Laboratorio Regresión Logística

Emmanuel del Piero Martínez Salcedo
Ciencia de la Computación
Universidad Católica San Pablo
Arequipa, Perú
emmanuel.martinez@ucsp.edu.pe

I. Introducción

En el análisis de datos enfocado a la clasificación binaria, la Regresión Logística [1] es una técnica estadística esencial utilizada para modelar la probabilidad de que un evento ocurra, en función de una o más variables independientes. A diferencia de la Regresión Lineal, su objetivo no es predecir un valor continuo, sino una probabilidad que pueda ser interpretada como una clase: por ejemplo, 0 o 1.

Este trabajo de laboratorio aplica la regresión logística al campo de la inmunología computacional, donde se busca predecir si una determinada región de una proteína invasora (antígeno) puede inducir la producción de anticuerpos por parte del sistema inmune.

Para evaluar la calidad del modelo, se utilizan métricas como la precisión (accuracy), la matriz de confusión, la tasa de verdaderos positivos (TPR) y la tasa de verdaderos negativos (TNR). Estas métricas permiten analizar el rendimiento del modelo tanto en el conjunto de entrenamiento como en los de validación y prueba. Así, se busca comprender cómo los datos y las características elegidas afectan la capacidad del modelo para hacer predicciones precisas y útiles en el ámbito de la inmunología predictiva.

II. Implementación

Para la implementación se crearon las funciones principales de balancear, normalización, *sigmoid*, costo, y la gradiente descendiente.

A. Dataset

Los datos usados son obtenidos del *dataset proteins_set* que contiene datos sobre las características de distintas proteinas incluyendo su capacidad de producir anticuerpos *Target*, este *dataset* contiene sus datos ya separados para la fase de validación y experimentación. En el siguiente mapa de calor se puede observar la dependencia de los datos con la característica *Target* para seleccionar las características mas representaticas

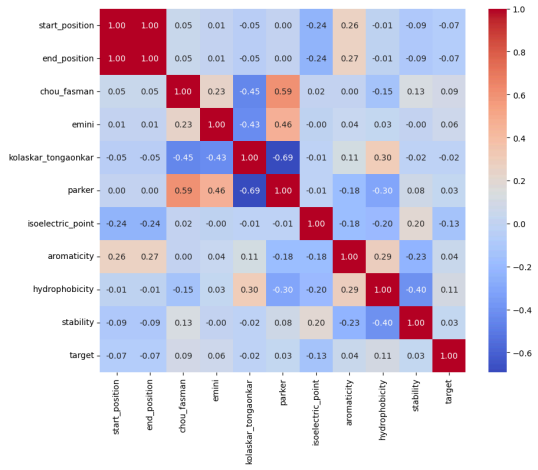

Fig. 1: Mapa de calor de las características del *dataset*.

B. Funciones

a) *Balanceo de los datos:* Para el balanceo de los datos se dicidió hacer un *oversamplig* para aumentar aleatoriamente la cantidad de muestras de la clase minoritaria haciendo copias ya que al realizar un undersampling quedan alrededor de ~2000 muestras para cada clase y podrían ser muy pocas para el entrenamiento. Fig. 2

```python
def balancear(df, target_col='target', oversample=False):
  """
  Balancea un DataFrame df basado en la columna target.
  """
  counts = df[target_col].value_counts()

  clase_mayoritaria = counts.idxmax()
  clase_minoritaria = counts.idxmin()

  df_min = df[df[target_col] == clase_minoritaria]
  df_maj = df[df[target_col] == clase_mayoritaria]

  if oversample:
    df_min_balanceado = df_min.sample(n=len(df_maj), replace=True, random_state=42)
    df_balanceado = pd.concat([df_min_balanceado, df_maj], ignore_index=True)
  else:
    df_maj_balanceado = df_maj.sample(n=len(df_min), random_state=42)
    df_balanceado = pd.concat([df_min, df_maj_balanceado], ignore_index=True)

  df_balanceado = df_balanceado.sample(frac=1, random_state=42).reset_index(drop=True)

  return df_balanceado
```

Fig. 2: Función para el del *dataframe*.

b) *Normalización de datos:* Se uso la función de normalización *Min-max*, esta normalización es necesaria ya que la cantidad poblacional de los distintos estados y condados es muy variable, en algunos estados superando el millón. Fig. 3

$$x_{\text{new}} = \frac{x_{\text{old}} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{1}$$

```python
def normalizar(X):
  """
  Normaliza los datos de entrada X utilizando la normalización min-max.
  """
  min_vals = np.min(X, axis=1, keepdims=True)
  max_vals = np.max(X, axis=1, keepdims=True)
  X_normalizado = (X - min_vals) / (max_vals - min_vals)
  return X_normalizado
```

Fig. 3: Función para la normalización del *dataframe*

c) *Sigmoid:* La función sigmoid se usa para transformar cualquier valor real en un valor probabilístico entre 0 y 1. Fig. 4

```python
def sigmoid(z):
    """
    Funcion sigmoide.
    """
    return 1 / (1 + np.exp(-z))
```

Fig. 4: Función sigmoide.

d) *Función de costo:* La función de costo en Regresión Logística mide el error entre las predicciones del modelo y los valores reales. Utiliza la log-loss para penalizar las predicciones incorrectas, asignando un costo más alto cuando la probabilidad predicha se aleja del valor real (0 o 1). El objetivo es minimizar esta función para ajustar los parámetros del modelo de manera que las predicciones sean lo más precisas posibles. Fig. 5

```python
def costo(x, y, theta):
    """
    Funcion de costo para regresion logistica.
    """
    h = sigmoid(np.dot(theta.T, x))
    if y == 1:
        return -np.log(h)
    elif y == 0:
        return -np.log(1 - h)
    else:
        raise ValueError("La etiqueta y debe ser 0 o 1.")
```

Fig. 5: Funcion de Costo.

e) *Gradiente descendiente:* La función de gradiente descendente se usa para ajustar los parámetros del modelo. Consiste en calcular el gradiente de la función de costo con respecto a los parámetros y actualizar estos parámetros en la dirección opuesta al gradiente, reduciendo gradualmente el error en cada iteración. Esto se repite hasta que los parámetros convergen hacia los valores óptimos que minimizan la función de costo.. Fig. 6

```python
def gradiente_descendiente(x, y, theta, alpha, num_iters):
    """
    Funcion de gradiente descendiente para optimizar los parametros theta.
    """
    m = x.shape[1]

    for _ in range(num_iters):
        Z = np.dot(theta.T, x)
        H = sigmoid(Z)
        gradiente = (1/m) * np.dot(x, (H - y).T)
        theta = theta - alpha * gradiente
    return theta
```

Fig. 6: Gradiente descendiente.

III. Experimentos y Resultados

Para todos los modelos, se limpia el *dataframe* de entrenamiento (aunque se notó que en este *dataset* no existen filas vacías o NaN), se realiza el balancedo de la clase con *oversampling*, se separa el *dataframe* para usar en Y la varaible de clase *Target* y itera en la función de gradiente para obtener la función de regresión logística para finalmente calcular el costo final. Para los entrenamientos se uso un valor de aprendizaje *alpha* de 0.1 y un total de 10000 iteraciones. Fig. 7

```python
df_train = pd.read_csv('proteins_training_set.csv')

# Visualización de la distribución de clases
target_counts = df_train['target'].value_counts()
print(f"Total de datos con target 1: {target_counts.get(1, 0)}")
print(f"Total de datos con target 0: {target_counts.get(0, 0)}")

df_train = df_train.dropna()

# Visualización de la distribución de clases
corr_matrix = df_train.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f',
cbar=True)
plt.show()

df_train_balanceado = balancear(df_train, oversample=True)

X = df_train_balanceado.drop(columns=['target']).values.T
X = normalizar(X)
Y = df_train_balanceado['target'].values.reshape(1, -1)
X = np.vstack([np.ones((1, X.shape[1])), X])

n = X.shape[0]
theta = np.zeros((n, 1))
alpha = 0.1
epochs = 10000

# Entrenar
theta = gradiente_descendiente(X, Y, theta, alpha, epochs)
```

Fig. 7: Balanceo, normalización y entrenamiento del modelo de Regresión Logística.

A. Revisión de los datos

Como se mencionó anteriormente, se eliminaron las filas del *datafra* si tienen características faltantes o si contienen NaN con un *dropna*. Dado que el *dataset* esta altamente desbalanceado (teniendo un total de datos con target 1: 2495 y total de datos con target 0: 6709) se realizo un balancedo con *oversampling* y se normalizarón los datos.

B. Prueba 1 con todas las características, Modelo Base

Al experimentar con los conjuntos de entrenamiento, validación y prueba se obtuvieron los siguientes resultados. Fig. 8, Fig. 9, Fig. 10.

```
Matriz de Confusión:
TP: 1613 | FN: 882
FP: 2772 | TN: 3937
Precisión en prueba: 0.6030
Tasa de Verdaderos Positivos (TPR): 0.6465
Tasa de Verdaderos Negativos (TNR): 0.5868
```

Fig. 8: Resultados obtenidos en la prueba 1 con el conjunto de entrenamiento.

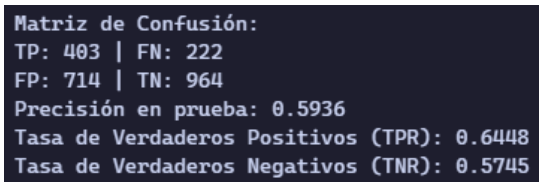

Fig. 9: Resultados obtenidos en la prueba 1 con el conjunto de validación.

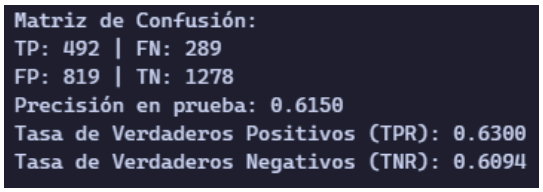

Fig. 10: Resultados obtenidos en la prueba 1 con el conjunto de experimentación.

C. Prueba 2 con las características mas representativas, Modelo Nuevo

Como se puede observar en Fig. 1 no existen características muy representativas para el conjunto de datos con respecto a *Target*, de esta forma se escogieron las cinco más representativas que son *start_position, end_position, chou_fasman, isoelectric_point and hydrophobicity*.

Al experimentar con los conjuntos de entrenamiento, validación y prueba se obtuvieron los siguientes resultados. Fig. 11, Fig. 12, Fig. 13.

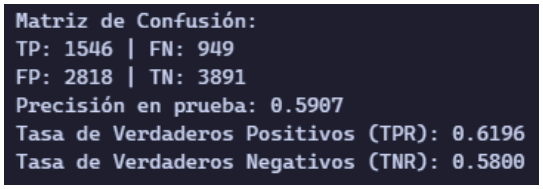

Fig. 11: Resultados obtenidos en la prueba 2 con el conjunto de entrenamiento.

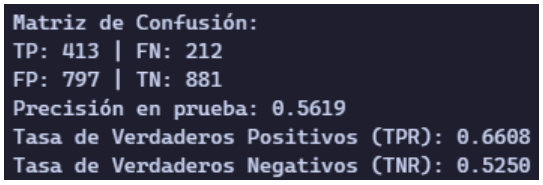

Fig. 12: Resultados obtenidos en la prueba 2 con el conjunto de validación.

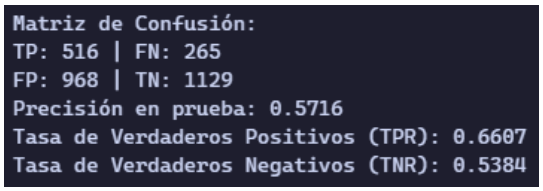

Fig. 13: Resultados obtenidos en la prueba 2 con el conjunto de experimentación.

IV. Conclusiones

Como se observa en los resultados obtenidos, el mejor desempeño lo tuvo el modelo base, que utilizó todas las características disponibles. Esto se debe a que la correlación entre las variables y la variable objetivo *target* es muy baja en todos los casos, por lo que eliminar características supuestamente no representativas no generó mejoras en el rendimiento del modelo.

Además, se observó una precisión baja, cercana al 60% en todos los experimentos. Esto podría explicarse por la falta de variables predictivas relevantes, o bien por una posible insuficiencia de datos para permitir un entrenamiento más efectivo.

Este trabajo de laboratorio cumplió con el objetivo de experimentar con el conjunto de datos provisto y evaluar el desempeño de la Regresión Logística en este contexto. Se identificó que, dada la débil relación entre las características y el *target*, el modelo más adecuado fue aquel que utilizó la totalidad de las variables disponibles, lo cual refuerza la importancia de evaluar la selección de características en función del problema específico.

References

[1] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002.