

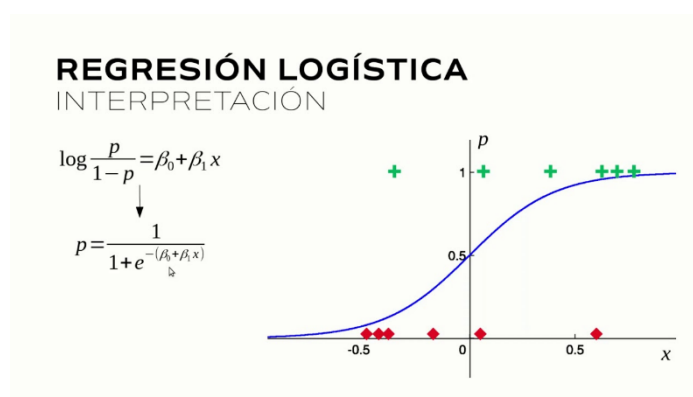


**Centro Universitario de Ciencias Exactas e
Ingenierías**
Universidad de Guadalajara



Práctica 3: Regresión Logística

Aprendizaje Máquina



Alumno: Samuel David Pérez Brambila

Código: 222966286

Profesora: Karla Ávila Cárdenas

Sección: D01

Fecha de Entrega: 06 de Octubre de 2024

Introducción

La regresión logística es “un modelo estadístico para estudiar las relaciones entre un conjunto de variables cualitativas X_i y una variable cualitativa Y . Se trata de un modelo lineal generalizado que utiliza una función logística como función de enlace.” (DataScientest, 2023)

Un modelo de regresión logística también permite predecir la probabilidad de que ocurra un evento (valor de 1) o no (valor de 0) a partir de la optimización de los coeficientes de regresión. Este resultado siempre varía entre 0 y 1. Cuando el valor predicho supera un umbral, es probable que ocurra el evento, mientras que cuando ese valor está por debajo del mismo umbral, no es así.

¿Por qué es importante la regresión logística? De acuerdo con AWS (s.f.), los modelos de machine learning creados mediante regresión logística ayudan a las organizaciones a obtener información procesable a partir de sus datos empresariales. Pueden usar esta información para el análisis predictivo a fin de reducir los costos operativos, aumentar la eficiencia y escalar más rápido. Por ejemplo, las empresas pueden descubrir patrones que mejoran la retención de los empleados o conducen a un diseño de productos más rentable.

A continuación se mencionan algunos beneficios del uso de la regresión logística en comparación con otras técnicas de machine learning:

- **Simplicidad:** Los modelos de regresión logística son matemáticamente menos complejos que otros métodos de machine learning. Por lo tanto, puede implementarlos incluso si nadie de su equipo tiene una profunda experiencia en machine learning.
- **Velocidad:** Los modelos de regresión logística pueden procesar grandes volúmenes de datos a alta velocidad porque requieren menos capacidad computacional, como memoria y potencia de procesamiento. Esto los hace ideales para que las organizaciones que están empezando con proyectos de machine learning obtengan ganancias rápidas.
- **Flexibilidad:** Se puede usar la regresión logística para encontrar respuestas a preguntas que tienen dos o más resultados finitos. También se puede usar para preprocesar datos. Por ejemplo, puede ordenar los datos con un amplio rango de valores, como las transacciones bancarias, en un rango de valores más pequeño y finito mediante la regresión logística.
- **Visibilidad:** El análisis de regresión logística ofrece a los desarrolladores una mayor visibilidad de los procesos de software internos que otras técnicas de análisis de datos. La solución de problemas y la corrección de errores también son más fáciles porque los cálculos son menos complejos.

Además, la regresión logística tiene varias aplicaciones del mundo real en muchos sectores diferentes:

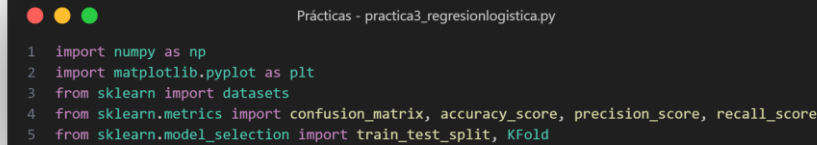
- **Fabricación:** Las empresas de fabricación utilizan el análisis de regresión logística para estimar la probabilidad de fallo de las piezas en la maquinaria. Luego, planifican los programas de mantenimiento en función de esta estimación para minimizar los fallos futuros.
- **Sanidad:** Los investigadores médicos planifican la atención y el tratamiento preventivos mediante la predicción de la probabilidad de enfermedad en los pacientes. Utilizan modelos de regresión logística para comparar el impacto de los antecedentes familiares o los genes en las enfermedades.
- **Finanzas:** Las empresas financieras tienen que analizar las transacciones financieras en busca de fraudes y evaluar las solicitudes de préstamos y seguros en busca de riesgos. Estos problemas son adecuados para un modelo de regresión logística porque tienen resultados discretos, como alto riesgo o bajo riesgo y fraudulento o no fraudulento.
- **Marketing:** Las herramientas de publicidad en línea utilizan el modelo de regresión logística para predecir si los usuarios harán clic en un anuncio. Como resultado, los especialistas en marketing pueden analizar las respuestas de los usuarios a diferentes palabras e imágenes y crear anuncios de alto rendimiento con los que los clientes interactuarán.

En la presente práctica, se trabajará con una regresión logística utilizando el lenguaje de programación Python y algunas de sus bibliotecas más populares, como Matplotlib, SciKit Learn y Numpy. Estas herramientas son fundamentales para el análisis de datos y la visualización de resultados, permitiendo gestionar y manipular grandes conjuntos de datos de manera eficiente.

Contenido de la Actividad

El presente código realiza una regresión logística sobre el dataset de sklearn "Breast cancer wisconsin (diagnostic)", donde se toman las características de "worst radius" (col 0, mide el radio más grande entre los tres núcleos celulares más grandes presentes en la muestra, lo que representa el tamaño de las células) y "mean concavity" (col 8, mide la concavidad media de los contornos de los núcleos celulares, es decir, cuán irregulares o hundidos son sus bordes). Estas características se utilizan como variables independientes para predecir si un tumor es benigno (clase 0) o maligno (clase 1), lo cual es la variable dependiente.

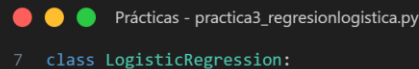
Importación de bibliotecas



```
Prácticas - practica3_regresionlogistica.py
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn import datasets
4 from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
5 from sklearn.model_selection import train_test_split, KFold
```

- numpy: Se usa para trabajar con arreglos y operaciones matemáticas.
- matplotlib.pyplot: Se usa para la visualización de gráficos (curva sigmoide y matriz de confusión).
- sklearn.datasets: Contiene datasets de ejemplo. Aquí se usa el conjunto de datos de cáncer de mama.
- confusion matrix, accuracy score, precision score, recall score: Métricas de evaluación para el rendimiento del modelo.
- train_test_split, KFold: Se usan para dividir el conjunto de datos en conjuntos de entrenamiento y prueba, y para la validación cruzada con K-folds.

Definición de la clase de regresión logística



```
Prácticas - practica3_regresionlogistica.py
7 class LogisticRegression:
```

Se define la clase LogisticRegression, que implementa un modelo de regresión logística personalizado, es decir, que es no es el que viene por defecto con la librería de sklearn.

Constructor de la clase

```
Prácticas - practica3_regresionlogistica.py

7 class LogisticRegression:
8     def __init__(self, learning_rate=0.01, num_iterations=1000):
9         self.learning_rate = learning_rate
10        self.num_iterations = num_iterations
11        self.weights = None
12        self.bias = None
13        self.losses = []
```

- learning_rate: Tasa de aprendizaje del algoritmo (qué tan rápido ajusta los pesos).
- num_iterations: Número de iteraciones para el entrenamiento del modelo.
- weights: Pesos de las características del modelo (inicializados más tarde).
- bias: Sesgo del modelo (se inicializa después).
- losses: Lista que almacena los valores de la función de pérdida en cada iteración, para monitorear el entrenamiento.

Función sigmoide

```
Prácticas - practica3_regresionlogistica.py

15 def sigmoid(self, z):
16     return 1 / (1 + np.exp(-z))
```

Calcula la función sigmoide, que transforma el valor de entrada en un número entre 0 y 1. Esto es clave para la regresión logística.

Función de pérdida

```
Prácticas - practica3_regresionlogistica.py

18 def loss(self, h, y):
19     return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
```

Implementa la función de pérdida (log-loss o binaria). Compara las predicciones (h) con las etiquetas verdaderas (y) y mide el error.

Entrenamiento del modelo

```
Prácticas - practica3_regresionlogistica.py

21 def fit(self, X, y):
22     num_samples, num_features = X.shape
23     self.weights = np.zeros(num_features)
24     self.bias = 0
25
26     for _ in range(self.num_iterations):
27         linear_model = np.dot(X, self.weights) + self.bias
28         y_predicted = self.sigmoid(linear_model)
29         loss = self.loss(y_predicted, y)
30         self.losses.append(loss)
31         dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
32         db = (1 / num_samples) * np.sum(y_predicted - y)
33         self.weights -= self.learning_rate * dw
34         self.bias -= self.learning_rate * db
```

Método que entrena el modelo. Inicializa los pesos a 0 y el sesgo también a 0.

- X.shape: Obtiene el número de muestras (num_samples) y características (num_features) de los datos de entrada X.
- Con apoyo de un bucle for que controla el número de iteraciones del algoritmo de entrenamiento:
 - linear_model: Calcula la combinación lineal de las características y los pesos: $X * \text{weights} + \text{bias}$.
 - y_predicted: Aplica la función sigmoide a la salida de la combinación lineal para obtener probabilidades.
 - loss: Calcula el error o la pérdida usando la función de pérdida y almacena el valor en la lista losses para su seguimiento.
 - dw: Gradiente de los pesos.
 - db: Gradiente del sesgo. Estos gradientes indican cómo ajustar los pesos y el sesgo.
 - self.weights y self.bias: Ajusta los weights y el bias restando una fracción de los gradientes, controlado por learning_rate.

Predicción de etiquetas

```
Prácticas - practica3_regresionlogistica.py

36 def predict(self, X):
37     linear_model = np.dot(X, self.weights) + self.bias
38     y_predicted = self.sigmoid(linear_model)
39     y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
40     return y_predicted_cls
```

Calcula las probabilidades de cada muestra usando la función sigmoide (con las líneas **linear_model** y **y_predicted**), y luego asigna clases binarias (1 para maligno, 0 para benigno) con un umbral de 0.5 (**y_predicted_cls**).

Validación cruzada con K-folds

```
Prácticas - practica3_regresionlogistica.py

42 def cross_validation(self, X, y, k=5):
43     kf = KFold(n_splits=k, shuffle=True, random_state=42)
44     accuracies, precisions, recalls = [], [], []
45
46     for train_index, test_index in kf.split(X):
47         X_train, X_test = X[train_index], X[test_index]
48         y_train, y_test = y[train_index], y[test_index]
49
50         self.fit(X_train, y_train)
51
52         y_pred = self.predict(X_test)
53
54         acc = accuracy_score(y_test, y_pred)
55         prec = precision_score(y_test, y_pred)
56         rec = recall_score(y_test, y_pred)
57
58         accuracies.append(acc)
59         precisions.append(prec)
60         recalls.append(rec)
61
62     print(f"Resultados con validación cruzada de {k} folds:")
63     print(f"Accuracy (Exactitud): {np.mean(accuracies):.4f}")
64     print(f"Precision (Precisión): {np.mean(precisions):.4f}")
65     print(f"Recall (Exhaustividad): {np.mean(recalls):.4f}")
```

Método para realizar validación cruzada K-fold. Divide los datos en k subconjuntos para evaluar el rendimiento del modelo con diferentes conjuntos de entrenamiento y prueba.

- kf: Crea el generador de K-Folds.
- Inicializa listas vacías para almacenar accuracy, precision, y recall en cada iteración (accuracies, precisions, recalls).
- El bucle for, primero divide los datos en conjuntos de entrenamiento y prueba en cada iteración del K-Fold. Entrena el modelo (self.fit) con el conjunto de entrenamiento (X_train, y_train), predice las etiquetas para el conjunto de prueba (y_pred) y calcula las métricas accuracy, precision, y recall; y almacena dichas métricas en las listas correspondientes en cada iteración.
- Los print imprimen los resultados promediados de las k (5) iteraciones de la validación cruzada.

Graficar la sigmoide y las predicciones

```
Prácticas - practica3_regresionlogistica.py

67 def plot_sigmoid_with_predictions(self, X, y):
68     # Graficar la curva sigmoide
69     z = np.linspace(-10, 10, 100)
70     sigmoid_curve = self.sigmoid(z)
71     plt.plot(z, sigmoid_curve, label="Curva Sigmoide", color='b')
72
73     # Predecir valores usando el conjunto X
74     linear_model = np.dot(X, self.weights) + self.bias
75     y_predicted = self.sigmoid(linear_model)
76
77     # Convertir las predicciones en clases (0 o 1)
78     y_predicted_cls = np.array([1 if i > 0.5 else 0 for i in y_predicted])
79
80     # Graficar los puntos predichos (0 o 1) con los colores correspondientes a las etiquetas
81     plt.scatter(linear_model, y_predicted_cls, c=y, edgecolors='k', marker='o', label='Datos Predichos', cmap=plt.cm.RdBu)
82
83     # Título y etiquetas del gráfico
84     plt.title("Regresión logística")
85     plt.xlabel("Combinación Lineal (X * weights + bias)")
86     plt.ylabel("Predicción (0: Benigno, 1: Maligno)")
87     plt.yticks([0, 1]) # Mostrar solo 0 y 1 en el eje y
88     plt.grid(True)
89     plt.legend()
90     plt.show()
```

Este método grafica la curva sigmoide junto a las predicciones realizadas.

Primero, se grafica la curva con apoyo de **z = np.linspace(-10,10,100)** el cual es un array de 100 valores que comienzan en -10, terminan en 10, y están espaciados uniformemente entre esos dos extremos, estos valores de z se utilizarán como entrada para la función sigmoide, por último **plt.plot()** grafica dicha curva resultante.

Luego, calcula la combinación lineal, aplica la función sigmoide y asigna clases (linear_model, y_predicted, y_predict_cls), se usa **plt.scatter()** para graficar los puntos de los datos con las etiquetas. Por último, simplemente se definen los parámetros para el gráfico final, con el título, nombres de ejes, mostrar únicamente 0 y 1 en y (**plt.yticks([0,1])**), **plt.grid()** para mostrar un fondo con cuadrícula, **plt.legend()** para mostrar la simbología y **plt.show()** que permite visualizar el gráfico.

Cargar y dividir el dataset

```
Prácticas - practica3_regresionlogistica.py

92 # Cargar el dataset Breast Cancer de scikit-learn
93 cancer = datasets.load_breast_cancer()
94 X = cancer.data[:, [0, 8]] # "worst radius" (col 0) y "mean concavity" (col 8)
95 y = (cancer.target != 0) * 1 # Clases (0: benigno, 1: maligno)
96
97 # Dividir los datos en conjuntos de entrenamiento y prueba
98 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Carga el conjunto de datos de breast cancer, selecciona dos características (worst radius y mean concavity, las cuales ya se definieron anteriormente) y las etiquetas, y luego se dividen los datos en conjuntos de entrenamiento y prueba.

Entrenar el modelo

```
Prácticas - practica3_regresionlogistica.py

100 # Crear y entrenar el modelo
101 model = LogisticRegression(learning_rate=0.01, num_iterations=10000)
102 model.fit(X_train, y_train)
```

Crea un objeto de `LogisticRegression` con los hiperparámetros deseados (en este caso un `learning_rate` de 0.01 y 10,000 iteraciones) y entrena el modelo con los conjuntos de entrenamiento `X_train`, `y_train`.

Predicciones y matriz de confusión

```
Prácticas - practica3_regresionlogistica.py

107 # Calcular la matriz de confusión
108 cm = confusion_matrix(y_test, y_pred)
109 print("Matriz de confusión")
110 print(cm)
```

Predice las etiquetas con el método `predict` para el conjunto de prueba, calcula la matriz de confusión y la imprime en consola.

```
Prácticas - practica3_regresionlogistica.py

112 # Visualizar la matriz de confusión
113 plt.figure(figsize=(8, 6))
114 plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Oranges)
115 plt.title('Matriz de Confusión')
116 plt.colorbar()
117 plt.xticks([0, 1], ['Benigno', 'Maligno'])
118 plt.yticks([0, 1], ['Benigno', 'Maligno'])
119 plt.xlabel('Etiqueta Predicha')
120 plt.ylabel('Etiqueta Verdadera')
121 plt.show()
```

Se realiza el gráfico para mostrar la matriz de confusión de una manera más llamativa, donde con **`plt.figure(figsize=(8,6))`** se definen las dimensiones de la misma, **`plt.imshow()`** donde se define la escala de colores a utilizar, **`interpolation='nearest'`** este parámetro especifica cómo se deben interpolar los píxeles cuando se amplía la imagen. Se define el título del gráfico, en este caso “Matriz de Confusión” con **`plt.title()`**, **`plt.colorbar()`** el cual es útil ya que nos muestra los valores por la escala de color que se muestra en el gráfico. Las líneas de código **`plt.xticks([0, 1], ['Benigno', 'Maligno'])`** y **`plt.yticks([0, 1], ['Benigno', 'Maligno'])`** se utilizan para configurar las etiquetas de los ejes “x” e “y” en el gráfico de la matriz de confusión. Por último, se definen los ejes que se forman los nombres

anteriormente definidos, es decir, en x Benigno y Maligno representan a Etiqueta Predicha y se muestra dicho gráfico.

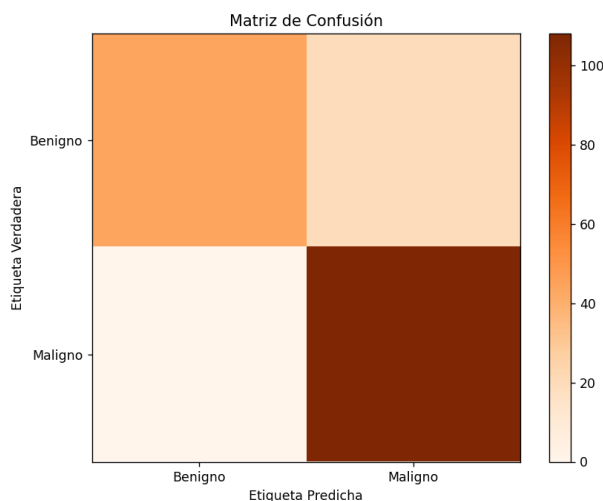
Validación cruzada e impresión de la curva sigmoide con las predicciones

```
Prácticas - practica3_regresionlogistica.py

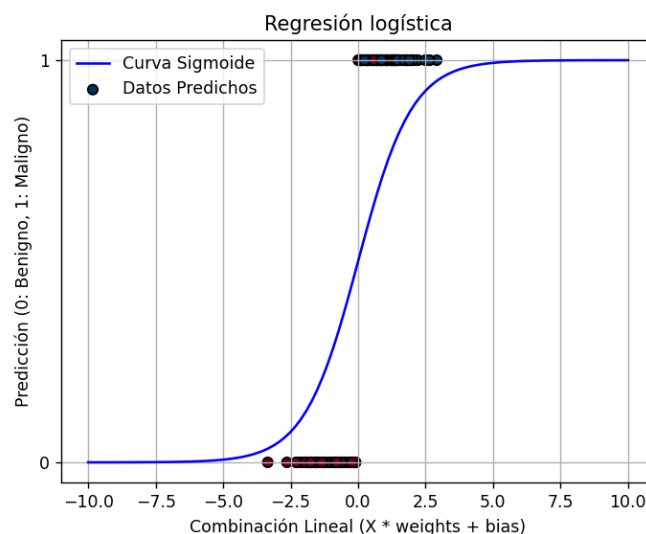
123 # Realizar validación cruzada con K-folds
124 model.cross_validation(X, y, k=5)
125
126 # Visualizar la curva sigmoide y los valores predichos
127 model.plot_sigmoid_with_predictions(X_test, y_test)
```

Realiza la validación cruzada y visualiza los resultados de la regresión logística junto con la curva sigmoide, pasándole los respectivos argumentos a los métodos de la clase LogisticRegression.

Es así que obtenemos los siguientes resultados:



```
Matriz de confusión
[[ 44  19]
 [   0 108]]
Resultados con validación cruzada de 5 folds:
Accuracy (Exactitud): 0.8734
Precision (Precisión): 0.8470
Recall (Exhaustividad): 0.9744
```



Conclusiones

Al concluir la práctica de regresión logística en Python con el apoyo de Scikit-learn, es evidente que este modelo es una herramienta poderosa y versátil para resolver problemas de clasificación binaria. La implementación de la regresión logística resultó ser eficiente y accesible gracias a la simplicidad de la biblioteca Scikit-learn, que permite ajustar modelos de manera rápida con un enfoque modular. A lo largo del proceso, se pudo apreciar la importancia de la preparación adecuada de los datos, así como la elección de las métricas de evaluación correctas para garantizar un análisis riguroso del rendimiento del modelo. Además, la capacidad de ajustar hiperparámetros y realizar validación cruzada proporcionó una forma robusta de evitar el sobreajuste y mejorar la generalización del modelo. También fue notable la efectividad de visualizar la matriz de confusión, lo que permitió comprender mejor cómo se comportó el modelo en términos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Esta visualización, junto con el cálculo de métricas como la precisión, la exactitud y la exhaustividad, facilitó una evaluación más completa del rendimiento.

En resumen, esta práctica no solo ha permitido implementar un modelo de regresión logística, sino que también ha facilitado un entendimiento más profundo de cómo los algoritmos de machine learning pueden ser aplicados de manera eficiente y efectiva utilizando herramientas como lo es el caso de la librería de Scikit-learn en el lenguaje Python. La experiencia ha sido enriquecedora al combinar tanto la teoría como la práctica, proporcionando una base sólida para futuros trabajos en el campo de la ciencia de datos y el aprendizaje automático.

Referencias

- AWS. (s.f.). *¿Qué es la regresión logística?*. Amazon Web Services. Recuperado el 27 de Septiembre de 2024 de: <https://aws.amazon.com/es/what-is/logistic-regression/>
- DataScientest. (2023, 16 de diciembre). *¿Qué es la regresión logística?*. DataScientest. Recuperado el 27 de Septiembre de 2024 de: <https://datascientest.com/es/que-es-la-regresion-logistica>