# Apuntes Semana 6

Apuntes del 11 de setiembre de 2025
Andrey Ureña Bermúdez – 2022017442
Inteligencia Artificial
andurena@estudiantec.cr

***Resumen*—En este documento, se resume la clase del 11 de setiembre de 2025, en la cuál se realizó primeramente un repaso de lo visto en la clase anterior. De manera general, este documento recopila información sobre verosimilitud en la regresión logística, la función de costo e información sobre un notebook de regresión logística compartido por el profesor.**

***Index Terms*—Verosimilitud, regresión logística, gradiente descendiente, función sigmoide, derivada.**

## I. Nota sobre Tarea I

Se hace recordatorio sobre darle importancia y no descuidar el trabajo escrito de la tarea, así como su documentación, pues de este se dará el *feedback* para los escritos que haya que realizar en tareas próximas y etapas del proyecto.

## II. Repaso sobre clase del martes

### II-A. Verosimilitud

Es la probabilidad de observar cada uno de los datos cambiando ciertos parámetros. Lo que se busca es maximizar para llegar al punto de máxima probabilidad.

La diferencia entre MSE y *maximum likelihood* radica en su aplicación: para la predicción de valores continuos, se utiliza MSE, mientras que para modelar probabilidades, se utiliza *maximum likelihood*.

Así, nuestra función de costo es:

$$L = \prod_{i=1}^{N} f_{w,b}(x_i)^{y_i} \cdot (1 - f_{w,b}(x_i))^{(1-y_i)} \tag{1}$$

Se vió el desarrollo de cada uno de los casos de $y_i$ en $f_{w,b}(x_i)^{y_i}(1 - f_{w,b}(x_i))^{(1-y_i)}$:

***Caso*** $y_i = 1$*:*

$$f_{w,b}(x_i)^{y_i}(1 - f_{w,b}(x_i))^{(1-y_i)} = f_{w,b}(x_i)^1(1 - f_{w,b}(x_i))^0 \tag{1}$$

$$= f_{w,b}(x_i)^1 \tag{2}$$

Acá el modelo nos da el valor directo. Ejemplo: **Calabaza no es naranja**:

- $wx + b = 1{,}458$
- $f_{w,b}(x_i) = \sigma(wx_i + b) = \sigma(1{,}458) = 0{,}81$
- $= f_{w,b}(x_i)^1(1 - f_{w,b}(x_i))^0$
- $= f_{w,b}(x_i)^1 = \sigma(1{,}458)$
- $= 0{,}81$

La probabilidad de que $x_i$ no sea naranja es $0{,}81$.

***Caso*** $y_i = 0$*:*

$$f_{w,b}(x_i)^{y_i}(1 - f_{w,b}(x_i))^{(1-y_i)} = f_{w,b}(x_i)^0(1 - f_{w,b}(x_i))^1 \tag{1}$$

$$= (1 - f_{w,b}(x_i))^1 \tag{2}$$

Con la misma fórmula puedo estudiar de que ocurra o no un evento. Ejemplo: **Calabaza es naranja**:

- $wx + b = -1{,}32$
- $f_{w,b}(x_i) = \sigma(wx_i + b) = \sigma(-1{,}32) = 0{,}21$
- $= f_{w,b}(x_i)^0(1 - f_{w,b}(x_i))^1$
- $= (1 - f_{w,b}(x_i))^1 = (1 - \sigma(-1{,}32))$
- $= (1 - 0{,}21) = 0{,}79$

La probabilidad de que $x_i$ sea naranja es $0{,}79$.

Al final lo que obtenemos es la probabilidad de que la muestra $x_i$ tenga la etiqueta $y_i$.

### II-B. Derivada de la función de costo

Primero, se debe calcular la probabilidad de que $x_i$ tome la etiqueta de $y_i$, así con cada muestra.

Dado que esto implica la multiplicación de probabilidades, el cálculo de la derivada se vuelve complejo. Para simplificarlo, se busca una expresión equivalente que evite la multiplicación, lo cual se logra aplicando logaritmos.

### II-C. Logaritmos

- $\ln(a^n) = n \cdot \ln(a)$
- $\ln(a \cdot b) = \ln(a) + \ln(b)$
- $\ln(a^n \cdot b^n) = n \cdot \ln(a) + n \cdot \ln(b)$

### II-D. Aplicación de logaritmo a la verosimilitud

- $L = \prod f_{w,b}(x_i)^{y_i} \cdot (1 - f_{w,b}(x_i))^{(1-y_i)}$
- $\ln(L) = \sum \ln(f_{w,b}(x_i)^{y_i}) + \ln((1 - f_{w,b}(x_i))^{(1-y_i)})$
- $\ln(L) = \sum y_i \cdot \ln(f_{w,b}(x_i)) + (1 - y_i) \cdot \ln(1 - f_{w,b}(x_i))$

Esto lo vamos a llamar **log-likelihood**. Es mucho más fácil de computar y derivar, además de que quita errores al momento de computar las multiplicaciones de probabilidades. Ahora esta es la función de costo que se va a usar.

Para minimizar maximizando lo que se puede hacer es darle vuelta a la función, para eso se multiplica por $-1$:

$$L = \frac{1}{N} \sum y_i \cdot \ln(f_{w,b}(x_i)) + (1 - y_i) \cdot \ln(1 - f_{w,b}(x_i))$$

$$L = -\frac{1}{N}\left[\sum y_i \cdot \ln(f_{w,b}(x_i)) + (1-y_i)\cdot \ln(1-f_{w,b}(x_i))\right]$$

Ahora puedo minimizar la función, lo que permite aplicar el descenso del gradiente que se ha estado trabajando.

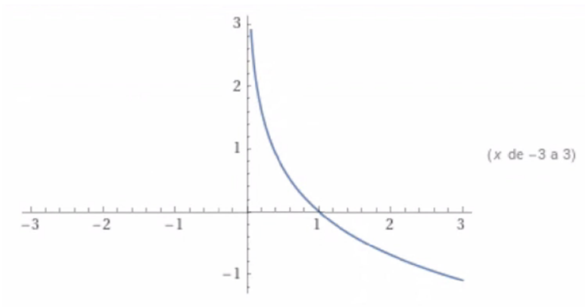



Figura 1. Gráfica minimizando L

Aquí lo ideal es intentar que el *loss* llegue a cero; si se obtiene un *loss* negativo, significa que algo se está haciendo mal.

### II-E. *Actualización de parámetros*

Es necesario actualizar los parámetros $w$ y $b$, ya que son los que permiten modificar los resultados de las probabilidades obtenidas.

- Para actualizar el parámetro $w$ se necesita: $\frac{\partial L}{\partial w}$
- Para actualizar el parámetro $b$ se necesita: $\frac{\partial L}{\partial b}$

### II-F. *Composición de funciones*

Se va a utilizar el concepto de composición de funciones para que el cálculo de derivadas sea más sencillo.

- Derivada función de costo para un sample:

$$L = y_i \cdot \ln(f_{w,b}(x_i)) + (1-y_i)\cdot \ln(1-f_{w,b}(x_i))$$

- Modelo: $f_{w,b}(x) = a(z(x))$
- $a(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
- $z(x) = wx + b$
- El resultado de combinar ambas es:

$$L = y_i \cdot \ln(a(z(x))) + (1-y_i)\cdot \ln(1-a(z(x)))$$

Cuando se habla de la técnica de composición de funciones se aplica la regla de la cadena. Se deben calcular las derivadas parciales:

$$L = y_i \cdot \ln(a(z(x))) + (1-y_i)\cdot \ln(1-a(z(x)))$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\cdot\frac{\partial a}{\partial z}\cdot\frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\cdot\frac{\partial a}{\partial z}\cdot\frac{\partial z}{\partial b}$$

#### 1) *Cálculo de derivadas parciales*

Importante recordar que el $L$ que se está usando es:

$$L = -[y_i \cdot \ln(a(z(x))) + (1-y_i)\cdot \ln(1-a(z(x)))]$$

Primero se inicia calculando la derivada parcial de $L$ con respecto a la función sigmoide:

$$\begin{aligned}\frac{\partial L}{\partial a} &= -\left[\left(y_i \cdot \frac{1}{a(x)}\cdot a(x)'\right) + \left((1-y_i)\cdot\frac{1}{1-a(x)}\cdot(1-a(x))'\right)\right]\\ &= -\left[\left(\frac{y_i}{a(x)}\cdot 1\right) + \left(\frac{(1-y_i)}{1-a(x)}\cdot -1\right)\right]\\ &= -\left[\left(\frac{y_i}{a(x)}\right) - \left(\frac{(1-y_i)}{1-a(x)}\right)\right]\\ &= \frac{-y_i}{a(x)} + \frac{(1-y_i)}{1-a(x)}\end{aligned}$$

Ahora, se calcula la derivada parcial de la función sigmoide respecto a $z$. Importante recordar que la derivada de sigmoide es $\sigma(x)\cdot(1-\sigma(x))$, por lo que la derivada parcial sería:

$$\frac{\partial a}{\partial z} = \sigma(z(x))\cdot(1-\sigma(z(x)))$$

Por último, se debe calcular de manera individual la derivada parcial de cada uno de los parámetros con respecto a la regresión lineal:

$$z(x) = wx + b$$

$$\frac{\partial z}{\partial w} = x$$

$$\frac{\partial z}{\partial b} = 1$$

Ya que se hizo el cálculo de cada derivada de manera individual, se prosigue a realizar las multiplicaciones:

- $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\cdot\frac{\partial a}{\partial z} = \frac{-y_i}{a(z(x))} + \frac{1-y_i}{(1-a(z(x)))}\cdot a(z(x))\cdot(1-a(z(x)))$
- $\frac{\partial L}{\partial z} = \frac{-y_i + a(z(x))\cdot y_i + a(z(x)) - a(z(x))\cdot y_i}{a(z(x)) - a(z(x))^2}\cdot\left[a(z(x)) - a(z(x))^2\right]$
- $\frac{\partial L}{\partial z} = \frac{-y_i + \cancel{a(z(x))\cdot y_i} + a(z(x)) \cancel{- a(z(x))\cdot y_i}}{a(z(x)) - a(z(x))^2}\cdot\left[a(z(x)) - a(z(x))^2\right]$
- $\frac{\partial L}{\partial z} = \frac{-y_i + a(z(x))}{\cancel{a(z(x)) - a(z(x))^2}}\cdot\left[\cancel{a(z(x)) - a(z(x))^2}\right]$
- $\frac{\partial L}{\partial z} = a(z(x)) - y_i$

Figura 2. Derivada parcial de L respecto a z

- $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z}\cdot\frac{\partial z}{\partial w} = (a(z(x)) - y_i)\cdot x$
- $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}\cdot\frac{\partial z}{\partial b} = (a(z(x)) - y_i)\cdot 1$

Figura 3. Derivada parcial de L respecto a w y b

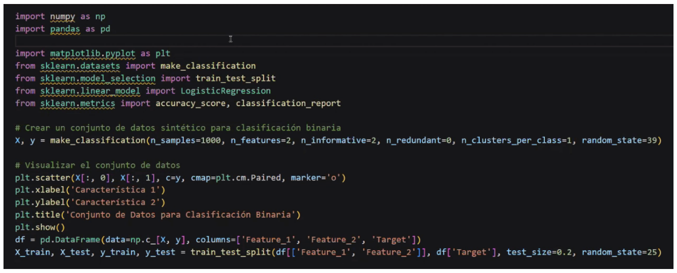
Se procede a actualizar parámetros:

- $z(x) = wx + b$
- $w = w - \alpha \frac{\partial L}{\partial w}$
- $b = b - \alpha \frac{\partial L}{\partial b}$

Donde $\alpha$ es un hiperparámetro (learning rate).

## III. CÓDIGO

Se muestra un notebook con el fin de comprender mejor cómo hacer una regresión logística. Enlace a notebook.



```python
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Crear un conjunto de datos sintético para clasificación binaria
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=39)

# Visualizar el conjunto de datos
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired, marker='o')
plt.xlabel('Característica 1')
plt.ylabel('Característica 2')
plt.title('Conjunto de Datos para Clasificación Binaria')
plt.show()
df = pd.DataFrame(data=np.c_[X, y], columns=['Feature_1', 'Feature_2', 'Target'])
X_train, X_test, y_train, y_test = train_test_split(df[['Feature_1', 'Feature_2']], df['Target'], test_size=0.2, random_state=25)
```

Figura 4. Código Clasificación

Como se muestra en la **figura 4**, se hace la importación de librerías necesarias, muchas de las cuales pertenecen a sklearn. Luego, se hace la clasificación con make_classification, el cual es un método para crear un dataset de clasificación. En este caso, se indica que sea de 1000 samples, con 2 features informativas, sin features redundantes y con un solo clúster por clase.

Posteriormente, se visualiza el conjunto de datos utilizando plt.scatter, donde los puntos se colorean según su clase (y). Luego, se crea un DataFrame con pd.DataFrame que contiene las dos características (Feature_1 y Feature_2) y la variable objetivo (Target).

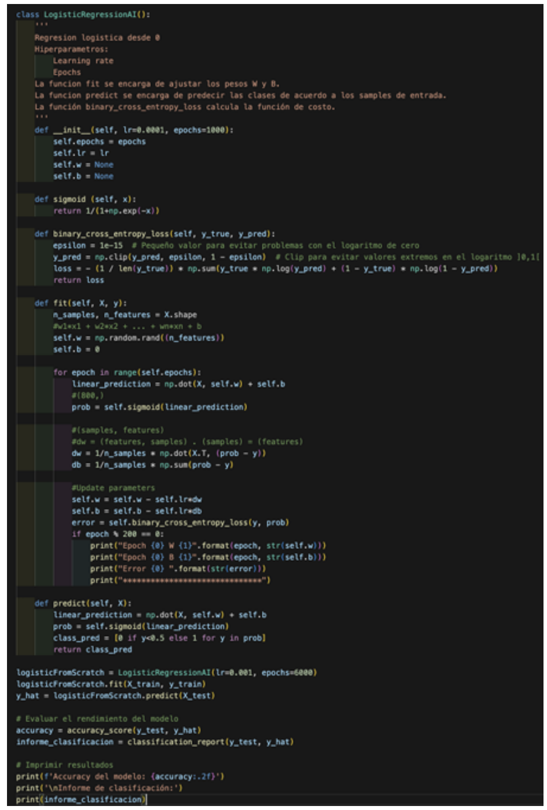
Finalmente, se divide el dataset en entrenamiento y prueba con train_test_split, reservando el 80 % de los datos para entrenamiento y el 20 % para prueba, asegurando reproducibilidad con random_state=225.

La **figura 5** muestra la implementación manual de la regresión logística. La clase recibe como parámetros la cantidad de epochs a ejecutar, el learning rate que se aplicará y los parámetros de la regresión w y b, que serán ajustados durante el entrenamiento.

Primero, se define la función **sigmoide**, utilizada para convertir la predicción lineal en una probabilidad. Luego, se implementa la función de costo binary_cross_entropy_loss, que calcula la pérdida negativa con el objetivo de minimizarla durante el entrenamiento.

En la función **fit**, se reciben todos los features y las etiquetas correspondientes. Antes de iniciar el ajuste de los parámetros, se inicializan aleatoriamente los valores de w, cuyo tamaño corresponde al número de features, ya que cada uno necesita un peso asociado. Luego, se ejecuta el ciclo de entrenamiento por la cantidad de epochs definida, donde primero se calcula la predicción lineal, que luego pasa por la función sigmoide para obtener una probabilidad. A partir de esta probabilidad, se actualizan los valores de w y b, y se calcula el error en cada iteración.

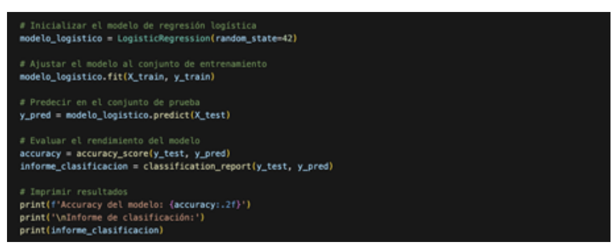
La función **predict** se encarga de predecir la clase de una nueva muestra en base al umbral que se define. Una vez obtenida la probabilidad, se asigna la clase correspondiente.



```python
class LogisticRegressionAI():
    '''
    Regresion logistica desde 0
    Hiperparametros:
        Learning rate
        Epochs
    La funcion fit se encarga de ajustar los pesos W y B.
    La funcion predict se encarga de predecir las clases de acuerdo a los samples de entrada.
    La función binary_cross_entropy_loss calcula la función de costo.
    '''
    def __init__(self, lr=0.0001, epochs=1000):
        self.epochs = epochs
        self.lr = lr
        self.w = None
        self.b = None

    def sigmoid (self, x):
        return 1/(1+np.exp(-x))

    def binary_cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-15  # Pequeño valor para evitar problemas con el logaritmo de cero
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)  # Clip para evitar valores extremos en el logaritmo ]0,1[
        loss = - (1 / len(y_true)) * np.sum(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
        return loss

    def fit(self, X, y):
        n_samples, n_features = X.shape
        #w1*x1 + w2*x2 + ... + wn*xn + b
        self.w = np.random.rand((n_features))
        self.b = 0

        for epoch in range(self.epochs):
            linear_prediction = np.dot(X, self.w) + self.b
            #(800,)
            prob = self.sigmoid(linear_prediction)

            #(samples, features)
            #dw = (features, samples) . (samples) = (features)
            dw = 1/n_samples * np.dot(X.T, (prob - y))
            db = 1/n_samples * np.sum(prob - y)

            #Update parameters
            self.w = self.w - self.lr*dw
            self.b = self.b - self.lr*db
            error = self.binary_cross_entropy_loss(y, prob)
            if epoch % 200 == 0:
                print("Epoch {0} W {1}".format(epoch, str(self.w)))
                print("Epoch {0} B {1}".format(epoch, str(self.b)))
                print("Error {0} ".format(str(error)))
                print("******************************")

    def predict(self, X):
        linear_prediction = np.dot(X, self.w) + self.b
        prob = self.sigmoid(linear_prediction)
        class_pred = [0 if y<0.5 else 1 for y in prob]
        return class_pred

logisticFromScratch = LogisticRegressionAI(lr=0.001, epochs=6000)
logisticFromScratch.fit(X_train, y_train)
y_hat = logisticFromScratch.predict(X_test)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_hat)
informe_clasificacion = classification_report(y_test, y_hat)

# Imprimir resultados
print(f'Accuracy del modelo: {accuracy:.2f}')
print('\nInforme de clasificación:')
print(informe_clasificacion)
```

Figura 5. Código Regresión Logística

Finalmente, el modelo se implementa instanciando la clase y entrenándola con X train y y train. Luego, se evalúa con X test y y test, calculando la accuracy y generando un clas sification report para medir su desempeño, el cuál muestra métricas como el nivel de accuracy, precision, recall, f1-score y support.



```python
# Inicializar el modelo de regresión logística
modelo_logistico = LogisticRegression(random_state=42)

# Ajustar el modelo al conjunto de entrenamiento
modelo_logistico.fit(X_train, y_train)

# Predecir en el conjunto de prueba
y_pred = modelo_logistico.predict(X_test)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
informe_clasificacion = classification_report(y_test, y_pred)

# Imprimir resultados
print(f'Accuracy del modelo: {accuracy:.2f}')
print('\nInforme de clasificación:')
print(informe_clasificacion)
```

Figura 6. Enter Caption

De igual forma, en lugar de implementar la regresión logística manualmente, se puede utilizar el método que facilita `sklearn`, tal y como se muestra en la figura 6. En este caso, se instancia el modelo de regresión logística con

## IV. Conclusión

A lo largo de este documento se profundizó en los fundamentos de la regresión logística, en particular en el uso de la verosimilitud como función de costo y en la aplicación del logaritmo para simplificar su derivación. Se revisaron ejemplos prácticos que ilustran cómo interpretar probabilidades según los valores de entrada, y se abordó el proceso de actualización de parámetros mediante gradiente descendente. Además, el repaso permitió conectar la teoría con la implementación práctica en Python, reforzando la comprensión del modelo y su utilidad en la clasificación de datos. Con esto, se sientan las bases para continuar con técnicas más avanzadas de aprendizaje supervisado.