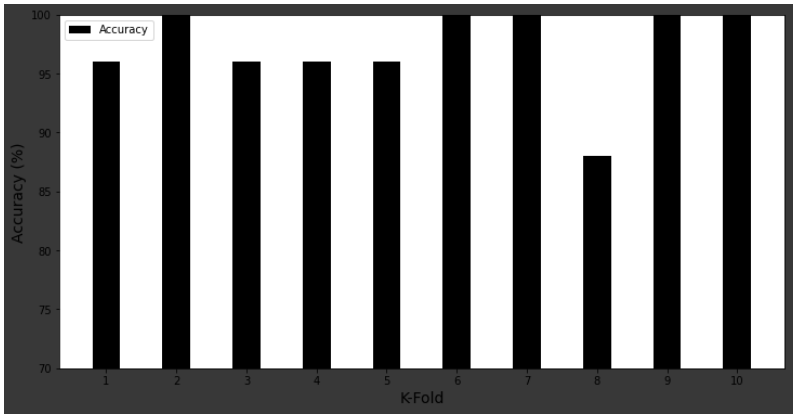

Fig 6 - Distribución del accurracy para cada K-Fold durante validación cruzada.

Con la información anterior podemos ver que durante la validación cruzada obtuvimos accuracies distribuidos con una desviación estándar de 3.7 lo que nos indica que sí hay una varianza relativamente amplia en nuestros resultados, pero no lo suficiente como para invalidar los mismos.

III. Conclusión

Como primera conclusión se puede mostrar que este algoritmo de Perceptrón tuvo un accuracy promedio de 97% lo cual es excelente especialmente tomando en cuenta que solo requerimos un epoch para lograr dichos resultados. Al mismo tiempo, es muy importante recalcar la importancia de entender que más epochs no son equivalentes a un mejor desempeño del modelo, pues como vimos en este caso, desde el primer epoch obtuvimos nuestro mejor accuracy, y a partir de ahí comenzó un proceso de overfitting que empeoró el desempeño del modelo. Finalmente es también importante entender las limitaciones del modelo, por ejemplo, un Perceptrón solo puede clasificar un conjunto de datos donde nuestras categorías puedan ser divididas por un hiperplano. Y en particular, esta implementación de Perceptrón solo funciona con conjuntos de dos clases.

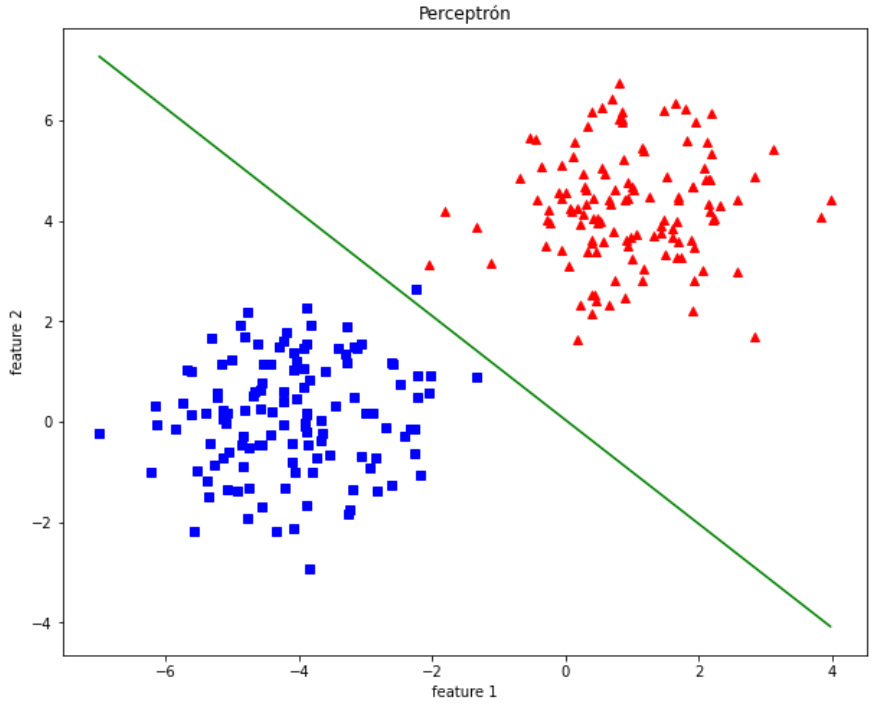

Fig. 7 – En la gráfica se muestra el hiperplano que genera el Perceptrón dividiendo el dataset en dos clases, así como algunos puntos del mismo que fueron erróneamente clasificados.