Inteligencia artificial avanzada para la ciencia de datos
Momento de Retroalimentación: Módulo 2 Análisis y Reporte sobre el desempeño del modelo. (Portafolio Análisis)

Kathia Bejarano Zamora
A01378316

11 de Septiembre 2023

Instituto Tecnológico y de Estudios Superiores de Monterrey.
Campus Estado de México.

"Yo, como integrante de la comunidad estudiantil del Tecnológico de Monterrey, soy consciente de que la trampa y el engaño afectan mi dignidad como persona, mi aprendizaje y mi formación, por ello me comprometo a actuar honestamente, respetar y dar crédito al valor y esfuerzo con el que se elaboran las ideas propias, las de los compañeros y de los autores, así como asumir mi responsabilidad en la construcción de un ambiente de aprendizaje justo y confiable."

PERCEPTRON

Un perceptrón es una neurona artificial, es fundamental para las redes neuronales. Basicamente lo que se busca es simular las neuronas de un ser humano, SI recordamos el funcionamiento tenemos las ramificaciones, que se encargan de recibir toda la información y trasladarla a los núcleo que hace todo el tratado de esta información y finalmente sale la información ya tratada.

Lo que hace la neurona artificial es imitar lo descrito anteriormente pero basada en una función matemática, cada neurona recibirá datos, los pesa, calcula la suma y obtiene un resultado.

A continuación muestro una imagen en donde se ve de manera gráfica una neurona y su comparativa con la neurona aritficial.

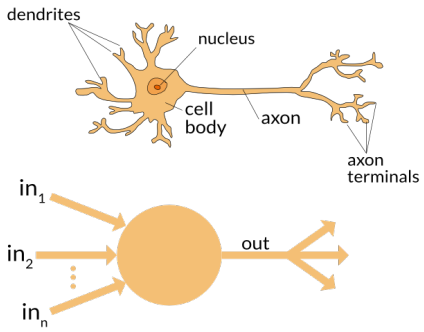


Imagen 1. Perceptron

https://appliedgo.net/perceptron/

El perceptrón se forma por distintos componenetes, que a pesar de ya haberlos visto de manera gráfica es importante describirlos uno a uno.
Input (Entrada): En este caso las entradas serán las características del conjunto de datos.
Pesos(w): Son valores preliminares y con el paso del aprendizaje se van ajustando, basicamente son quienes nos irán dictando la mejora y una buen apredicción
Función de activación: Es la función que se realiza dentro de la neurona que muy comunmente es la sigmoidal.
Output(salida): Suma ponderada que pasó por la activación y será el valor obtenido apartir de este flujo.

A continuación hablaremos de manera más especóifica sobre el trabajo realizado.

BIAS, ACCURACY Y VARIANCE

Al ser un modelo tenemos métricas de validación que nos ayudan a interpretar nuestros resultados. A continuación describire los 3 fundamentales.

El primero de ellos es el “bias”, o sesgo en español, es una medición de la exactitud y representa ese error sistemático del modelo. Su principal parámetro es el error. Su interpretación se da de la siguiente manera:

Si lo obtenido es alto, el modelo es simple y existe una posibilidad de underfitting. Lo que se busca es un sesgo lo más bajo posible.

El segundo de ellos es “accuracy” o precisión, es el cocientre entre los verdaderos positivos y las predicciones positivas producidas en el modelo. (algo que se ve muy bien en la primer entrega de mi perceptron manual). En este caso el valor esperado debe ser 1.

Finalmente tenemos a la varianza, concepto de estadística, representa la variabilidad de una serie de datos, con respecto su promedio.

SET DE DATOS

Para la selección del set de datos utilizamos el set de datos Iris, que provee directamente sklearn. Este set de datos se escogió principalmente por los datos numéricos que contiene.

-El análisis del dataset se genera a partir de las especies de iris tales: setosa, versicolor y virginica.

-El conjunto consta de 150 muestras, por lo que es un buen npumero para el análisis por medio del preceptrón.

-Existen cuatro características en el conjunto de datos.

-Tenemos muy claramente definidos nuestros targets que se mencionaron anteriormente.

A continuación muestro un preview del set de datos.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Imagen 1.1 Preview Iris

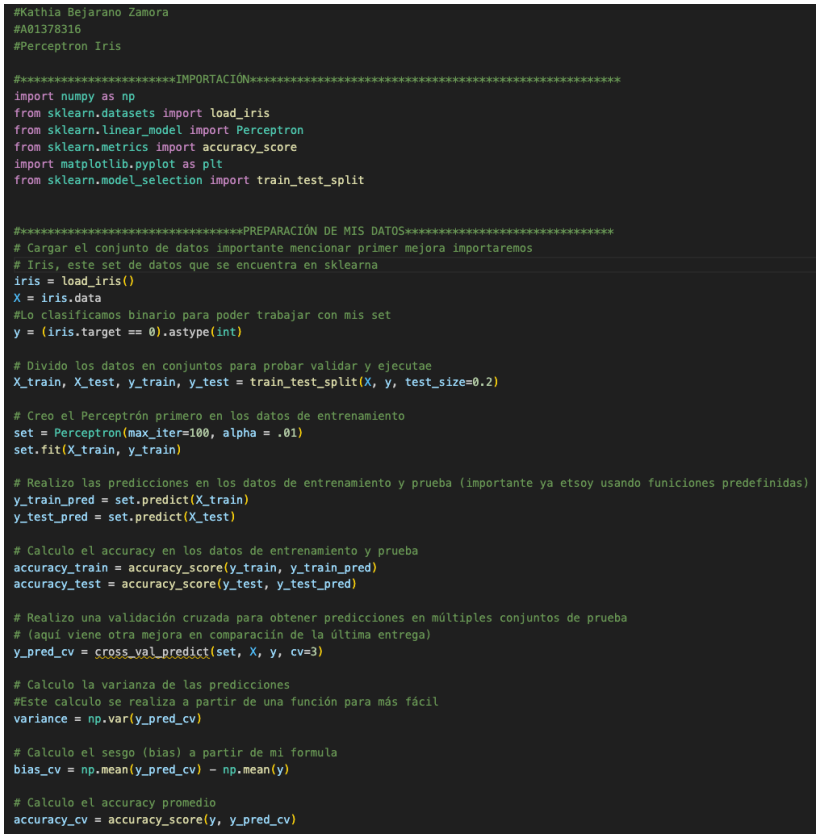
https://www.kaggle.com/code/kostasmar/exploring-the-iris-data-set-scikit-learn

CÓDIGO

Para esta primer implementación del código utilizamos sklearn para básicamente realizar todo el análisis. Importante mencionar que para el llamado de la función Perceptron definimos el número de iteraciones y alpha.

Basicamente el npumero de iteraciones nos ayuda a las corridas y el alpha es ese ajuste de pesos que se mencionó en la introducción.

A diferencia de la entrega previa, este código ya corre con el set de datos bien seleccionado por lo que pudimos ver muchas mejoras.



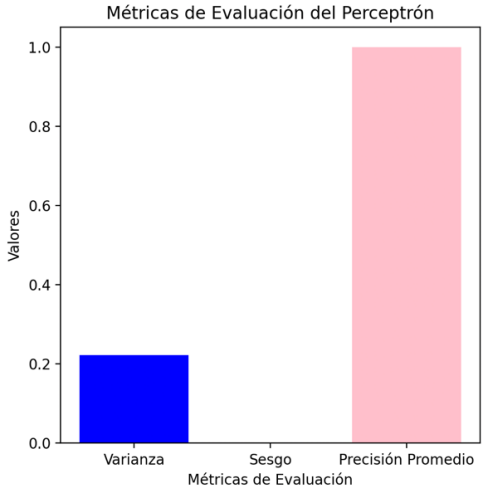
```python
#Kathia Bejarano Zamora
#A01378316
#Perceptron Iris

#**********************IMPORTACIÓN*************************************************************
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


#*******************************PREPARACIÓN DE MIS DATOS*****************************
# Cargar el conjunto de datos importante mencionar primer mejora importaremos
# Iris, este set de datos que se encuentra en sklearna
iris = load_iris()
X = iris.data
#Lo clasificamos binario para poder trabajar con mis set
y = (iris.target == 0).astype(int)

# Divido los datos en conjuntos para probar validar y ejecutae
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Creo el Perceptrón primero en los datos de entrenamiento
set = Perceptron(max_iter=100, alpha = .01)
set.fit(X_train, y_train)

# Realizo las predicciones en los datos de entrenamiento y prueba (importante ya etsoy usando funiciones predefinidas)
y_train_pred = set.predict(X_train)
y_test_pred = set.predict(X_test)

# Calculo el accuracy en los datos de entrenamiento y prueba
accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)

# Realizo una validación cruzada para obtener predicciones en múltiples conjuntos de prueba
# (aquí viene otra mejora en comparaciín de la última entrega)
y_pred_cv = cross_val_predict(set, X, y, cv=3)

# Calculo la varianza de las predicciones
#Este calculo se realiza a partir de una función para más fácil
variance = np.var(y_pred_cv)

# Calculo el sesgo (bias) a partir de mi formula
bias_cv = np.mean(y_pred_cv) - np.mean(y)

# Calculo el accuracy promedio
accuracy_cv = accuracy_score(y, y_pred_cv)
```

Imagen 1.3

"PerceptronIris. Py

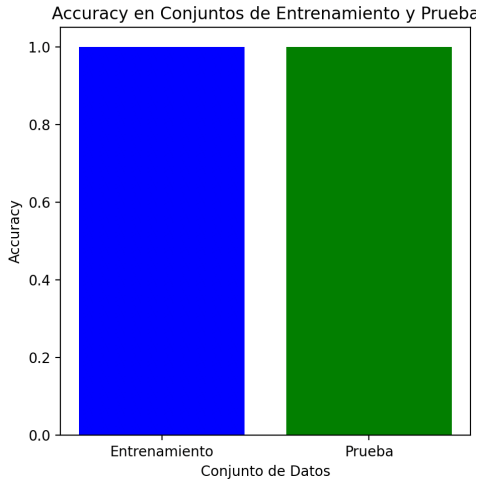
GRÁFICAS DE EVIDENCIA

Para este código podemos observar que realmenet el códgio es muy bueno. EL resultado arrojado muestra las siguientes gráficas:



Gráfica 1.

PerceptronIris.py



Gráfica 1.1

PerceptronIris.py

Como podemos observar el código es bastante bueno. Tenemos una congruencia bastante buena entre la precisión de los datos de prueba y los datos de entrenamiento. Además de eso podemos observar que no tenemos sesgo, y tenemos un 0.222 de varianza.

CÓDIGO MEJORADO

Fue bastante díficil mejora nuestros resultados, debido a la gran interpretación obtenida, sin embargo tomamos la decisión de hacer cambios puntuales y a continuación los describiré.
Primero implementamos un método que se denomina bootstrap en donde basicmanete lo que se hace es que toma muchas muestras (en est ecaso pusimos 100) aleatorias a partir del conjuitno de datos dado y empieza a entrenar sobre estos, de tal manera que se aplica el perceptron a cada una de las muestras y finalmente para la obtención de las métricas de evaluación se hace un promedio.
Además de eso añadp 3 claras diferencias a la hora de llamar a mi función de perceptron que son: definir el númeor de iteraciones a 1000, definir mi alpha con .001, y mi random state 42, npumeor óptimo para la estandarización de la sálida.

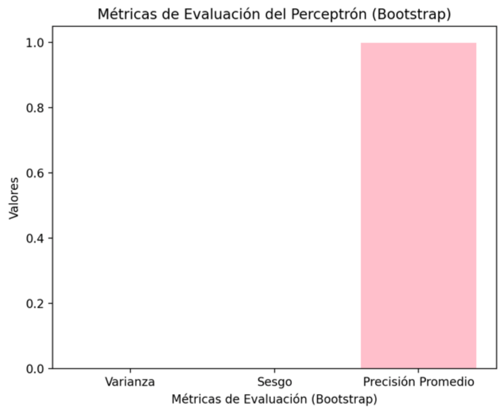
```python
#Kathia Bejarano Zamora
#A01378316
#Perceptron Iris en teoría mejorado

#**********************IMPORTACIÓN*******************************************************
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

#*********************************PREPARACIÓN DE MIS DATOS******************************
# Cargar el conjunto de datos Iris de ejemplo
iris = load_iris()
X = iris.data
y = (iris.target == 0).astype(int)

#la parte nueva del código incluye lo que se puede conocer como bootstrap
#VAmos a hacer muchas replicas para entrenar más
n_replicas = 100

# Almaceno las predicciones de cada réplica
y_pred_replicas = []
```

```python
# Creo y entreno un Perceptrón para cada réplica
for _ in range(n_replicas):
    # Creo un conjunto de entrenamiento
    indices_bootstrap = np.random.choice(len(X), len(X), replace=True)
    X_bootstrap = X[indices_bootstrap]
    y_bootstrap = y[indices_bootstrap]

    set = Perceptron(random_state=42,max_iter = 1000, alpha =.001) #Random state mejora
    set.fit(X_bootstrap, y_bootstrap)

    # Realizo predicciones en el conjunto original
    y_pred = set.predict(X)
    y_pred_replicas.append(y_pred)

# Calculo la varianza de las predicciones
variance_bootstrap = np.var(y_pred_replicas, axis=0)

# Calculo el sesgo (bias) promedio de todo
bias_bootstrap = np.mean(y_pred_replicas, axis=0) - np.mean(y)

# Calculo la precisión promedio
accuracy_bootstrap = [accuracy_score(y, y_pred) for y_pred in y_pred_replicas]
accuracy_avg_bootstrap = np.mean(accuracy_bootstrap)
```

Imagen 2

"PerceptronIrisMejora. Py

GRÁFICAS DE EVIDENCIA



Gráfica 1.

PerceptronIrisMejora.py

Podemos observar un comportamiento de manera muy clara y es que nuestra varianza se redujo sin embargo para entender exactamente qué paso decidí imprimir las varianzas de nuestras pruebas y obtenemos lo siguiente:

```
[0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.0564 0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.    ]
>>> print(np.mean(variance_bootstrap))
0.0003760000000000003
```

Imagen 2.1

"PerceptronIrisMejora. Py

Hubo una clara reducción en la varianza lo cual nos indica que los datos están saliendo muy cercanos e incluso iguales.

Algo que es importante mencionar es que la técnica implementada puedo ser a favor o en contra pues juega mucho con el aumento de datos y la aleatoridad.

CONCLUSIONES

El Perceptrón, a pesar de su simplicidad, es un algoritmo esencial en el aprendizaje automático, especialmente en tareas de clasificación binaria. Al explorar sus características clave, se pueden destacar los siguientes aspectos:

1. Facilidad de Uso: El Perceptrón se distingue por su sencillez. Opera realizando una combinación lineal de las entradas y aplicando una función de activación de tipo escalón para tomar decisiones de clasificación.

2. Limitaciones Fundamentales: Es importante reconocer que el Perceptrón presenta limitaciones considerables. Su capacidad se restringe a la resolución de problemas que pueden separarse linealmente, lo que significa que no puede manejar situaciones donde la división de clases requiere una frontera no lineal. Esto lo restringe en la resolución de problemas más complejos.

3. Impacto de la Tasa de Aprendizaje: La elección adecuada de la tasa de aprendizaje es esencial para garantizar que el Perceptrón converja de manera efectiva y aprenda patrones de datos de manera apropiada. Una tasa de aprendizaje muy alta puede impedir la convergencia, mientras que una tasa demasiado baja puede ralentizar el proceso de aprendizaje.

En resumen, el Perceptrón es un algoritmo de clasificación elemental pero crucial en el ámbito del aprendizaje automático, especialmente en la tarea de clasificación binaria. No obstante, es vital tener en cuenta sus restricciones, ya que solo puede abordar problemas con separación lineal, lo que lo hace insuficiente para tareas más complejas. Por ende, este algoritmo ha servido como punto de partida en el desarrollo de modelos de aprendizaje automático más avanzados, como las redes neuronales profundas, diseñadas para enfrentar desafíos más complejos y variados en el procesamiento de datos.

REFERENCIAS

1.- *Perceptrons - the most basic form of a neural network*. (2016, 9 junio). Applied Go. https://appliedgo.net/perceptron/

2.- González, L. (2022). ¿Qué es el perceptrón? Perceptrón simple y multicapa. 🤖 *Aprende IA*. https://aprendeia.com/que-es-el-perceptron-simple-y-multicapa/

3.- Romero, I. (2021, 4 marzo). *La dicotomía sesgo-varianza en modelos de machine learning - Keepler | Cloud Data Driven Partner*. Keepler | Cloud Data Driven Partner. https://keepler.io/es/2021/03/la-dicotomia-sesgo-varianza-en-modelos-de-machine-learning/

4.- López, J. F. (2022). Varianza. *Economipedia*. https://economipedia.com/definiciones/varianza.html

5.- Kostasmar. (2021). Exploring the Iris data set - Scikit-Learn. *Kaggle*. https://www.kaggle.com/code/kostasmar/exploring-the-iris-data-set-scikit-learn