Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Escuela de Ingeniería y Ciencias

TC3006C.101

Inteligencia artificial avanzada para la ciencia de datos I

Momento de Retroalimentación: Módulo 2

Alumno:

Gustavo Alejandro Gutiérrez Valdes - A01747869

Profesor:

Jorge Adolfo Ramírez Uresti

Fecha de entrega:

Domingo 1 de agosto de 2024

Algoritmo desarrollado: Regresión logística

El ejercicio planteado trata acerca de hacer una predicción de si un estudiante será admitido a una escuela de renombre, basado en su puntaje del examen de admisión, así como en su promedio acumulado en el nivel educativo anterior utilizando un modelo de regresión lineal.

La función de activación utilizada es la sigmoide, la cual es ideal para problemas binarios porque proporciona una salida en forma de probabilidad entre 0 y 1, lo que facilita la interpretación y permite determinar límites claros (como 0.5) para clasificaciones binarias. Además, es matemáticamente sencilla y fácil de implementar. Esta función se utiliza igualmente en la hipótesis que es la encargada de implementar los parámetros obtenidos en las muestras para así evaluar la función actual y obtener una predicción. Esto se hace por medio de la multiplicación (producto punto) entre los parámetros y datos recibidos.

```python
# Función sigmoide (Activación)
def sigmoide(z):
    return 1 / (1 + np.exp(-z))

# Hipótesis de regresión logística (Aplicando Sigmoide)
def hipotesis(params, samples):
    valor_acumulado = np.dot(params, samples)
    return sigmoide(valor_acumulado)
```

La función de costo seleccionada es la Cross-Entropy (CE), que genera gradientes significativos para corregir grandes errores en las predicciones, ayudando al modelo a ajustarse de manera eficiente. A medida que el modelo se aproxima a la solución óptima, los gradientes más pequeños facilitan la convergencia precisa que es un detalle muy importante para Gradient Descent.

Esta función también penaliza de forma severa los errores, mejorando la precisión del modelo. Se aplica la fórmula matemática vista en clase, pero con el uso de la función de clip de numpy con el objetivo de prevenir el cálculo de un logaritmo de 0, dando un rango de valores válidos para H. Finalmente, se calcula el error medio que será utilizado para determinar si el modelo ya esta lo suficientemente entrenado, así como para la visualización de la evolución de este.

```python
# Función de costo: Entropía cruzada
def cross_entropy_loss(params, samples, valor_y):
    sum_error = 0
    for i in range(len(samples)):
        h = hipotesis(params, samples[i])
        h = np.clip(h, 1e-10, 1 - 1e-10) # Manejo de log(0) para evitar errores
        error = -valor_y[i] * np.log(h) - (1 - valor_y[i]) * np.log(1 - h)
        sum_error += error
    error_medio = sum_error / len(samples)
    print("Error medio (entropía cruzada): " + str(error_medio))
    errores.append(error_medio)
    return error_medio
```

Para la evolución del modelo, se emplea el Gradient Descent (GD), una técnica versátil que no depende de la función de costo específica, lo que la convierte en una herramienta flexible. A través de iteraciones, permite una mejora gradual del modelo, siendo menos costosa que encontrar una solución cerrada.

La capacidad de elegir el learning rate otorga control sobre la velocidad y precisión del aprendizaje. Esta función regresa nuevos parámetros que será utilizados en la siguiente iteración, con lo que conforme el paso de estas los parámetros se van ajustando para dar mejores resultados. Primero se recorre cada característica para el cálculo del gradiente, y se inicializa la suma de errores. Después se calcula la diferencia (error) entre la predicción y el valor real, para después multiplicarla por la característica y sumarlo a 'sum_error' y promediar el gradiente acumulado para cada característica. Finalmente se actualizan los parámetros restando el valor actual menos la tasa de aprendizaje por el gradiente acumulado.

```
def GradientDescent(params, samples, learn_rate, valor_y):
    params = np.array(params)
    num_samples = len(samples)
    num_caracteristicas = len(params)
    params_nuevos = np.zeros(num_caracteristicas)

    for j in range(num_caracteristicas):
        sum_error = 0
        for i in range(num_samples):
            error = hipotesis(params, samples[i]) - valor_y[i]
            sum_error += error * samples[i][j]
        params_nuevos[j] = (1 / num_samples) * sum_error

    return params - learn_rate * params_nuevos
```

La normalización se realiza mediante una escala de características (Min-Max), que asegura una contribución equilibrada de todas las características al modelo, evitando que las escalas más grandes dominen la función de costo. Este método es efectivo en combinación con Gradient Descent, ya que facilita una convergencia más rápida.

Además, mejora la comparabilidad entre características, haciendo más sencilla la interpretación de los pesos en modelos como la regresión logística, y su implementación es sencilla, incluso al añadir nuevos datos al conjunto de muestras. Esta función regresa las muestras normalizadas, así como los valores mínimos y la diferencia que es la resta entre los máximos y los mínimos. De igual forma se utiliza la función de clip de numpy para evitar la división entre 0 que puede originar errores.

Adicionalmente, se implementó una nueva función de normalización para los nuevos datos, esto con el objetivo de que este proceso para estos datos se realice con los datos obtenidos durante el entrenamiento del modelo para así mantener una distribución de datos parecida a la obtenida en los datos de entrenamiento y de igual forma mantener una escala como la que el modelo ha aprendido, evitando desestabilizar o sesgar las predicciones del modelo.

```python
def Normalizacion(samples):
    samples = np.array(samples, dtype=float)
    min_vals = np.min(samples, axis=0)
    max_vals = np.max(samples, axis=0)
    diferencia = max_vals - min_vals

    diferencia = np.clip(diferencia, a_min=1e-8, a_max=None) #Evita la división entre 0

    normalized_samples = (samples - min_vals) / diferencia

    return normalized_samples.tolist(), min_vals, diferencia

def Normalizacion_nuevos_datos(nuevos_datos, min_vals, range_vals):
    nuevos_datos_norm = (np.array(nuevos_datos) - min_vals) / range_vals
    return nuevos_datos_norm
```

Finalmente, tenemos la función donde se realiza la regresión logística, implementando todos los métodos anteriormente mencionados. Primero se normalizan las muestras recibidas, y después a cada una se le agrega el término intercepto. Se genera un loop “infinito” que terminará en el momento en que los parámetros ya no sufran cambios o que el valor del error sea menor a 0.01.

Dentro de este bucle, se actualizan los parámetros calculados por la función de GD y de igual manera se obtiene el error actual con la función de CE. Se sigue este proceso hasta que alguna de las condiciones de fin del loop se cumplan. Adicionalmente, se hacen algunas impresiones como la época en la que se encuentra, los parámetros obtenidos en esta iteración y un mensaje que indica que el entrenamiento finalizo exitosamente. Finalmente, esta función regresa los parámetros correctos para realizar las predicciones, así como los valores mínimos y la diferencia entre estos y los máximos para normalizar los nuevos datos que se utilicen.

```python
def logistic_regression(params, samples, valor_y, learning_rate):
    samples, min_vals, range_vals = Normalizacion(samples)

    for i in range(len(samples)):
        samples[i] = [1] + samples[i]

    epochs = 0
    while True:
        oldparams = np.array(params)
        print(f"Epoch #: {epochs}")
        print("Parametros actuales: " + str(params))
        params = GradientDescent(params, samples, learning_rate, valor_y)
        error = cross_entropy_loss(params, samples, valor_y)
        print(params)
        epochs += 1
        # Verifica si los parámetros han cambiado suficientemente
        if np.allclose(oldparams, params, atol=1e-6) or error < 0.01:
            print("************************************")
            print("ENTRENAMIENTO FINALIZADO")
            break
    return params, min_vals, range_vals
```

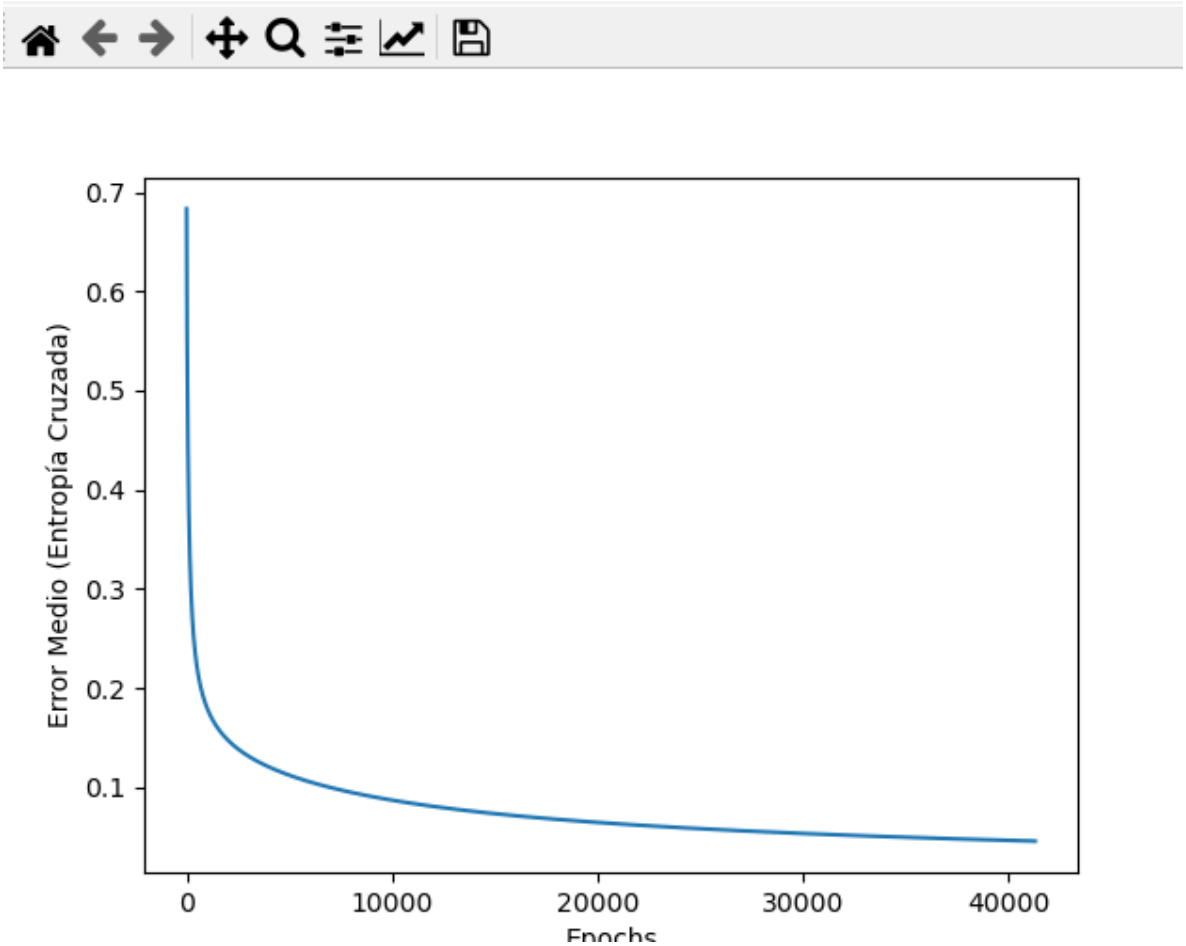
En el siguiente fragmento de código, se inicializan los parámetros que cambiarán una vez ejecutado el modelo para así realizar las predicciones. Después se obtiene el dataset de entrenamiento del archivo “train.csv” y se asigna un valor al learning rate, para que finalmente se ejecute la regresión logística.

```
if __name__ == "__main__":
    # Coeficiente para las características del estudiante y el término independiente
    params = [0, 0, 0]

    # Conjunto de características de los estudiantes (Puntaje de examen, Promedio acumulado) - Set de testing
    training_data = pd.read_csv('train.csv')
    datos_estudiantes = training_data[['PuntajeExamen','PromedioAcumulado']]

    # Este es el estado de su admisión. (0 - No admitido / 1 - Admitido)
    admision = training_data['Admision']

    #Este es el learning rate que se utilizará con el modelo
    learning_rate = 0.01

    #Aquí se ejecuta la función del modelo y se obtienen valores que serán utilizados para el testing
    params_finales, min_vals, range_vals = logistic_regression(params, datos_estudiantes, admision, learning_rate)
```

Con el siguiente fragmento de código, se realiza una gráfica donde se puede apreciar el comportamiento del valor del error medio conforme avanzan las épocas y el modelo aprende utilizando la librería de matplitlib y asignando los labels para cada eje, buscando así un mejor entendimiento de esta.

```
import matplotlib.pyplot as plt
plt.plot(errores)
plt.xlabel('Epochs')
plt.ylabel('Error Medio (Entropía Cruzada)')
plt.show()
```

El resultado es el siguiente:



Aquí podemos notar como el valor del error medio fue disminuyendo conforme se entrenaba el modelo, viendo así una evolución en los resultados. El número de épocas necesario fue alrededor de 41000, es por esto por lo que no fue considerado dentro de las condiciones para dar fin al modelo, ya que este se vislumbraba que requería ser muy grande, pero garantiza obtener los mejores resultado con la regresión logística.

Entrando a la etapa de validación del modelo, primero inicializamos los valores de los distintos cuadrantes de la matriz de confusión (True Positive, True Negative, False Positive, False Negative) que nos ayudarán a calcular algunas métricas para evaluar el funcionamiento del modelo. Así mismo, se obtiene el fragmento del dataset destinado a la validación y se

incorpora al modelo. Finalmente, este se recorre para normalizar cada registro y así aplicarle la regresión logística obteniendo una predicción que se resumirá en un valor binario (0 o 1) que después se comparará con el valor real, y se podrá identificar la capacidad del modelo para realizar predicciones con datos que no ha visto.

```
#Aqui se entra a la etapa de validación
print("************************************")
print("VALIDACIONES DEL MODELO")
print("************************************")

# Inicializar contadores
TP = 0
TN = 0
FP = 0
FN = 0

validation_data = pd.read_csv('validation.csv')
datos_validation = validation_data[['PuntajeExamen','PromedioAcumulado']].values
resultados_validation = validation_data[['Admision']].values
contador = 1

for estudiante,resultado in zip (datos_validation,resultados_validation):
    estudiante_validation_normalizado = Normalizacion_nuevos_datos(estudiante,min_vals,range_vals)
    estudiante_validation_normalizado = [1] + estudiante_validation_normalizado.tolist()
    probabilidad_validation = hipotesis(params_finales,estudiante_validation_normalizado)
    if probabilidad_validation < 0.5:
        resultado_validation = 0
        if resultado_validation == resultado:
            print(f"{contador}) La predicción fue correcta")
        else:
            print(f"{contador})La predicción no fue correcta")
    else:
        resultado_validation = 1
        if resultado_validation == resultado:
            print(f"{contador})La predicción fue correcta")
        else:
            print(f"{contador})La predicción no fue correcta")
    contador += 1
```

Después de aplicar la regresión lineal a los registros, se compara el valor obtenido con el valor real que igualmente se encuentra presente en el dataset. De esta manera, se puede verificar si es que los resultados obtenidos son correctos o no.

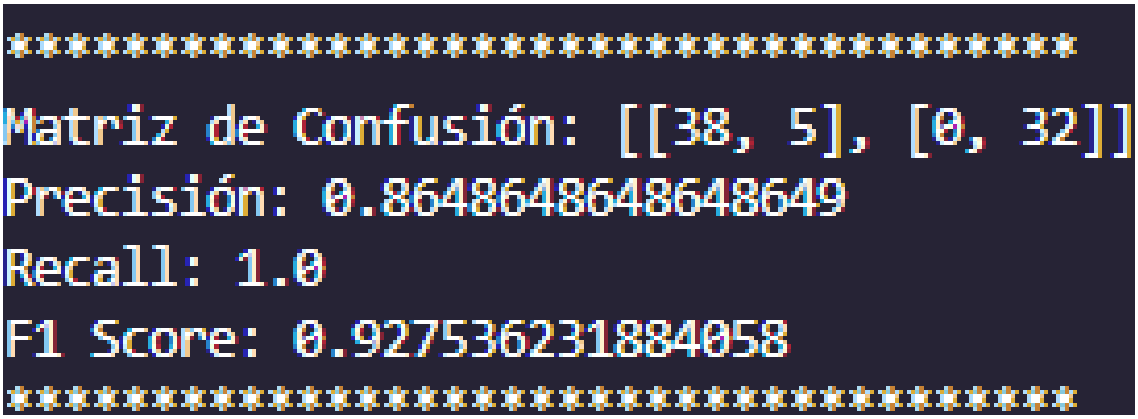
Utilizando los datos arrojados por la etapa de validación, se modifica el valor de los cuadrantes de la matriz de confusión para poder calcular las distintas métricas. Aquí se comparan los valores predichos con los reales, y se suma su indicador correspondiente.

```
if resultado_validation == 1 and resultado == 1:
    TP += 1
elif resultado_validation == 0 and resultado == 0:
    TN += 1
elif resultado_validation == 1 and resultado == 0:
    FP += 1
elif resultado_validation == 0 and resultado == 1:
    FN += 1
```

Siguiendo con este proceso, ahora se realiza el cálculo de la matriz de confusión, de la precisión, del recall y el f1 score para poder entender de mejor manera el desempeño del modelo. Estos cálculos se realizaron siguiendo las fórmulas matemáticas estudiadas anteriormente.

```
# Calcular las métricas
precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1_score = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0

# Matriz de confusión
matriz_confusion = [[TN, FP], [FN, TP]]
```

Los resultados obtenidos fueron los siguientes:



Se puede notar que el modelo genera algunos falsos positivos, pero en general es un modelo confiable para realizar predicciones al menos en el escenario para el que se va a utilizar. El modelo tiene una precisión del 86.5%, lo que indica que la mayoría de las veces que predice un positivo, es correcto, pero tiene algunas dificultades con el aspecto antes mencionado.

El recall es 1.0, lo que significa que el modelo está capturando todos los casos positivos correctamente y para terminar el F1 Score de 0.9275 indica un buen equilibrio entre precisión y recall, que era uno de los objetivos que tenía al inicio del desarrollo. Finalmente, se entra en la última etapa que es la de testing, donde se realizaran predicciones de valores no antes vistos y de los que no se conocen los valores reales. El proceso es muy similar al ejecutado en la etapa de validation, solo que ahora se imprime la predicción acerca del estado de admisión de los alumnos.

```python
#Aqui se entra a la etapa de testing
print("************************************")
print("PREDICCIONES PARA EL SET DE TESTING")
print("************************************")

contador = 1
testing_data = pd.read_csv('test.csv')
nuevos_estudiantes = testing_data[['PuntajeExamen','PromedioAcumulado']].values
for estudiante in nuevos_estudiantes:
    nuevo_estudiante_normalizado = Normalizacion_nuevos_datos(estudiante, min_vals, range_vals)
    nuevo_estudiante_normalizado = [1] + nuevo_estudiante_normalizado.tolist()
    probabilidad = hipotesis(params_finales, nuevo_estudiante_normalizado)
    if probabilidad < 0.5:
        print(f"{contador}) El nuevo estudiante será admitido en la universidad")
    else:
        print(f"{contador}) El nuevo estudiante no será admitido en la universidad ")
    contador += 1
```

Las predicciones obtenidas fueron las siguientes:

```
PREDICCIONES PARA EL SET DE TESTING
************************************
1) El nuevo estudiante será admitido en la universidad
2) El nuevo estudiante será admitido en la universidad
3) El nuevo estudiante no será admitido en la universidad
4) El nuevo estudiante será admitido en la universidad
5) El nuevo estudiante será admitido en la universidad
6) El nuevo estudiante no será admitido en la universidad
7) El nuevo estudiante no será admitido en la universidad
8) El nuevo estudiante será admitido en la universidad
9) El nuevo estudiante no será admitido en la universidad
10) El nuevo estudiante no será admitido en la universidad
11) El nuevo estudiante será admitido en la universidad
12) El nuevo estudiante será admitido en la universidad
13) El nuevo estudiante no será admitido en la universidad
14) El nuevo estudiante será admitido en la universidad
15) El nuevo estudiante será admitido en la universidad
16) El nuevo estudiante será admitido en la universidad
17) El nuevo estudiante no será admitido en la universidad
18) El nuevo estudiante será admitido en la universidad
19) El nuevo estudiante será admitido en la universidad
20) El nuevo estudiante será admitido en la universidad
```

Por lo tanto, se puede concluir que el modelo tiene la capacidad de generar predicciones correctas para el escenario que se planteó, aunque tiene algunas áreas de mejorar sobre todo en el tema de los falsos positivos. Algunos puntos a mejorar podrían ser el trabajar con un dataset aún más grande, o seguir experimentando con el learning rate pero sin tener la certeza de que estos cambios tengan un impacto significativo en los resultados obtenidos.