Práctica 2
Regresión logística

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

Jesus Abraham Haros Madrid
A01252642@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

ABSTRACT

1 INTRODUCCIÓN

Actualmente, los modelos de regresión se han convertido en uno de los componentes más útiles del análisis de datos y el aprendizaje automático, en donde sea necesario describir la relación entre una variable resultante y una o más variables independientes. En algunos casos, la variable de resultado es discreta y adopta dos o más valores, es por eso que el modelo de regresión logística se ha convertido en muchos campos como el método estándar de análisis en esta situación.[1]

En esta práctica demostraremos dos implementaciones de este modelo, la implementación de la librería de Python *sci-kit learn* y nuestra implementación usando el método de gradiente descendente.

2 CONCEPTOS PREVIOS

- Programación básica en los lenguajes R y Python
- Conocimiento de las librerías scikit-learn, pandas y numpy
- Conocimientos de estadística y de regresión logística

3 METODOLOGÍA

3.1 Datasets

Para comparar ambas implementaciones utilizamos dos datasets, expuestos a continuación:

3.1.1 *Dataset DEFAULT.*

Este dataset está compuesto por 10,000 filas, cada una representa un cliente de un banco que puede o no cumplir con los pagos de su tarjeta de crédito (columna "default"). De cada cliente tenemos la siguiente información:

- Columna "default": Tiene los valores "Yes"/"No", representa si la persona realizó el pago mínimo a su tarjeta de crédito.
- Columna "student": Valores "Yes"/"No", representa si el cliente es un estudiante en ese momento.
- Columna "balance": Número decimal positivo que representa el balance de la tarjeta de crédito del cliente. Promedio de 835.4, números en el rango [0, 2654.3].
- Columna "income": Número decimal positivo que representa los ingresos que tiene el cliente. Promedio de 33517, números en el rango [772, 73554]

De este dataset, nuestro objetivo es predecir la columna "default" a partir de los otros tres parámetros, y como preparación cambiamos los valores de "student" ("Yes"/"No") a valores 1 y 0 respectivamente.

3.1.2 *Dataset GENERO.*

Este dataset representa las mediciones de peso y altura de 10,000 personas, en conjunto con el género de la persona a la que se realizaron las medidas. Las columnas son:

- "Gender": Valores "Male"/"Female", el género de la persona a la que le corresponde esta fila de mediciones.
- "Height": La altura de la persona en pulgadas, promedio de 66.37, en el rango [54.26 y 79.00].
- "Weight": El peso de la persona en libras, promedio de 161.4, en el rango [64.7, 270.0].

La columna objetivo seleccionada de este dataset fue el género ya que tiene dos clasificaciones.

3.2 Regresión logística con *sklearn*

Para la primera parte de la práctica, en la que utilizamos la implementación de *sci-kit learn*, seleccionamos la clase *sklearn.linear_model.LogisticRegression*[2] para nuestros scripts.

3.2.1 *Dataset DEFAULT.*

Para este dataset primero leemos el archivo CSV a un *Dataframe* de *pandas*, transformamos las columnas "default" y "student" para que contengan valores booleanos y enteros respectivamente, seleccionamos las columnas que nos servirán como variables independientes (columnas "student", "balance" e "income") y variable dependiente (columna "default"). Después partimos las filas del dataset en una porción del 80% que usaremos para entrenar el modelo, y otra porción del 20% para probarlo.

Instanciamos el modelo de regresión de *sklearn*, lo entrenamos con los datos y después predecimos la variable dependiente con el modelo para así compararlo con los datos reales de prueba, usando una medida de tasa de precisión y la matriz de confusión.

El código de este ejemplo puede se encuentra en el apéndice A.

3.2.2 *Dataset GENERO.*

Para este dataset realizamos un procedimiento similar: leer el dataset para crear un *Dataframe*, seleccionar las columnas de variables independientes ("Height" y "Weight") y la dependiente ("Gender"), dividir el dataset en 80%/20%, entrenar el modelo, y predecir la variable dependiente para los datos de prueba, para así comparar estos con los datos reales.

El código para esta sección se encuentra en el apéndice B.

3.3 Regresión logística con Gradiente Descendente

En esta sección de la práctica implementamos nuestra propia clase de gradiente descendente, e intentamos que los métodos de nuestro

modelo fueran muy parecidos a los de la clase de *sklearn* para que así fuera fácil intercambiar las implementaciones. A pesar de esto, implementamos algunas diferencias para que el código funcionara correctamente.

Nuestra implementación del modelo de regresión logística se encuentra en el apéndice C.

Para implementar el gradiente descendente en esta práctica se tuvo que hacer algunas modificaciones al algoritmo descrito en nuestra práctica 1. Se tuvo que cambiar el algoritmo internamente al momento de hacer el cálculo de θ, agregando el cálculo de μ como se describe en los requisitos de la práctica, con esto podemos calcular correctamente la θ de nuestro modelo, y al momento de usar la predicción se pasa por la función sigmoide para que posicionara todos los resultados predichos en el rango de 0 a 1 y así poder clasificar los datos en grupos, es por ello que se hizo una función para redondear el resultado de la función sigmoide. La predicción de nuestro gradiente descendente regresa una lista de números 0 y 1, por ello se necesitan reconvertir estos valores a los originales del dataset, a continuación se explica lo que se hizo en cada uno para llegar al resultado deseado.

3.3.1 Dataset DEFAULT.

Muy parecido a lo que ya se ha estado haciendo con *scikit learn*, para poder trabajar con este dataset tendremos que leerlo para crear un *Dataframe*, transformando los valores de la columna "default" y "student" a valores enteros y así poder entrenar nuestro modelo. Después de esto partimos el dataset para 80% de entrenamiento y 20% de pruebas.

Instanciamos nuestro Gradiente Descendente, lo entrenamos con los datos y después predecimos la variable dependiente con el modelo, la diferencia y el paso extra que tuvimos que hacer fue reconvertir la variable "default" de número entero a valor booleano para poder comparar con el dataset original (esto pudo haberse hecho también convirtiendo el dataset original a números enteros, pero se decidió hacer de esta manera para preservar los valores del dataset).

El código fuente de este ejemplo se encuentra en el apéndice D.

3.3.2 Dataset GENERO.

Al igual que con el dataset default, para poder predecir la variable dependiente en este dataset se utilizó lo mismo que en su implementación para *scikit learn* pero agregando una conversión de valores para la variable "genero", convirtiéndola de los strings "Male" y "Female" a los valores enteros 0 y 1.

Una vez entrenado el modelo se revierte la conversión de valores para poder preservar los valores del dataset.

El código fuente puede ser encontrado en el apéndice E.

4 RESULTADOS

4.1 Regresión logística con *sklearn*

4.1.1 Dataset DEFAULT.

El modelo de regresión logística de *sklearn* tuvo una tasa de precisión alrededor de 0.965, con una gráfica de la matriz de confusión así:

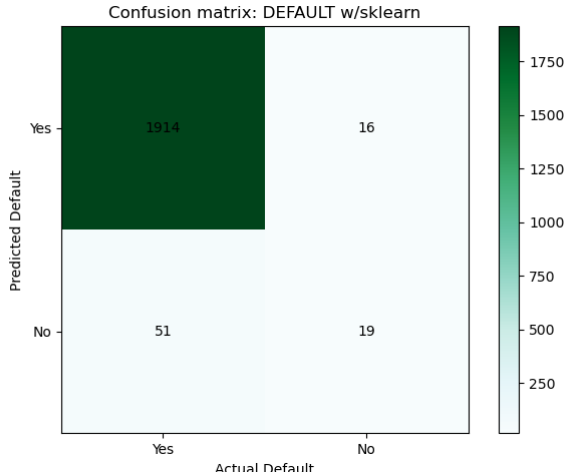

Figure 1: Matriz de confusión del dataset DEFAULT con la implementación de sklearn

Se puede observar que la gran mayoría de los resultados son positivos, por lo que el modelo usualmente acierta al clasificar la casi la totalidad de las filas como positivas.

4.1.2 Dataset GENERO.

En este ejemplo, el modelo de *sklearn* tuvo una tasa de precisión cercana a 0.925, y una matriz de confusión de esta forma:

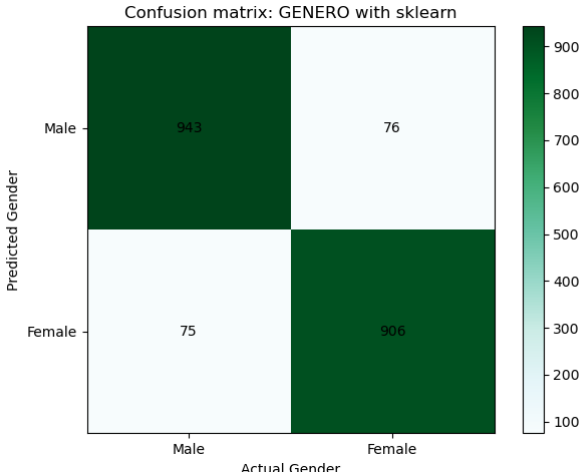

Figure 2: Matriz de confusión del dataset GENERO con la implementación de sklearn

También se creó un diagrama de dispersión utilizando como eje X la altura de la persona, y en el eje Y el peso. Los puntos rojos representan personas del género femenino y los azules personas del género masculino.

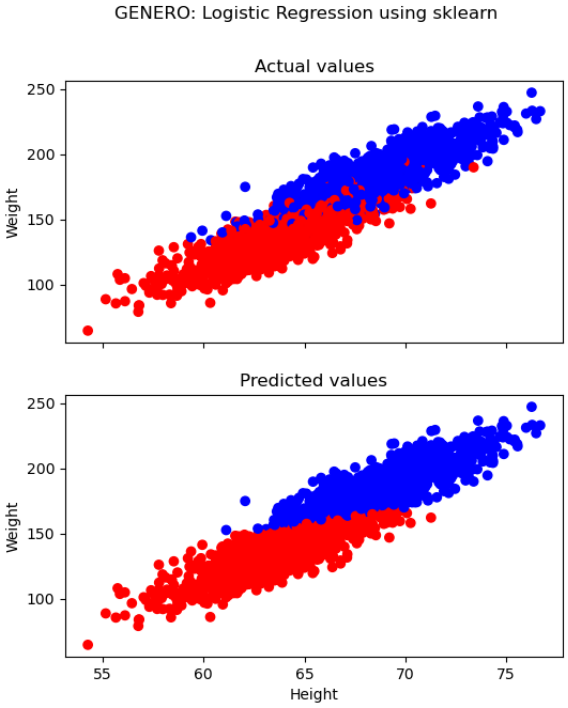

Figure 3: Diagrama de dispersión del dataset GENERO con la implementación de sklearn

La gráfica superior representa el conjunto de valores de prueba, mientras que la gráfica inferior muestra el género predecido por el modelo de regresión a partir de los valores de prueba.

Para la generación de estas gráficas se utilizó el código incluído en el apéndice F.

4.2 Regresión logística con Gradiente Descendente

4.2.1 Dataset DEFAULT.

El modelo de regresión logística con *gradiente descendente* tuvo resultados muy parecidos a los resultados del modelo de *scikit learn*, ya que la mayoría de los resultados son positivos, lo cual es un acierto debido a que en el dataset original la mayoría de los datos se encuentran como positivos.

Este modelo tuvo una tasa de precisión alrededor de 0.965, con matriz de confusión así:

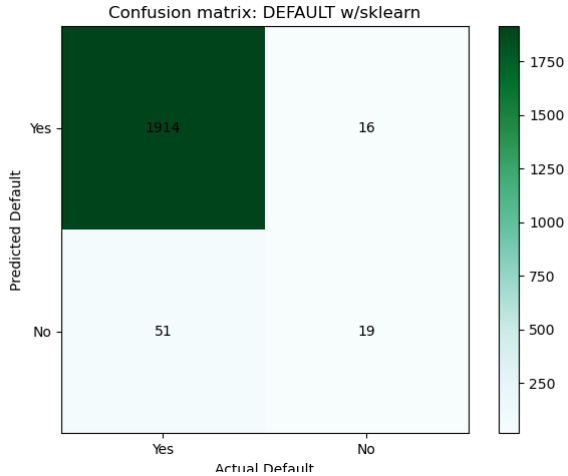

Figure 4: Matriz de confusión del dataset DEFAULT con la implementación del gradiente descendente

4.2.2 Dataset GENERO.

El modelo de regresión logística con *gradiente descendente* tuvo una tasa de precisión de 0.8965, con una gráfica de la matriz de confusión así:

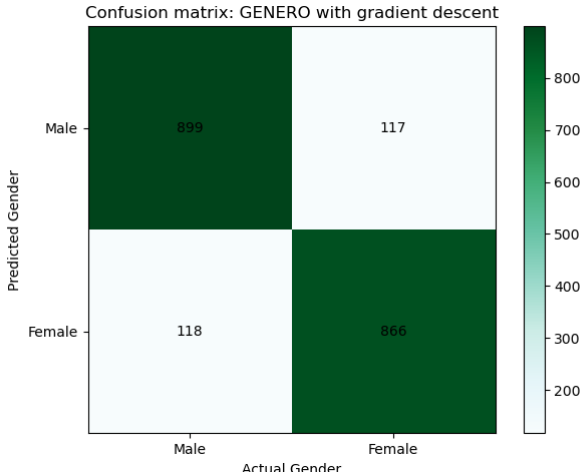

Figure 5: Matriz de confusión del dataset GENERO con la implementación del gradiente descendente

En este caso podemos ver cómo lo predicho por el modelo entrenado con *scikit learn* es un poco más preciso (aunque no muy notorio), aún asi su precisión es muy buena utilizando el gradiente descendente.

Para esto también se creó un diagrama de dispersión para poder comparar el dataset original contra el predicho por el modelo del gradiente descendente, en el eje X utilizamos la altura y en el eje Y el peso, los puntos rojos representan las personas de género femenino y de azul las del género masculino.

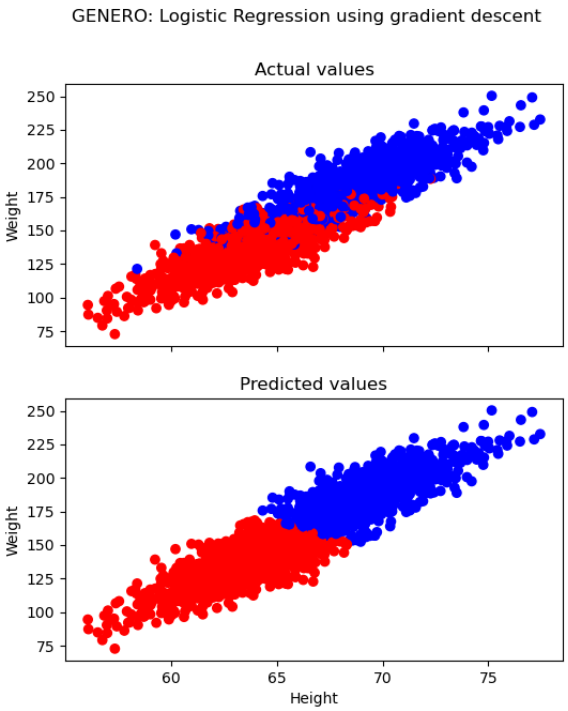

Figure 6: Diagrama de dispersión del dataset GENERO con la implementación del gradiente descendente

5 CONCLUSIONES Y REFLEXIONES

Existen múltiples formas de clasificar los datos de un dataset para poder hacer predicciones de variables y poder clasificarlas en grupos definidos, para esta práctica se utilizó la regresión logística que nos ayudó a clasificar los datos de ambos datasets; para el caso del dataset GENERO, la regresión nos ayudo a predecir el sexo dado altura y peso, mientras que para el dataset DEFAULT nos ayudó a predecir la columna ”default” dadas las columnas ”student”, ”balance” e ”income”. Al igual que en la práctica anterior se predijeron los datos utilizando la librería *scikit learn* y el gradiente descendiente,

Se observaron resultados muy similares entre *sklearn* y el gradiente descendiente, en ambos hubo un problema al momento de predecir el dataset DEFAULT y es que, como consecuencia de que muchos valores en la columna ”default” son “Yes”, ambos modelos predecían “Yes” para muchos casos, si se observan las matrices de confusión de ambos modelos podemos apreciar como casi nunca predicen “No”.

5.1 Refrexión de Abraham

La práctica pasada se me hizo muy complicado entender el funcionamiento interno del gradiente descendiente, sabía qué hacía pero no cómo funcionaba, en esta práctica, al tener que modificar el código interno de este, pude comprender mejor que está haciendo. También se me hizo muy interesante la regresión logística y comprender la función sigmoide para poder clasificar y acotar los datos para dos categorías, me queda la duda sobre como clasificar para varias categorías pero supongo que ser verá en futuras prácticas.

Esta práctica fue más sencilla de hacer por el hecho de que fue parecida a la anterior, fue más un refuerzo de la clase en vez de mucho investigación como la práctica anterior. Cada vez aprendo más sobre las funciones de numpy y python para trabajar con datos y matrices, aunque aún me falta mucho por aprender, estoy satisfecho de poder hacer prácticas como estas que confirman y refuerzan lo aprendido en clase.

5.2 Reflexión de Mario

Considero que esta práctica resolvió muchas de las dudas que tenía sobre la regresión logística y me ayudó a comprender en qué situaciones son útiles este tipo de algoritmos de clasificación, además de que, con el primer dataset, pude observar un ejemplo de un caso en el que probablemente la regresión logística no es la mejor solución a este problema ya que el dataset está demasiado sesgado hacia los clientes que no pagaron su tarjeta contra los que si.

Además, mejoré mis habilidades en Python utilizando librerías como sklearn, numpy y pandas, herramientas usadas frecuentemente en este tipo de análisis de datos por su facilidad de manipular grandes cantidades de información y abstraer modelos complejos, para ser expresados en pocas líneas de código.

REFERENCES

[1] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression*. Vol. 398. John Wiley & Sons.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

A CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET DEFAULT

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from graphs import graphConfusionMatrix

print(" ~ Reading default.txt and generating train and test sets")
data = pd.read_csv('default.txt', sep=" ")
# Transform 'default' column from Yes/No to a boolean
data["default"] = (data["default"] == "Yes").astype(bool)
# Transform 'student' column from Yes/No to an integer
data["student"] = (data["student"] == "Yes").astype(int)
x = data.iloc[:, 1:4].values.reshape(-1, 3)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

print(" ~ Creating sklearn's logistic regression model")
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())

print(" ~ Testing sklearn's logistic regression model")
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

graphConfusionMatrix(cm, ["Yes", "No"], "Default",
                     "DEFAULT w/ sklearn", "default_cm_sklearn.png")
```

B CÓDIGO DE REGRESIÓN LOGÍSTICA CON SKLEARN DEL DATASET GENERO

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from graphs import graphConfusionMatrix, graphGENERO

print(" ~ Reading genero.txt and generating train and test sets")
data = pd.read_csv('genero.txt')
x = data.iloc[:, 1:3].values.reshape(-1, 2)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

print(" ~ Creating sklearn's logistic regression model")
regressor = LogisticRegression()
regressor.fit(xTrain, yTrain.ravel())

print(" ~ Testing sklearn's logistic regression model")
yPredicted = regressor.predict(xTest)
accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

graphGENERO(xTest, yTest, yPredicted, "Logistic Regression using sklearn",
            "genero_sklearn.png")
graphConfusionMatrix(cm, ["Male", "Female"], "Gender",
                     "GENERO with sklearn", "genero_cm_sklearn.png")
```

C NUESTRA IMPLEMENTACIÓN DE GRADIENTE DESCENDENTE

```python
from typing import Tuple
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error


def normalize(x: pd.DataFrame) -> Tuple[np.ndarray, float, float]:
    mu = np.mean(x, axis=0)
    sigma = np.std(x, axis=0, ddof=1)
    x_norm = (x - mu) / sigma
    return x_norm, mu, sigma


class GradientDescent:
    def __init__(self, learning_rate: float = 0.1, max_iterations: int = 200,
                 precision: float = 0.00001):
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.precision = precision
        self.theta = None
        self.mu = None
        self.sigma = None

    def fit(self, x: pd.DataFrame, y: pd.DataFrame):
        x, self.mu, self.sigma = normalize(x)
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        self.theta = np.zeros(x.shape[1])
        prev_cost = -1
        for _ in range(self.max_iterations):
            predictions = x.dot(self.theta)
            muPredictions = 1 / (1 + np.exp(-1 * predictions))
            errors = np.subtract(muPredictions, y)
            sum_delta = (self.learning_rate / x.shape[0]) * x.T.dot(errors)
            self.theta -= sum_delta

            cost = mean_squared_error(y, predictions)
            if abs(cost - prev_cost) < self.precision:
                break
            prev_cost = cost

    def predict(self, x: pd.DataFrame):
        if self.theta is None or self.mu is None or self.sigma is None:
            raise Exception(
                "GradientDescent::predict() called before model was trained")

        x = (x - self.mu) / self.sigma
        x = np.hstack((x, np.ones((x.shape[0], 1))))
        predictions = x.dot(self.theta)
        predictionSigmoide = 1 / (1 + np.exp(-1 * predictions))
        result = np.array(list(map(self.roundSigmoide, predictionSigmoide)))
        return result

    # Function to round to 0 or 1 the result of the sigmoide function
    def roundSigmoide(self, t):
        if t <= 0.5: return 0
        return 1
```

D CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET DEFAULT

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from GradientDescent import GradientDescent
from graphs import graphConfusionMatrix, graphGENERO

print(" ~ Reading default.txt and generating train and test sets")
data = pd.read_csv('default.txt', sep=" ")
# Transform 'default' column from Yes/No to a boolean
data["default"] = (data["default"] == "Yes").astype(bool)
# Transform 'student' column from Yes/No to an integer
data["student"] = (data["student"] == "Yes").astype(int)
x = data.iloc[:, 1:4].values.reshape(-1, 3)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

# Convert Bool true/false to Int 1/0
yTrainToInt = yTrain.ravel().astype(int)  # True = 1, False = 0
yTest = yTest.ravel()

print(" ~ Creating our logistric regression model with gradient descent")
regressor = GradientDescent()
regressor.fit(xTrain, yTrainToInt)

print(" ~ Testing our logistric regression model with gradient descent")
yPredicted = regressor.predict(xTest)

# Convert Int 1/0 to True / False
yPredicted = np.array(list(map((lambda x: x == 1), yPredicted)))

accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

graphConfusionMatrix(cm, ["Yes", "No"], "Default",
                     "DEFAULT w/ gradient descent", "default_cm_gradient.png")
```

E CÓDIGO DE REGRESIÓN LOGÍSTICA CON GRADIENTE DESCENDENTE DEL DATASET GENERO

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

from GradientDescent import GradientDescent
from graphs import graphConfusionMatrix, graphGENERO

print(" ~ Reading genero.txt and generating train and test sets")
data = pd.read_csv('genero.txt')
x = data.iloc[:, 1:3].values.reshape(-1, 2)
y = data.iloc[:, 0].values.reshape(-1, 1)
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.2)

# Convert String Male/Female to Int 0/1
yTrainToInt = (yTrain.ravel() == "Male").astype(int)  # Male = 1, Female = 0
yTest = yTest.ravel()

print(" ~ Creating our logistric regression model with gradient descent")
regressor = GradientDescent()
regressor.fit(xTrain, yTrainToInt)

print(" ~ Testing our logistric regression model with gradient descent")
yPredicted = regressor.predict(xTest)
# Convert Int 0/1 to Female/Male
yPredicted = np.array(
    list(map((lambda x: ["Female", "Male"][x]), yPredicted)))
accuracy = accuracy_score(yTest, yPredicted)
print(" →   Accuracy:", accuracy)
cm = confusion_matrix(yTest, yPredicted)
print(" →   Confusion matrix:", cm.tolist())

graphGENERO(xTest, yTest, yPredicted, "Logistic Regression using gradient descent",
            "genero_gradient.png")
graphConfusionMatrix(cm, ["Male", "Female"], "Gender",
                     "GENERO with gradient descent", "genero_cm_gradient.png")
```

F CÓDIGO DE GENERACIÓN DE GRÁFICAS

```
import sys
import matplotlib.pyplot as plt
import numpy as np

shouldDisplay = "--display-graphs" in sys.argv
shouldSave = "--save-graphs" in sys.argv

mapColor = np.vectorize(lambda x: "b" if x == "Male" else "r")


def graphGENERO(xys, actualVals, predictedVals, title, filename):
    fig, (ax1, ax2) = plt.subplots(2, sharex=True)
    fig.set_size_inches(6, 7)
    fig.suptitle("GENERO: " + title)

    ax1.set_title("Actual values")
    ax1.set_ylabel("Weight")
    actualColors = mapColor(actualVals).flatten()
    ax1.scatter(xys[:, 0], xys[:, 1], c=actualColors)

    ax2.set_title("Predicted values")
    ax2.set_xlabel("Height")
    ax2.set_ylabel("Weight")
    predictedColors = mapColor(predictedVals).flatten()
    ax2.scatter(xys[:, 0], xys[:, 1], c=predictedColors)

    if shouldSave:
        fig.savefig(filename)
    if shouldDisplay:
        plt.show()


def graphConfusionMatrix(cm, labels, variableName, title, filename):
    plt.figure()
    plt.imshow(cm, interpolation="nearest", cmap="BuGn")
    plt.title("Confusion matrix: " + title)
    plt.colorbar()
    plt.xticks(np.arange(2), labels, size=10)
    plt.yticks(np.arange(2), labels, size=10)
    plt.tight_layout()
    plt.xlabel("Actual " + variableName)
    plt.ylabel("Predicted " + variableName)

    for x in range(2):
        for y in range(2):
            plt.annotate(cm[x][y], xy=(y, x),
                         horizontalalignment="center",
                         verticalalignment="center")

    if shouldSave:
        plt.savefig(filename)
    if shouldDisplay:
        plt.show()
```