

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Inteligencia artificial avanzada para la ciencia de datos I

(Gpo 101)



**Tecnológico
de Monterrey**

Inteligencia artificial avanzada para la ciencia de datos I Concentración, (Gpo 101)

Jorge Adolfo Ramírez Uresti

**Momento de Retroalimentación: Módulo 2 Implementación de
una técnica de aprendizaje máquina sin el uso de un framework.
(Portafolio Implementación)**

Oswaldo Daniel Hernández de Luna | A01753911

01 de septiembre 2024

Introducción

Este proyecto tiene como objetivo desarrollar una implementación manual de uno de los algoritmos más fundamentales en el campo del machine learning, como en este caso el escogido fue, la regresión logística, el cual sin recurrir a bibliotecas especializadas o frameworks avanzados, este ejercicio no solo refuerza los conceptos teóricos aprendidos durante el módulo, sino que también brinda una comprensión profunda de cómo funcionan estos algoritmos en su nivel más básico debido a la inicialización de comprensión de este tema.

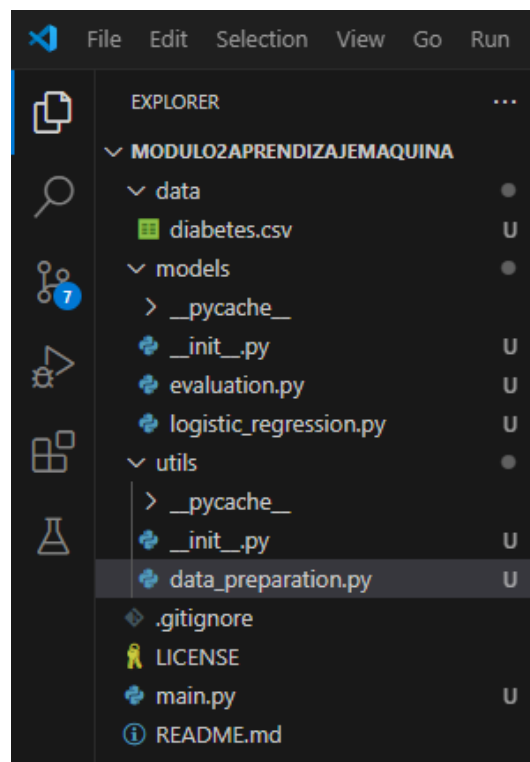
La regresión logística en base a lo aprendido en clase y material del mismo, es un modelo predictivo utilizado para clasificar datos binarios, es decir, donde la variable de salida puede tomar solo dos valores posibles, por lo que, este tipo de modelo es la base sobre la cual se construyen técnicas más complejas, como las redes neuronales, y se utiliza ampliamente en problemas que requieren la clasificación de eventos, como la detección de enfermedades, la predicción de fallas en sistemas, y el análisis de riesgo crediticio, etc.

El reto propuesto en este proyecto es implementar el algoritmo de regresión logística desde cero, abordando aspectos clave como la preparación de datos, la optimización de parámetros mediante el descenso de gradiente, y la evaluación del modelo utilizando métricas estándar como precisión, recall, y F1 score.

A lo largo de este documento, se detallará el proceso de implementación del modelo, desde la carga y preparación del dataset hasta la generación de predicciones, finalmente se presentarán los resultados de la evaluación del modelo, discutiendo su desempeño y las decisiones tomadas durante el desarrollo del modelo.

Desarrollo del proyecto

El proyecto desarrollado se estructura en torno a cuatro archivos .py interrelacionados, cada uno diseñado para manejar un aspecto específico del proceso de aprendizaje automático, siguiendo una lógica modular que permite una clara separación de responsabilidades y facilita la comprensión, el mantenimiento, y la extensión del código, por lo que la decisión de estructurar el proyecto de esta manera refleja no solo una preferencia personal por la organización del código, sino también una adherencia a las mejores prácticas de desarrollo de software, tal como se nos ha enseñado a lo largo de nuestra formación académica en programación y desarrollo de algoritmos. Quedando de la siguiente manera:



Además, esta estructura modular facilita la reutilización del código, por ejemplo, el módulo encargado de la preparación de datos el cual lo nombré “data_preparation.py” puede ser fácilmente adaptado y reutilizado en otros proyectos que requieran un preprocesamiento similar, teniendo esta capacidad de reutilización es clave en proyectos de mayor complejidad, donde los mismos procesos o cálculos pueden ser aplicables en diferentes partes del sistema o en proyectos futuros.

Estructura del código

Preparación de Datos: data_preparation.py

Como primer paso es la preparación de datos, y el archivo `data_preparation.py`, se maneja la carga del dataset desde un archivo CSV y se realiza un preprocesamiento detallado que incluye el manejo de valores faltantes y la normalización de las características, esto por el simple hecho de mantener una buena interpretación y organización de los datos, el dataset de diabetes utilizado, contenía valores faltantes en columnas críticas como Glucose, BloodPressure, y BMI que pueden afectar significativamente la precisión del modelo, por lo que, en este módulo, dichos valores son reemplazados con la mediana de cada columna, para mantener la integridad de los datos.

```
def prepare_data(filepath):  
    """  
    aquí la idea es preparar los datos para el entrenamiento del modelo, incluyendo manejo de valores  
    faltantes y normalización.  
  
    dentro de lo que se hace menciona una lista de lo que devuelve esta funcion:  
    - X: array con las características.  
    - y: array con las etiquetas (0 o 1).  
    - means: medias de las características.  
    - stds: desviaciones estándar de las características.  
    """  
    df = pd.read_csv(filepath)  
  
    # con eso lo que se hace es reemplazar valores de 0 con NaN y luego llena con la mediana de la columna  
    cols_to_replace = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']  
    for col in cols_to_replace:  
        df[col] = df[col].replace(0, np.nan)  
        df[col] = df[col].fillna(df[col].median())  
  
    X = df.drop('Outcome', axis=1).values # Características  
    y = df['Outcome'].values # Etiquetas  
  
    # Normalización de las características  
    means = np.mean(X, axis=0)  
    stds = np.std(X, axis=0)  
    X = (X - means) / stds  
  
    # Añadir un término de sesgo  
    X = np.c_[np.ones(X.shape[0]), X]  
  
    return X, y, means, stds
```

Además, la normalización de las características mediante la resta de la media y la división por la desviación estándar es fundamental, además como se menciona en el material del módulo, la normalización es crucial para algoritmos como la regresión logística, que se basan en el descenso de gradiente para optimizar los parámetros del modelo, y la normalización asegura que todas las características contribuyan de manera igual al proceso de optimización, evitando que aquellas con escalas más grandes dominen el proceso de ajuste de pesos.

Finalmente, se realiza la división del dataset en tres subconjuntos: entrenamiento, validación y prueba, esta división es crítica para evaluar el modelo de manera objetiva y evitar el overfitting, por lo que, el conjunto de validación permite ajustar parámetros sin comprometer la evaluación final, mientras que el conjunto de prueba proporciona una medida imparcial del desempeño del modelo.

```
def split_data(X, y, train_size=0.7, validation_size=0.15):  
    """  
    Se va a dividir los datos en conjuntos de entrenamiento, validación y prueba como se nos especifica  
    realizar.  
  
    Con esto lo esperado a devolver es:  
    - X_train, X_validation, X_test: características para entrenamiento, validacion y prueba.  
    - y_train, y_validation, y_test: etiquetas para entrenamiento, validacion y prueba.  
    """  
    m = len(y)  
    indices = np.arange(m)  
    np.random.shuffle(indices)  
  
    train_end = int(train_size * m)  
    validation_end = int(validation_size * m) + train_end  
  
    X_train = X[indices[:train_end]]  
    y_train = y[indices[:train_end]]  
  
    X_validation = X[indices[train_end:validation_end]]  
    y_validation = y[indices[train_end:validation_end]]  
  
    X_test = X[indices[validation_end:]]  
    y_test = y[indices[validation_end:]]  
  
    return X_train, X_validation, X_test, y_train, y_validation, y_test
```

Implementación del Algoritmo: logistic_regression.py

El archivo logistic_regression.py es la parte fundamental del proyecto, donde se implementa manualmente el algoritmo de regresión logística, las cuales son:.

Función Sigmoidal: Se implementa aquí la base matemática sobre la cual se construye la regresión logística, esta función mapea cualquier valor real a un rango entre 0 y 1, lo que permite interpretar la salida del modelo como una probabilidad, como se discutió en los materiales del módulo, la regresión logística no es simplemente una regresión lineal, sino que utiliza esta función sigmoidal para transformar la salida y hacer posible la clasificación binaria.

Función log loss: La función, mide la variación entre las predicciones del modelo y las etiquetas reales, además, se ayuda a prevenir el overfitting, siendo esta fundamental para mejorar la capacidad de generalización del modelo.

```
def sigmoid(z):  
    """  
    Esta funcion lo que hara es calcular la función sigmoial, que mapea cualquier valor real en un rango  
    entre 0 y 1.  
  
    la cual retorna:  
    - La probabilidad entre 0 y 1.  
    """  
    return 1 / (1 + np.exp(-z))  
  
def log_loss(y_true, y_pred, params, reg_strength):  
    """  
    la funcion de perdida logaritmica (log loss) con regularizacion L2 para evitar el overfitting.  
  
    Y aqui vuelve:  
    - El valor de la función de perdida logaritmica regularizada.  
    """  
    epsilon = 1e-15 # aqui lo ajusto debido a que tenia problemas y evitar log(0)  
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon) # Clipping de predicciones para evitar problemas numericos,  
    regularization = (reg_strength / 2) * np.sum(params[1:] ** 2) # Aqui finaliza la regulacion  
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred)) + regularization
```

Descenso de Gradiente: La optimización de los parámetros del modelo se realiza mediante un algoritmo iterativo que ajusta los pesos en la dirección que minimiza la función de pérdida, siendo este proceso esencial, ya que,, permite encontrar el conjunto de parámetros que mejor se ajusta a los datos de entrenamiento, minimizando así los errores de predicción.

```
def gradient_descent(X, y, params, learning_rate, epochs, reg_strength):  
    """  
    Optimiza los parametros del modelo utilizando descenso de gradiente con regularizacion L2.  
  
    El gradiente descendiente nos va a devolver:  
    - params: que es la variable array numpy con los parámetros optimizados.  
    """  
    m = len(y) # Numero de ejemplos a tomar  
    for epoch in range(epochs):  
        y_pred = predict(X, params) # esto lo que hace es hacer las predicciones actuales  
        error = y_pred - y # y aqui el error es calculado  
        gradient = (np.dot(X.T, error) / m) + (reg_strength * np.r_[0, params[1:]]) / m # Gradiente con regular  
        params -= learning_rate * gradient # finalmente se actualizan los parametros  
  
        if epoch % 100 == 0:  
            loss = log_loss(y, y_pred, params, reg_strength)  
            print(f"Epoch {epoch}: Loss = {loss:.4f}")  
  
    return params
```

Evaluación del Modelo: evaluation.py

Una vez entrenado el modelo, es importante evaluar su desempeño utilizando métricas objetivas, por lo que se implementan varias de estas métricas, como la precisión, recall, F1 score, y se genera una matriz de confusión, estas métricas, brindan una visión más completa del desempeño del modelo, más allá de la simple precisión de lo que se implementa normalmente.

La matriz de confusión, en particular, permite visualizar los verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, haciendo ver una representación clara de las fortalezas y debilidades del modelo en la clasificación, asimismo, las métricas de recall y precisión, y su combinación en el F1 score, son fundamentales para entender cómo el modelo maneja el equilibrio entre la sensibilidad y la especificidad.

```
def evaluate_model(X, y, params, threshold=0.5):
    """
    Con esto se va a evaluar un modelo de regresión logística utilizando las métricas de precision, recall,
    f1 score y una matriz de confusión.

    lo que se piensa devolver para esta seccion es:
    - threshold: el umbral utilizado para la clasificacion.
    """
    y_pred_prob = predict(X, params) # aqui se predeciran los datos usando el modelo entrenado
    y_pred = y_pred_prob >= threshold # todo eso se clasificaran con en el umbral que di

    # Caalculo de metricas de evaluacion, esto es para determinar la evaluacion del modelo una vez corrida la ap
    accuracy = np.mean(y_pred == y)
    precision = np.sum((y_pred == 1) & (y == 1)) / np.sum(y_pred == 1) if np.sum(y_pred == 1) > 0 else 0
    recall = np.sum((y_pred == 1) & (y == 1)) / np.sum(y == 1) if np.sum(y == 1) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    # Generacion de la matriz de confusion
    cm = pd.crosstab(y, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)

    # Imprime las metricas
    print(f"\nAccuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print("\nMatriz de confusión:")
    print(cm)

    return threshold
```

Ejecución: main.py

El archivo actúa como controlador principal que une todos los componentes que se realizaron, lo que se hace aquí es que se coordina la preparación de los datos, el entrenamiento del modelo, su evaluación y la generación de predicciones

.

Finalmente, se establece el cómo funciona: Se llama a las funciones de `data_preparation.py` para preparar los datos, luego se entrena el modelo usando `logistic_regression.py`, y finalmente se evalúa y prueba el modelo con las funciones en `evaluation.py`, por último, se proporciona la funcionalidad para ingresar nuevos datos y obtener predicciones en tiempo real, demostrando la funcionalidad del modelo.

```
# Preparacion de los datos, aqui se carga el archivo y se mandan los train, validation y test
X, y, means, stds = prepare_data('data/diabetes.csv')
X_train, X_validation, X_test, y_train, y_validation, y_test = split_data(X, y)

# se inicializan los parámetros del modelo
params = np.zeros(X_train.shape[1])

# aqui son los parametros del modelo, aqui practicamente fui jugando con los valores hasta terminar
# en cual es mejor para el modelo
learning_rate = 0.02
epochs = 1000
reg_strength = 0.1

# aqui ya se empezara a entrenar el modelo
params = gradient_descent(X_train, y_train, params, learning_rate, epochs, reg_strength)

# aqui el modelo es evaluado para la validación del mismo
print("Evaluación en el conjunto de validación:")
threshold = evaluate_model(X_validation, y_validation, params)

# y por ultimo se imprime lo que es la evaluacion de prueba
print("\nEvaluación en el conjunto de prueba:")
evaluate_model(X_test, y_test, params)
```

```
print("\nIntroduce los valores del nuevo paciente para hacer una predicción:")
pregnancies = float(input("Pregnancies: "))
glucose = float(input("Glucose: "))
blood_pressure = float(input("BloodPressure: "))
skin_thickness = float(input("SkinThickness: "))
insulin = float(input("Insulin: "))
bmi = float(input("BMI: "))
dpf = float(input("DiabetesPedigreeFunction: "))
age = float(input("Age: "))

# Creacion del array de características del nuevo paciente
new_data = np.array([[pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dpf, age]])

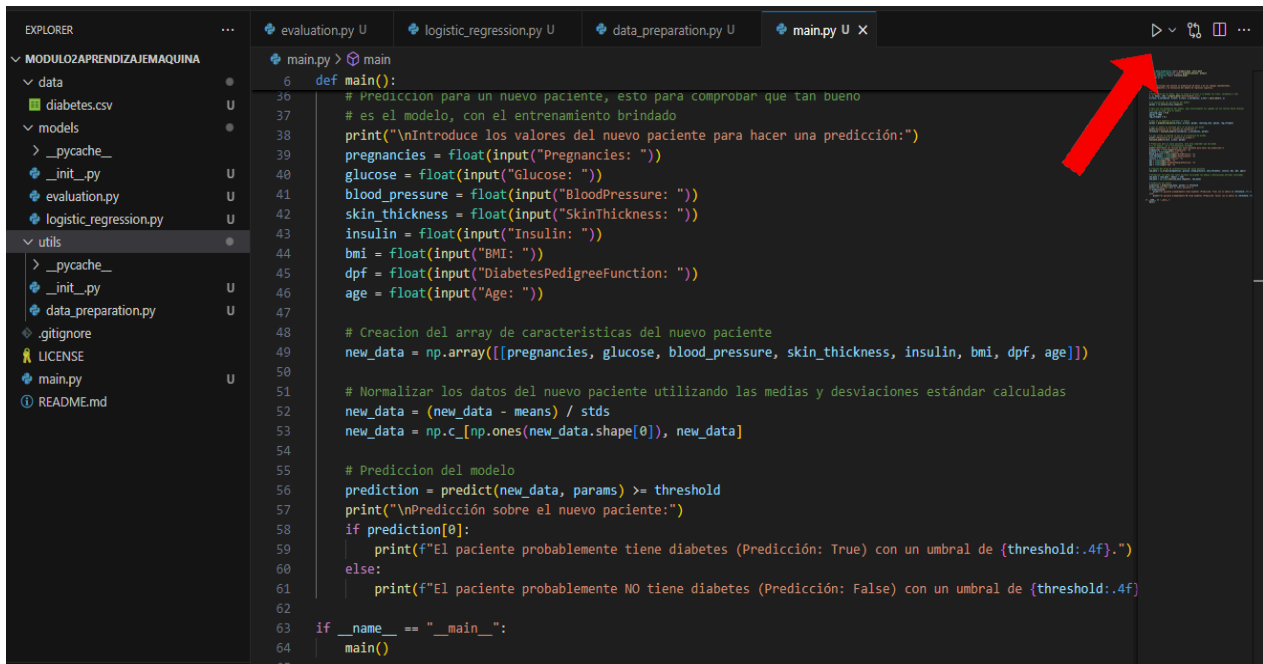
# Normalizar los datos del nuevo paciente utilizando las medias y desviaciones estándar calculadas
new_data = (new_data - means) / stds
new_data = np.c_[np.ones(new_data.shape[0]), new_data]

# Prediccion del modelo
prediction = predict(new_data, params) >= threshold
print("\nPredicción sobre el nuevo paciente:")
if prediction[0]:
    print(f"El paciente probablemente tiene diabetes (Predicción: True) con un umbral de {threshold:.4f}.")
else:
    print(f"El paciente probablemente NO tiene diabetes (Predicción: False) con un umbral de {threshold:.4f}.")

__name__ == "__main__":
    main()
```


Manual de usuario

Para la inicialización del modelo se debe permanecer en el archivo main.py, se presiona en el botón de run del ambiente, y el modelo iniciará su arranque de la demostración.



```
6 def main():
7     # Predicción para un nuevo paciente, esto para comprobar que tan bueno
8     # es el modelo, con el entrenamiento brindado
9     print("\nIntroduce los valores del nuevo paciente para hacer una predicción:")
10    pregnancies = float(input("Pregnancies: "))
11    glucose = float(input("Glucose: "))
12    blood_pressure = float(input("BloodPressure: "))
13    skin_thickness = float(input("SkinThickness: "))
14    insulin = float(input("Insulin: "))
15    bmi = float(input("BMI: "))
16    dpf = float(input("DiabetesPedigreeFunction: "))
17    age = float(input("Age: "))
18
19    # Creación del array de características del nuevo paciente
20    new_data = np.array([[pregnancies, glucose, blood_pressure, skin_thickness, insulin, bmi, dpf, age]])
21
22    # Normalizar los datos del nuevo paciente utilizando las medias y desviaciones estándar calculadas
23    new_data = (new_data - means) / stds
24    new_data = np.c_[np.ones(new_data.shape[0]), new_data]
25
26    # Predicción del modelo
27    prediction = predict(new_data, params) >= threshold
28    print("\nPredicción sobre el nuevo paciente:")
29    if prediction[0]:
30        print(f"El paciente probablemente tiene diabetes (Predicción: True) con un umbral de {threshold:.4f}.")
31    else:
32        print(f"El paciente probablemente NO tiene diabetes (Predicción: False) con un umbral de {threshold:.4f}")
33
34    if __name__ == "__main__":
35        main()
```

Una vez habiendo iniciado se mostrarán los datos sobre cómo se entrenó el modelo y además se le pedirá al usuario ingresar nuevos valores para un nuevo paciente y así determinará si este cuenta o no cuenta con la enfermedad diabetes.

Evaluación en el conjunto de prueba:

Accuracy: 0.8017
Precision: 0.6667
Recall: 0.4667
F1 Score: 0.5490

Matriz de confusión:

Predicted	False	True	All
Actual			
0	79	7	86
1	16	14	30
All	95	21	116

Introduce los valores del nuevo paciente para hacer una predicción:
Pregnancies:

Algunos de los ejemplos reales tomados en internet serían los siguientes:

Ejemplo de un Paciente con Diabetes (Predicción: True)

- Pregnancies: 7
- Glucose: 160 (nivel elevado de glucosa en sangre)
- BloodPressure: 90 (presión arterial alta)
- SkinThickness: 40 (grosor de la piel elevado)
- Insulin: 200 (nivel alto de insulina)
- BMI: 37.5 (obesidad)
- DiabetesPedigreeFunction: 0.85 (muy alta predisposición genética)
- Age: 55

```
Introduce los valores del nuevo paciente para hacer una predicción:
Pregnancies: 7
Glucose: 160
BloodPressure: 90
SkinThickness: 40
Insulin: 200
BMI: 37.5
DiabetesPedigreeFunction: 0.85
Age: 55

Predicción sobre el nuevo paciente:
El paciente probablemente tiene diabetes (Predicción: True) con un umbral de 0.5000.
```

Ejemplo de un Paciente al Límite (Predicción: False)

- Pregnancies: 3
- Glucose: 125 (un poco elevado, cerca del límite superior normal)
- BloodPressure: 80 (normal, aunque en el límite superior)
- SkinThickness: 30 (normal, pero en el rango superior)
- Insulin: 105 (ligeramente elevado, pero aún dentro del rango normal)
- BMI: 29.0 (sobrepeso, pero no obesidad)
- DiabetesPedigreeFunction: 0.5 (predisposición genética moderada)
- Age: 40

```
Introduce los valores del nuevo paciente para hacer una predicción:
Pregnancies: 3
Glucose: 125
BloodPressure: 80
SkinThickness: 30
Insulin: 105
BMI: 29
DiabetesPedigreeFunction: 0.5
Age: 40

Predicción sobre el nuevo paciente:
El paciente probablemente NO tiene diabetes (Predicción: False) con un umbral de 0.5000.
```

Interpretación de los resultados

Durante el entrenamiento, el modelo fue optimizado utilizando descenso de gradiente, como se observa en los valores de la función de **loss** que fueron calculados en intervalos de 1000 épocas, por lo que, al inicio del entrenamiento, la pérdida era de 0.6932, lo cual a mi entendimiento es consistente con el comportamiento de un modelo de regresión logística antes de cualquier ajuste, a medida que se avanzó en las épocas, la pérdida disminuyó, alcanzando un valor mínimo de 0.5162 en la época 400.

```
Epoch 0: Loss = 0.6932
Epoch 100: Loss = 0.5472
Epoch 200: Loss = 0.5220
Epoch 300: Loss = 0.5164
Epoch 400: Loss = 0.5162
Epoch 500: Loss = 0.5181
Epoch 600: Loss = 0.5209
Epoch 700: Loss = 0.5239
Epoch 800: Loss = 0.5270
Epoch 900: Loss = 0.5300
```

Sin embargo, después de este punto, se observa que la pérdida comienza a aumentar nuevamente, indicando un posible overfitting al conjunto de entrenamiento, lo que es una señal de que el modelo ha comenzado a aprender patrones específicos del conjunto de entrenamiento en lugar de generalizar bien en nuevos datos.

Evaluación en el Conjunto de Validación

```
Accuracy: 0.6957
Precision: 0.6000
Recall: 0.5581
F1 Score: 0.5783
```

Matriz de confusión:

Predicted \ Actual	False	True	All
0	56	16	72
1	19	24	43
All	75	40	115

El modelo fue evaluado en el conjunto de validación para determinar su capacidad de generalización, el accuracy del modelo en este conjunto fue de 0.6957, lo que indica que el modelo clasificó correctamente aproximadamente el 70% de los casos, sin embargo, al observar las métricas de **precisión** (0.6000) y **recall** (0.5581), se puede deducir que el modelo tiene una capacidad moderada para identificar correctamente los casos positivos, que en este caso

son pacientes con diabetes, pero también comete errores al clasificar algunos pacientes como positivos cuando no lo son que en este caso son falsos positivos.

El **F1 score** de 0.5783, que es la media armónica entre la precisión y el recall, hace entender que el modelo tiene y puede tener un desempeño balanceado, aunque puede brindarse algunas mejoras, en la clasificación de los casos positivos. Por último, la matriz de confusión hace ver que, de los 43 pacientes que realmente tienen diabetes, el modelo identificó correctamente a 24, pero clasificó mal a 19 como no diabéticos, este tipo de error es fatal en aplicaciones médicas, donde un falso negativo podría tener consecuencias graves.

Evaluación en el Conjunto de Prueba

El modelo mostró una mejora en la **precisión** cuando se presentó en el conjunto de prueba, alcanzando un valor de 0.8017, pero, esta mejora en la precisión se presentó con un desequilibrio en otras métricas, como lo indica la disminución del recall a 0.5000, lo que significa que el modelo solo identificó correctamente la mitad de los casos positivos en el conjunto de prueba, por otro lado, el **F1 score** fue de 0.5818, dejando ver que, aunque el modelo es más preciso, aún tiene dificultades para balancear la precisión y la sensibilidad.

Evaluación en el conjunto de prueba:

Accuracy: 0.8017
Precision: 0.6957
Recall: 0.5000
F1 Score: 0.5818

Matriz de confusión:

Predicted	False	True	All
Actual			
0	77	7	84
1	16	16	32
All	93	23	116

La matriz de confusión en el conjunto de prueba, muestra que el modelo clasificó correctamente 16 de los 32 casos positivos, mientras que 16 pacientes con diabetes fueron clasificados incorrectamente como no diabéticos.

Conclusión

La implementación manual de la regresión logística permitió una comprensión profunda de cómo este algoritmo maneja los datos y optimiza los parámetros para realizar predicciones, sin embargo, los resultados obtenidos indican que el modelo tiene limitaciones, especialmente en su capacidad para identificar correctamente los casos positivos que en este caso recall nos demuestra, esto considero que puede deberse a varios factores, como un dataset inherentemente ruidoso, ya que dentro del mismo se encuentran de igual manera datos que no tienen relevancia alguna.

Aunado a esto considero que los resultados obtenidos son consistentes con las expectativas para un modelo de regresión logística aplicado a un dataset real como el de diabetes.

Finalmente, el modelo, varía su rango de predicción de accuracy entre 75% y 82%, siendo así mismo con las demás métricas.