COLOMBIA POTENCIA DE LA VIDA

TIC

Lección 1

Modelos de regresión

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

UTP Universidad Tecnológica de Pereira

Contenido

1. Introducción a la regresión

Definición

La regresión es una técnica estadística utilizada para modelar la relación entre una variable dependiente (respuesta) y una o más variables independientes (predictores). Se busca entender cómo los cambios en las variables independientes afectan a la variable dependiente.

Importancia en el Aprendizaje Automático

La regresión es una de las técnicas fundamentales en el aprendizaje automático, ya que permite predecir valores continuos basados en variables de entrada. Es muy utilizada en la **predicción de precios, análisis de riesgo crediticio, pronóstico de ventas y muchas otras aplicaciones.**

- **Concepto de Variables Independientes y Dependientes**

Variables independientes

También conocidas como variables predictoras o características, son aquellas que se utilizan para predecir o explicar el valor de la variable dependiente. Son las entradas del modelo.

Variables dependientes

También llamada variable de respuesta, es la variable que se está tratando de predecir o explicar a partir de las variables independientes. Es la salida del modelo.

- **Diferencias entre Regresión y Clasificación:**

La regresión se utiliza para predecir valores continuos, como el precio de una casa o la temperatura.

La clasificación se utiliza para predecir categorías discretas, como si un correo electrónico es spam o no.

- **Algoritmos de Regresión Comunes:**

Algunos algoritmos de regresión comunes incluyen regresión lineal, regresión polinomial y regresión de vecinos más cercanos (KNN). La elección del algoritmo depende de la naturaleza del problema y los datos disponibles.

Creación de un Conjunto de Datos con Variables Independientes y Dependientes:

```
import numpy as np

horas_estudio = np.array([2, 3, 4, 5, 6])

calificacion_examen = np.array([65, 70, 75, 80, 85])
```

Cálculo de la Correlación entre Variables Independientes y Dependientes:

```
def calcular_correlacion(x, y):

    correlacion = np.corrcoef(x, y)[0, 1]

    return correlacion

correlacion = calcular_correlacion(horas_estudio, calificacion_examen)

print("La correlación entre horas de estudio y calificación del examen
es:", correlacion)
```

Ver documentación oficial de np.corrcoef en:

https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html

Graficar un Gráfico de Dispersión de los Datos:

```
import matplotlib.pyplot as plt

plt.scatter(horas_estudio, calificacion_examen)

plt.xlabel('Horas de Estudio')

plt.ylabel('Calificación del Examen')

plt.title('Relación entre Horas de Estudio y Calificación del Examen')

plt.show()
```

Ver documentación oficial de plt.scatter en:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html

Ajustar un Modelo de Regresión Lineal y Visualizar la Línea de Regresión:

```
from sklearn.linear_model import LinearRegression

modelo_regresion = LinearRegression()

modelo_regresion.fit(horas_estudio.reshape(-1, 1),

calificacion_examen)

plt.scatter(horas_estudio, calificacion_examen)
```

```
plt.plot(horas_estudio,

modelo_regresion.predict(horas_estudio.reshape(-1, 1)), color='red')

plt.xlabel('Horas de Estudio')

plt.ylabel('Calificación del Examen')

plt.title('Regresión Lineal: Horas de Estudio vs. Calificación del

Examen')

plt.show()
```

Ver documentación oficial de sklearn.linear_model.LinearRegression en
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

COLOMBIA POTENCIA DE LA VIDA

TIC

El propósito de ofrecer estos ejercicios y explicaciones es proporcionar una comprensión fundamental de la regresión y su aplicación práctica en Python.

TALENTO TECH | AZ PROYECTOS EDUCATIVOS | UTP Universidad Tecnológica de Pereira

2. Regresión lineal

En esta sección, exploraremos el concepto de Regresión Lineal en el contexto de la inteligencia artificial, comprendiendo su aplicación en la predicción de variables continuas, el método de mínimos cuadrados para estimar los parámetros del modelo, así como la interpretación de la pendiente y la intersección.

- **¿Qué es la regresión lineal?**

Es un modelo estadístico que busca establecer una relación lineal entre una variable dependiente y una o más variables independientes. Se representa mediante una ecuación de la forma:

$$y=\beta0+\beta1x1+\beta2x2+...+\beta nxn+\epsilon$$

Donde:

- yy es la variable dependiente que queremos predecir.
- x1,x2,...,xn son las variables independientes.
- β0,β1,...,βn son los coeficientes que representan la pendiente de cada variable independiente.
- ϵ es el término de error.

- **Aplicación en la predicción de variables continuas:**

La Regresión Lineal se utiliza para predecir valores numéricos continuos. Por ejemplo, puede emplearse para predecir el precio de una casa en función de su tamaño, número de habitaciones, ubicación, etc. También se aplica en áreas como:

Economía

Medicina

Ingeniería

Esto es así, porque son campos donde es necesario predecir valores numéricos basados en variables explicativas.

- **Método de mínimos cuadrados:**

El método de mínimos cuadrados es una técnica utilizada para estimar los parámetros del modelo de Regresión Lineal. Consiste en minimizar la suma de los cuadrados de las diferencias entre los valores observados y los valores predichos por el modelo. Esto se logra ajustando los coeficientes del modelo para que la diferencia entre los valores reales y los valores predichos sea lo más pequeña posible.

- **Interpretación de la pendiente y la intersección:**

Pendiente	Intersección
La pendiente ($\beta 1, \beta 2, \ldots, \beta n$) representa el cambio en la variable dependiente yy por unidad de cambio en la variable independiente correspondiente x1,x2,...,xn. Por ejemplo, si la pendiente es 2, significa que por cada unidad adicional en la variable independiente, la variable dependiente aumenta en 2 unidades.	La intersección ($\beta 0$) representa el valor esperado de la variable dependiente cuando todas las variables independientes son iguales a cero. Es el punto en el que la línea de regresión cruza el eje vertical (eje y).

Modelo Lineal y Aplicación en la Predicción:

```
import numpy as np

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error



# Generar datos ficticios

np.random.seed(0)

X = np.random.rand(100, 1) * 10

y = 3 * X.squeeze() + np.random.randn(100) * 3  # Generar y = 3X +

ruido
```

```
# Dividir los datos en conjuntos de entrenamiento y prueba

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.2, random_state=42)



# Entrenar el modelo de Regresión Lineal

model = LinearRegression()

model.fit(X_train, y_train)
```

```
# Evaluar el modelo

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)

print("Error cuadrático medio:", mse)
```

Ver documentación oficial de sklearn.model_selection.train_test_split en:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Ver documentación oficial de sklearn.metrics.mean_squared_error en:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

Método de Mínimos Cuadrados:

```
# Definir la función para el método de mínimos cuadrados

def least_squares(X, y):

      X = np.c_[np.ones((X.shape[0], 1)), X]  # Agregar columna de
unos para el término de intercepción

      coefficients = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

      return coefficients

# Usar la función para ajustar un modelo

coefficients = least_squares(X, y)
```

Interpretación de la Pendiente y la Intersección:

```
# Extraer los coeficientes de la pendiente e intersección del modelo
de Regresión Lineal

intercept = model.intercept_

coefficients = model.coef_

print("Intersección:", intercept)

print("Coeficientes de la pendiente:", coefficients)
```

Interpretación de la Pendiente y la Intersección:

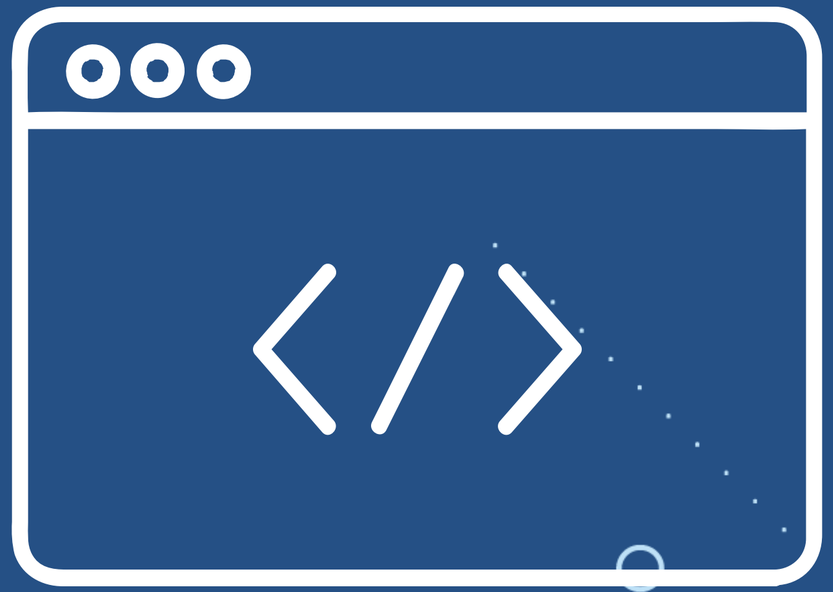
```
# Graficar la línea de regresión

import matplotlib.pyplot as plt

plt.scatter(X, y, color='blue')

plt.plot(X, model.predict(X), color='red')  # Línea de regresión

plt.xlabel('Características')

plt.ylabel('Variable Dependiente')

plt.title('Regresión Lineal')

plt.show()
```

3. Regresión Polinomial

La Regresión Polinomial es una extensión del modelo lineal que permite capturar relaciones no lineales entre las variables. En lugar de ajustarse a una línea recta, este método se ajusta a una curva polinómica que puede ser de grado mayor que 1. Esto se logra mediante el uso de términos polinomiales, donde cada término representa una potencia de la variable independiente. Por ejemplo, un modelo de regresión polinomial de grado 2 tendría términos como x2, x3, etc.

El uso de términos polinomiales permite ajustar mejor los datos cuando la relación entre las variables es más compleja que una simple línea recta. Esto puede ser útil en situaciones donde los datos muestran tendencias no lineales o patrones más intrincados.

Sin embargo, es importante tener en cuenta consideraciones sobre el sobreajuste y la selección del grado del polinomio.



El sobreajuste ocurre cuando el modelo se adapta demasiado a los datos de entrenamiento, capturando incluso el ruido aleatorio, lo que puede llevar a una mala generalización a nuevos datos. Para evitar esto, es crucial encontrar un equilibrio entre:

La flexibilidad del modelo (grado del polinomio)

Su capacidad para generalizar datos no vistos

Esto se logra mediante técnicas como la validación cruzada o la curva de validación para seleccionar el grado del polinomio óptimo que minimice el error de predicción en datos no vistos.

Regresión Polinomial:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error
```

```
# Generar datos ficticios

np.random.seed(0)

X = np.sort(10 * np.random.rand(100, 1), axis=0)

y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])

# Generar una función sinusoidal con ruido

# Dividir los datos en conjuntos de entrenamiento y prueba

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.2, random_state=42)
```

```
# Ajustar diferentes grados de polinomios y calcular el error cuadrático medio en el conjunto de prueba

degrees = [1, 2, 3, 5, 10]

mse_list = []
```

```
for degree in degrees:

    poly_features = PolynomialFeatures(degree=degree)

    X_train_poly = poly_features.fit_transform(X_train)

    X_test_poly = poly_features.transform(X_test)

    model = LinearRegression()

    model.fit(X_train_poly, y_train)

    y_pred = model.predict(X_test_poly)

    mse = mean_squared_error(y_test, y_pred)

    mse_list.append(mse)
```

```
# Graficar el error cuadrático medio en función del grado del
polinomio

plt.figure(figsize=(10, 6))

plt.plot(degrees, mse_list, marker='o')

plt.title('Error cuadrático medio vs Grado del polinomio')

plt.xlabel('Grado del polinomio')

plt.ylabel('Error cuadrático medio')

plt.xticks(degrees)

plt.grid(True)

plt.show()
```

4. Regresión Logarítmica

La Regresión Logarítmica es una técnica utilizada para modelar relaciones no lineales que exhiben un crecimiento o decrecimiento logarítmico en lugar de un crecimiento lineal. Este modelo es especialmente útil cuando los datos muestran un patrón de crecimiento o decrecimiento que no puede ser capturado adecuadamente por modelos lineales tradicionales. La aplicación de la regresión logarítmica implica ajustar una curva logarítmica a los datos, lo que permite una mejor representación de la relación entre las variables.

La interpretación de los coeficientes en el contexto logarítmico es crucial para comprender cómo afectan al crecimiento o decrecimiento de la variable dependiente. Por ejemplo:

Un coeficiente positivo indica un crecimiento logarítmico en la variable dependiente cuando la variable independiente aumenta.

Un coeficiente negativo indica un decrecimiento logarítmico.

Ejemplos de aplicación de la regresión logarítmica se encuentran en áreas como la economía y la biología. Veamos algunos casos puntuales:

En economía

Se puede utilizar para modelar el crecimiento de la población, el producto interno bruto (PIB) o el ingreso per cápita, ya que estos suelen seguir patrones de crecimiento logarítmico.

En biología

Se puede aplicar para modelar el crecimiento de poblaciones de organismos, la propagación de enfermedades o el decaimiento de sustancias radiactivas, donde las tasas de crecimiento o decrecimiento pueden ser no lineales y seguir un patrón logarítmico.

- # Regresión Logarítmica:

Generación de datos artificiales

```
import numpy as np

# Generar datos ficticios con una relación logarítmica creciente

np.random.seed(0)

X = np.linspace(1, 10, 100)

y = 2 * np.log(X) + np.random.normal(0, 0.2, 100)  # Función

logarítmica con ruido gaussiano
```

Ver documentación oficial de np.random.seed en:
https://numpy.org/doc/stable/reference/random/generated/numpy.random.seed.html

Aplicación de Regresión Logarítmica:

```
from scipy.optimize import curve_fit

def logarithmic_function(x, a, b):

      return a * np.log(x) + b

# Ajustar la curva logarítmica a los datos

params, _ = curve_fit(logarithmic_function, X, y)

a, b = params
```

Ver documentación oficial de scipy.optimize.curve_fit en:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

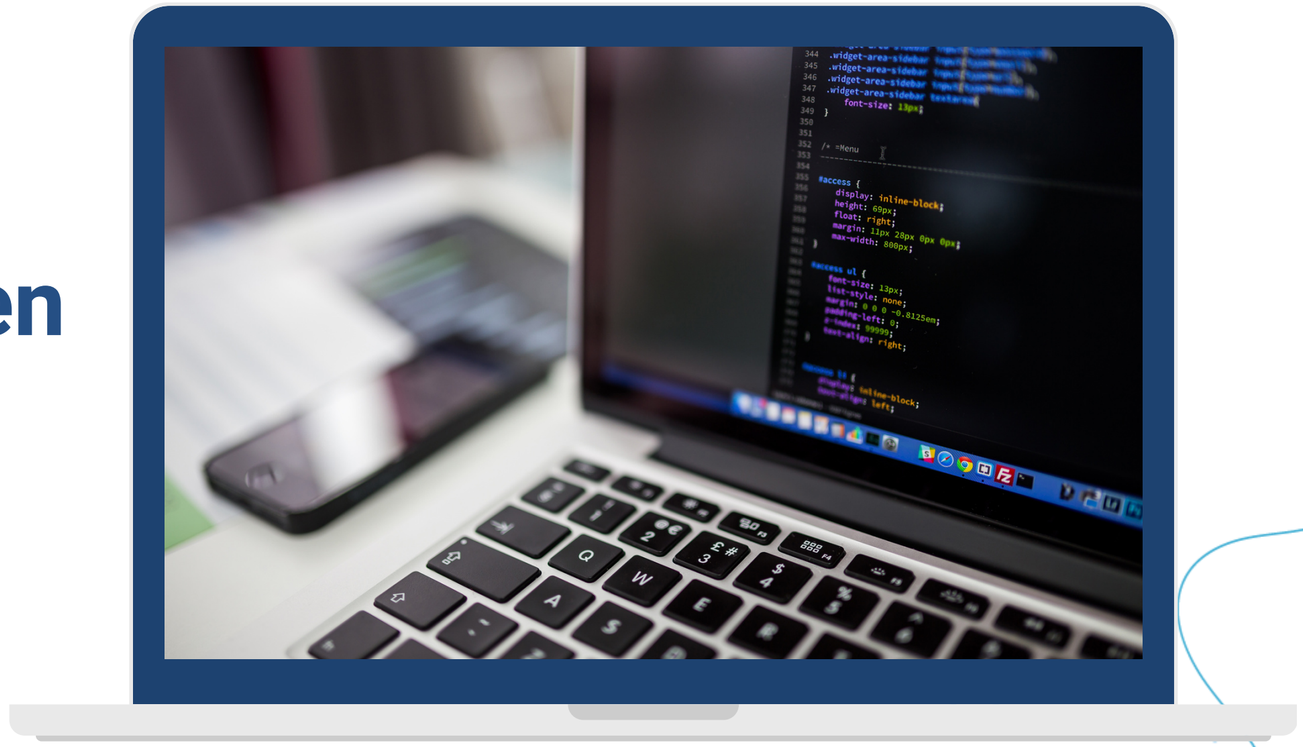
Interpretación de los coeficientes:

```
print("Coeficiente 'a':", a)

print("Coeficiente 'b':", b)

print("Interpretación:")

print("El coeficiente 'a' representa el crecimiento o decrecimiento
logarítmico de la variable dependiente por cada unidad de cambio en la
variable independiente.")

print("El coeficiente 'b' representa el intercepto de la curva
logarítmica.")
```

COLOMBIA POTENCIA DE LA VIDA

TIC

Ejemplos de aplicación en economía y biología



TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

Universidad Tecnológica de Pereira

Ejemplo de aplicación en economía:

```
# Generar datos ficticios para el crecimiento del PIB en función del
tiempo
X_economy = np.linspace(1, 10, 100)
y_economy = 3 * np.log(X_economy) + np.random.normal(0, 0.2, 100)  #
Función logarítmica con ruido gaussiano
# Aplicar el mismo procedimiento de ajuste de curva logarítmica
params_economy, _ = curve_fit(logarithmic_function, X_economy,
y_economy)
```

Ver documentación oficial de np.linspace en:
https://numpy.org/doc/stable/reference/generated/numpy.linspace.html

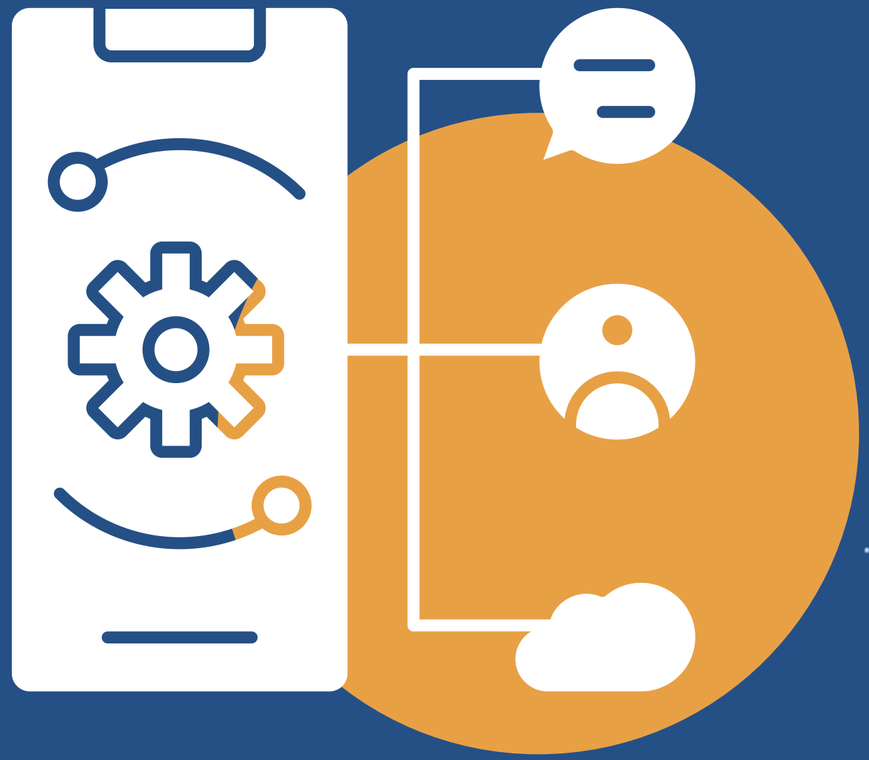
Ejemplo de aplicación en biología:

```
# Generar datos ficticios para el crecimiento de una población de
organismos en función del tiempo
X_biology = np.linspace(1, 10, 100)
y_biology = 2 * np.log(X_biology) + np.random.normal(0, 0.2, 100)  #
Función logarítmica con ruido gaussiano
# Aplicar el mismo procedimiento de ajuste de curva logarítmica
params_biology, _ = curve_fit(logarithmic_function, X_biology,
y_biology)
```

Ver documentación oficial de np.log en:
https://numpy.org/doc/stable/reference/generated/numpy.log.html

5. Regresión Logística

La Regresión Logística es un algoritmo de aprendizaje supervisado utilizado para la clasificación binaria, es decir, cuando el objetivo es predecir una variable categórica que puede tomar dos valores, como "sí" o "no", "positivo" o "negativo", "enfermo" o "sano", entre otros. A pesar de su nombre, la Regresión Logística se utiliza para problemas de clasificación, no de regresión.



- **Modelo Logístico para Clasificación Binaria:**

En la Regresión Logística, modelamos la probabilidad de que una observación pertenezca a una de las dos clases, utilizando una función logística. Dado un conjunto de características X, queremos predecir la probabilidad de que la variable dependiente yy pertenezca a una clase específica (por ejemplo, y=1). Formalmente, la función logística se define como:

$$p(y=1 \mid X)=11+e-\theta TX$$

Donde θθ son los parámetros del modelo que queremos aprender y X es el vector de características de la observación. Esta función toma como entrada el producto punto entre θ y X y produce una probabilidad entre 0 y 1. La función logística transforma la regresión lineal (θTX) a una escala de 0 a 1, lo que la hace adecuada para modelar probabilidades.

- **Función Logística y Transformación de la Regresión Lineal:**

La función logística (también conocida como sigmoide) es una curva en forma de 'S' que se utiliza para modelar la probabilidad de que una variable binaria dependiente esté presente en función de una o más variables independientes. Esta función tiene la forma:

$$g(z)=11+e-z$$

Donde $z=\theta TX$ es la combinación lineal de las características X y los parámetros del modelo θ. La función logística mapea cualquier valor de entrada a un valor en el rango de 0 a 1, lo que la hace adecuada para representar probabilidades.

- **Interpretación de las Probabilidades Predichas y la Frontera de Decisión:**

Una vez que hemos ajustado nuestros parámetros θ utilizando algún algoritmo de optimización (como la maximización de la verosimilitud), podemos utilizar el modelo entrenado para predecir la probabilidad de que una observación pertenezca a la clase positiva. Si la probabilidad predicha es **superior a un umbral** (generalmente 0.5), clasificamos la observación como perteneciente a la **clase positiva**; de lo contrario, la clasificamos como perteneciente a la **clase negativa**.

La frontera de decisión es el límite que separa las regiones donde se clasifican las observaciones como pertenecientes a una clase o a otra. En el caso de la Regresión Logística, la frontera de decisión es una función de las características y los parámetros del modelo que determina el límite entre las clases.

Implementación de Regresión Logística desde Cero:

```python
import numpy as np

class LogisticRegression:

    def __init__(self, learning_rate=0.01, num_iterations=1000):

        self.learning_rate = learning_rate

        self.num_iterations = num_iterations

        self.weights = None

        self.bias = None

    def sigmoid(self, z):

        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):

        m, n = X.shape

        self.weights = np.zeros(n)

        self.bias = 0
```

```
# Gradiente descendente
for _ in range(self.num_iterations):
    z = np.dot(X, self.weights) + self.bias
    y_pred = self.sigmoid(z)

    # Cálculo del gradiente
    dw = (1 / m) * np.dot(X.T, (y_pred - y))
    db = (1 / m) * np.sum(y_pred - y)

    # Actualización de parámetros
    self.weights -= self.learning_rate * dw
    self.bias -= self.learning_rate * db
```

```
def predict(self, X):
    z = np.dot(X, self.weights) + self.bias
    y_pred = self.sigmoid(z)
    return np.round(y_pred)
```

Prueba de la Implementación de Regresión Logística:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Generar datos de ejemplo
X, y = make_classification(n_samples=1000, n_features=5, n_classes=2,
random_state=42)


# Dividir datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Crear y entrenar modelo de regresión logística

model = LogisticRegression()

model.fit(X_train, y_train)


# Predecir en conjunto de prueba

y_pred = model.predict(X_test)


# Calcular precisión

accuracy = accuracy_score(y_test, y_pred)

print("Precisión del modelo de regresión logística:", accuracy)
```

Comparación con Sklearn:

```
from sklearn.linear_model import LogisticRegression as
SklearnLogisticRegression

# Crear y entrenar modelo de regresión logística de sklearn
sklearn_model = SklearnLogisticRegression()
sklearn_model.fit(X_train, y_train)

# Predecir en conjunto de prueba
sklearn_y_pred = sklearn_model.predict(X_test)

# Calcular precisión
sklearn_accuracy = accuracy_score(y_test, sklearn_y_pred)
print("Precisión del modelo de regresión logística de
sklearn:", sklearn_accuracy)
```