Apuntes Semana 6 - 27/03/2025

Elaborado por: Joselyn Montero Rodríguez - c.2022136356

***Abstract*—Este documento recopila información sobre verosimilitud en regresión logística, la función de costo, un notebook de regresión logística y métricas clave para evaluar modelos, además de incluir noticias recientes sobre inteligencia artificial.**

I. Noticias de la semana

A. Robots

Se destaca un video donde se muestra el robot 'Newton', desarrollado por Nvidia y Disney, el cual está basado en Star Wars. Enlace al video.

Adicionalmente se mencionó que actualmente los robots han mejorado en la simulación del movimiento al caminar.

B. OpenAI

OpenAI ha lanzado la generación de imágenes con GPT-4o, el cual es un modelo multimodal que permite crear imágenes a partir de instrucciones textuales (prompts). Este modelo interpreta con precisión las indicaciones y mantiene el contexto de la conversación, facilitando la generación de imágenes detalladas y coherentes. Enlace a noticia.

Se habló sobre por qué ChatGPT no puede generar una imagen de una copa de vino llena, esto se debe a los sesgos derivados de su entrenamiento y es solo uno de muchos ejemplos. Lo mismo ocurre con diversos aspectos sociales, debido a los filtros aplicados a los datos.

II. Retroalimentación tarea I

La transformación de una variable categórica a una representación vectorial es crucial si el feature es relevante, especialmente en modelos como la regresión lineal. En el caso de la regresión logística, se pueden realizar pruebas como el **chi-cuadrado** para evaluar la independencia entre variables y su impacto en el modelo.

Para la elección de parámetros la recomendación es probar, ya que esto varía según el modelo que se esté usando, la cantidad de datos, etc.

III. Repaso clase del martes

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 maximize likelihood radica en su aplicación: para la predicción de valores continuos, se utiliza MSE, mientras que para modelar probabilidades, se utiliza maximize likelihood. Acá nuestra función de costo es:

$$\mathbf{L} = \prod f_{w,b}(x_i)^{y_i}(1 - f_{w,b}(x_i))^{(1-y_i)}, i = 1...N$$

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

1) Caso $\mathbf{y_i = 1}$*:*

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

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

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

2) Caso $\mathbf{y_i = 0}$*:*

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

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

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

- $wx + b = -1.32$
- $f_{w,b}(x_i) = \sigma(wx_i + b) = \sigma(-1.32) = 0.21$
- $= f_{w,b}(x_i)^0(1 - f_{w,b}(x_i))^{1-0}$
- $= (1 - f_{w,b}(x_i))^1 = (1 - \sigma(-1.32))$
- $= (1 - 0.21) = 0.79$
- 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.

B. Derivada función de costo

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

C. Logaritmos

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

D. Aplicación de logaritmo a la verosimilitud

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

Esto lo vamos a llamar **log-likelihood**. Es mucho más fácil de computar y derviar, 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:
$L = \frac{1}{N} \sum y_i \cdot In(f_{w,b}(x_i)) + (1 - y_i) \cdot In(1 - f_{w,b}(x_i))$
$L = -\frac{1}{N} [\sum y_i \cdot In(f_{w,b}(x_i)) + (1 - y_i) \cdot In(1 - f_{w,b}(x_i))]$
Ahora puedo minimizar la función, lo que permite aplicar el descenso del gradiente que se ha estado trabajando.

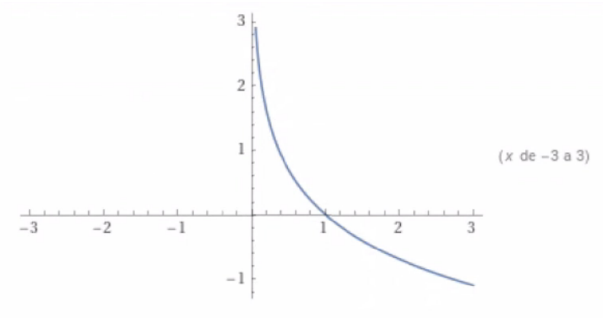

Fig. 1. Gráfica minimizando L

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

E. Actualización de parámetros

Es necesario actualizar los parámetros w y b, ya que son los que permitirán 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: $\frac{\partial L}{\partial b}$

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:
  $L = y_i \cdot In(f_{w,b}(x_i)) + (1 - y_i) \cdot In(1 - f_{w,b}(x_i))$
- 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:
  $L = y_i \cdot In(a(z(x))) + (1 - y_i) \cdot In(1 - a(z(x)))$

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

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

1) Cálculo de derivadas parciales: Importante recordar que el L que se está usando es
$L = -[y_i \cdot In(a(z(x))) + (1 - y_i) \cdot In(1 - a(z(x)))]$
Primero se inicia calculando la derivada parcial de L con respecto a la función sigmoide:

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

$$= -\left[\left(\frac{y_i}{a(x)} \cdot 1\right) + \left(\frac{(1 - y_i)}{1 - a(x)} \cdot -1\right)\right] \quad (2)$$

$$= -\left[\left(\frac{y_i}{a(x)}\right) - \left(\frac{(1 - y_i)}{1 - a(x)}\right)\right] \quad (3)$$

$$= \frac{-y_i}{a(x)} + \frac{(1 - y_i)}{1 - a(x)} \quad (4)$$

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 es 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:

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

Fig. 2. Derivada parcial de L respecto a z

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

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

Se procede a actualizar parámetros:

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

Donde a es un hiperparámetro (learning rate).

IV. 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('Caracteristica 1')
plt.ylabel('Caracteristica 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)
```

Fig. 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=25.

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)
```

Fig. 5. Código clase de 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 classification_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)
```

Fig. 6. Código regresión logística con sklearn

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

`LogisticRegression`, asegurando reproducibilidad con random_state=42.

Luego, el modelo se ajusta al conjunto de entrenamiento con `fit(X_train, y_train)`, permitiendo que los parámetros sean optimizados automáticamente. Posteriormente, se realiza la predicción en el conjunto de prueba utilizando `predict(X_test)`.

Para evaluar el rendimiento, se calcula la accuracy con `accuracy_score(y_test, y_pred)` y como se hizo anteriormente se genera un reporte con classification_report.

V. Métricas

Las métricas son medidas que sirven para indicar el rendimiento del modelo, se va a utilizar para elegir modelos. Estas métricas pueden ser métricas de rendimiento, de horas de entrenamiento, o de lo que se quiera para evaluar a los modelos. Esta es la forma más objetiva de evaluar y comparar un modelo.

A. Matriz de confusión

En un caso de predicciones de clases binarias se tienen 4 posibles resultados:

Target class / Predicted class	P	N
P	TP	FP (Type I)
N	FN (Type II)	TN

Fig. 7. Matriz de confusión

1) **True Positive (TP)**: La etiqueta real es positiva y el modelo también la predice como positiva, es decir, la predicción es correcta.
2) **True Negative (TN)**: La etiqueta real es negativa y el modelo la clasifica correctamente como negativa.
3) **False Positive (FP)**: La etiqueta real es negativa, pero el modelo la clasifica incorrectamente como positiva.
4) **False Negative (FN)**: La etiqueta real es positiva, pero el modelo la clasifica incorrectamente como negativa.

Estos últimos dos son tipos de error en estadística, el **error tipo I** es cuando ocurre un falso positivo (FP) y el **error tipo II** es cuando ocurre un falso negativo (FN).

En la figura 8 se muestra un caso cotidiano para entender mejor los resultados.

A continuación, se presentan algunas métricas que serán trabajadas a lo largo del curso. Es importante destacar que el profesor espera que sean incluidas en el proyecto, ya que serán clave para el análisis y la toma de decisiones.

Fig. 8. Ejemplo de caso cotidiano

B. Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Es la correcta clasificación de todos los samples. Mide la capacidad del modelo para predecir correctamente en general. Es útil cuando los errores en cada clase tienen la misma importancia, aunque en algunos casos puede no ser suficiente para evaluar el desempeño del modelo.

C. Precision

$$Precision = \frac{TP}{TP + FP}$$

Mide los errores tipo 1, es decir, todas las predicciones positivas correctas entre todas las predicciones positivas hechas.

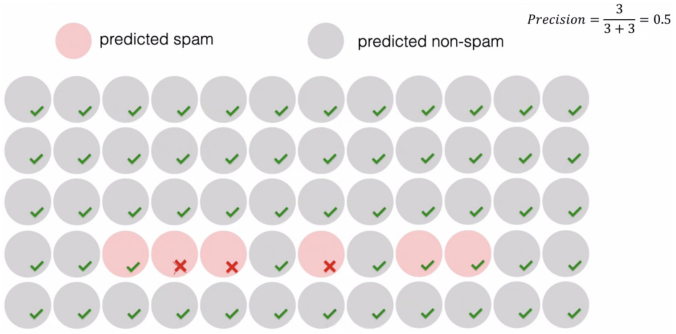

Fig. 9. Ejemplo precision

En el ejemplo se muestra un modelo que clasifica los correos como spam o no spam, los círculos que están en rojo son los clasificados como spam, haciendo el cálculo se puede ver que el modelo obtuvo un precision de 0.5, es decir, de 6 que clasificó como spam sólo la mitad era spam realmente.

D. Recall (Sensivity)

$$Recall = \frac{TP}{TP + FN}$$

Mide los errores tipo II, es decir, todas las predicciones positivas correctas entre todos los ejemplos positivos del dataset.

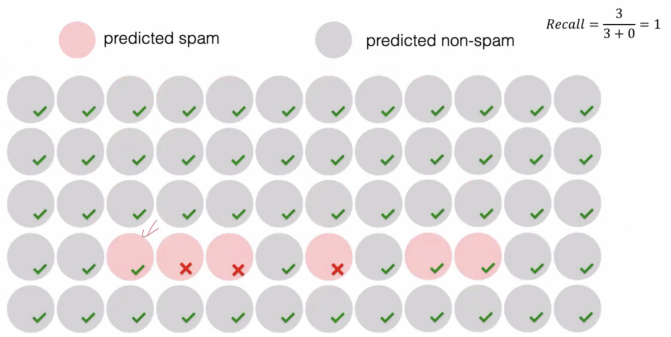

Fig. 10. Ejemplo recall

En el ejemplo se ve el mismo modelo que el ejemplo anterior que clasifica correos como spam o no spam, en este caso no hubo ningún correo spam que fuera clasificado incorrectamente como no spam, lo que da como resultado un recall de 1, lo que indica que está perfecto en este aspecto.

E. F1-Score

$$F1 = \frac{1 \cdot precision \cdot recall}{precision + recall}$$

Es usada para casos de clasificación donde hay desequilibrio de clases. Indica si el modelo logra un buen balance entre el precision y el recall.

F. Caso de estudio

- Total de pacientes: 1000
- Pacientes con cáncer (clase positiva): 30
- Pacientes sin cáncer (clase negativa): 970

Se realiza un modelo de clasificación y se obtienen los siguientes resultados:

- TP: 25 pacientes con cáncer correctamente identificados.
- FN: 5 pacientes con cáncer que el modelo clasificó incorrectamente como no cancerosos.
- TN: 950 pacientes sin cáncer correctamente identificados.
- FP: 20 pacientes sin cáncer que el modelo clasificó incorrectamente como con cáncer.

Predicción/Objetivo	Cáncer	No cáncer
Cáncer	25	20
No cáncer	5	950

Fig. 11. Matriz de confusión del caso de estudio

1) Acurracy:

$$Accuracy = \frac{TP + TN}{TOTALPACIENTES} = \frac{25 + 950}{1000} = 0.975$$

Si se usa sólo el accuracy, se tiene un resultado de 97.5%, por lo que parece un buen modelo.

2) Recall:

$$Recall = \frac{TP}{TP + FN} = \frac{25}{25 + 5} = 0.833$$

Cuando se calcula el recall, da un resultado del 83.3%, por lo que indica que hay espacio de mejora, ya que es porcentaje bajo para la seriedad del problema porque no está diagnosticando bien a las personas que tienen cáncer.

3) Precision:

$$Precision = \frac{TP}{TP + FP} = \frac{25}{25 + 10} = 0.55 = 55\%$$

Precision da un resultado de 55%, es demasiado baja, por lo que sigue indicando que hay un espacio de mejora en el modelo, acá se dan falsos positivos pero al final si se analiza la verdad no está poniendo en riesgo a la persona con los falsos positivos que se hicieron.

4) F1-Score:

$$F1 = \frac{1 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot 0.55 \cdot 0.833}{0.55 + 0.833} = 0.662$$

62.2% es resultado es bajo, este es muy afectado por el cálculo del precision. Por lo que al final se concluye que la capacidad del modelo de clasificar correctamente las clases minotarias no es bueno.

G. ROC y AUC

El área bajo la curva (AUC) representa la relación entre los falsos positivos y los falsos negativos a medida que se ajustan los pesos del modelo. Hay herramientas de sklearn que lo calculan pero acá lo importante es saber entenderlo.

Un valor de AUC cercano a 0.5 indica que el modelo no tiene capacidad de clasificación y su desempeño es similar al de lanzar una moneda, lo que sugiere que debe mejorarse. Si el AUC está en 0.7, significa que el modelo tiene un desempeño aceptable. Un rango de 0.7 a 0.8 se considera decente y tolerable.

Un **AUC de 1 representa un clasificador perfecto**, que es el objetivo ideal. Por otro lado, si el AUC es menor a 0.5, significa que el modelo está clasificando peor de lo esperado y necesita ajustes significativos.

VI. Comentarios finales

Para el quiz se recomienda apoyarse en las lecturas que se encuentran en el TecDigital. Enlace a lecturas.