Logistic Regression From Scratch!

Model Building

In [24]:
```
import numpy as np
```

In [26]:
```
class LogisticRegression:

    # Initializes the Logistic Regression model with learning rate and
    def __init__(self, lr=0.1, n_iters=5000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    # Sigmoid function: Converts linear model output to probabilities
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    # Fits the Logistic Regression model to the training data
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        # Performs gradient descent for the specified number of iterat
        for _ in range(self.n_iters):
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = self.sigmoid(linear_model)

            # Computes gradients for weights and bias
            dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
            db = (1 / n_samples) * np.sum(y_pred - y)

            # Updates weights and bias using the gradients and learnin
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    # Predicts class labels (0 or 1) for input data
    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = self.sigmoid(linear_model)
        return [1 if i > 0.5 else 0 for i in y_pred]
```

Model Interpretation and Memo for

Logistic Regression (AssemblyAI, 2022)

Initializes the Logistic Regression model with learning rate and number of iterations

```
def __init__(self, lr=0.01, n_iters=5000):
    self.lr = lr
    self.n_iters = n_iters
    self.weights = None
    self.bias = None
```

The **init** method initializes the logistic regression model. It sets the learning rate (lr), which controls the step size during optimization, and the number of iterations (n_iters) for gradient descent. It also initializes weights and bias to None, which will later store the model parameters. The learning rate and iteration frequencies (n_iters) can be changed based on your research requirement. The lower the learning rate and higher the iteration, the more precise your model will be. However, the training process may slow down. For a small sample dataset like in the present study did not change the outcome at all.

sigmoid [1 / (1 + e^-value)](AssemblyAI, 2022)

```
# Sigmoid function: Converts linear model output to
probabilities between 0 and 1
def sigmoid(self, z):
    return 1 / (1 + np.exp(-z))
```

The sigmoid function is a mathematical function used in logistic regression to ensure the output falls between 0 and 1, representing a probability. It belongs to a group of functions known as logistic functions, which share this property (Brownlee, 2016). The sigmoid method applies the sigmoid activation function to a given input z. It maps any real number into a range between 0 and 1, making it useful for converting linear model outputs into probabilities (AssemblyAI, 2022).

fitting (AssemblyAI, 2022)

```
# Fits the Logistic Regression model to the training
data
def fit(self, X, y):
    n_samples, n_features = X.shape
    self.weights = np.zeros(n_features)
    self.bias = 0
```

Similarly to the Linear Regression model, this is where the weight and bias are initially set to zero for the gradient descent loop that starts at the next code block. This process is repeated until the optimal balance is maintained between the two. Detailed description of how the gradient descent process work can be found in my transcript under the Linear Regression model. Based the coefficient estimates here allows to move onto the prediction.

predicting (AssemblyAI, 2022)

```
# Predicts class labels (0 or 1) for input data
def predict(self, X):
    linear_model = np.dot(X, self.weights) + self.bias
    y_pred = self.sigmoid(linear_model)
    return [1 if i > 0.5 else 0 for i in y_pred]
```

This code defines the prediction set up in the "LogisticRegression" class and tells the model to use the X variable to predict. np.dot() function provides the dot function, where it takes into account the increase or decrease of Y value based on the change in X and add the bias term. The concept of balance between the weights and bias is well explained by James et al, (2023). Highly recommended to read through.

Logistic Model Prediction (70/20/10)

In [30]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classifi
```

In [39]:
```
# Load data
df = pd.read_csv("PISA2018_data.csv")

# Use one predictor variable (e.g., 'WEALTH')
X = df['WEALTH'].values.reshape(-1, 1)
y = df['LANGUAGE'].values  # Binary outcome variable ONLY!!

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Train logistic regression model
clf = LogisticRegression(lr=0.001, n_iters=5000)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)
```

```
# Manual accuracy
def accuracy(y_pred, y_test):
    return np.sum(y_pred == y_test) / len(y_test)

manual_acc = accuracy(y_pred, y_test)
print(f"Manual Accuracy: {manual_acc:.4f}")

# Using sklearn metrics
print(f"Sklearn Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Manual Accuracy: 0.6667
Sklearn Accuracy: 0.6667
```

Prediction Model Interpretation and Note

Use one predictor variable (e.g., 'WEALTH')

```
df = pd.read_csv("PISA2018_data.csv")
X = df['WEALTH'].values.reshape(-1, 1)
y = df['LANGUAGE'].values  # Binary outcome variable
ONLY!!
```

The dataset PISA2018_data.csv was loaded into a Pandas DataFrame, with 'WEALTH' as the predictor feature and 'LANGUAGE' as the binary outcome. 'WEALTH' was reshaped into a 2D (rows and columns) array as required by sci-kit-learn for input variables, which works just like transposing in the MS Excel. This solution was facilitated by the ChatGPT-4o model. I modified the original code by AssemblyAI (2022) to utilize scikit-learn metrics for accuracy comparison against a manual calculation, ensuring the correct equation.

Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.2, random_state=42)
```

Just like in the linear regression model, the data was split into training and test sets using an 80-20 ratio. X_train and y_train are used to train the model, while X_test and y_test are reserved for evaluating the model's performance. A fixed random_state = 42 works as though setting a seed in R, ensuring reproducibility in randomness.

Train Logistic Regression Model (Scikit-learn, 2022)

```
clf = LogisticRegression(lr=0.001, n_iters=5000)
clf.fit(X_tran, y_train)
```

A custom logistic regression model (`LogisticRegression`) was initialized with a learning rate of 0.001 and set to run for 5000 iterations. Trained on the dataset (X_train and y_train), the model learns the relationship between WEALTH and the binary outcome LANGUAGE. In contrast to a previous Random Forest model that failed due to a lack of my machine specifications, this smaller dataset indicates that variable adjustments do not significantly affect accuracy or processing speed. For larger datasets, modifying these parameters can optimize accuracy and processing speed according to your research needs.

Make Predictions

```
y_pred = clf.predict(X_test)
```

The trained logistic regression model predicts the outcomes for the test data (X_test). These predictions (y_pred) are later used to calculate accuracy.

Manual Accuracy

```
def accuracy(y_pred, y_test):
    return np.sum(y_pred == y_test) / len(y_test)

manual_acc = accuracy(y_pred, y_test)
print(f"Manual Accuracy: {manual_acc:.4f}")
```

A manual accuracy function is defined to compare predictions (y_pred) with actual test outcomes (y_test). I differentiated the outcome between the manual and the pre-existing function for an easy comparison. The accuracy is calculated as the proportion of correctly predicted values over the total number of observations (AssemblyAI, 2022).

Using Sklearn Metrics

```
print(f"Scikit-learn Accuracy: {accuracy_score(y_test,
y_pred):.4f}")
```

The accuracy_score function from scikit-learn metrics is used to validate the manual accuracy calculation. Both results are printed and compared to ensure consistency.

References:

AssemblyAI (Director). (2022). (2) How to implement Logistic Regression from scratch with Python—YouTube [YouTube]. https://www.youtube.com/watch?v=YYEJ_GUguHw

Scilit-learn. (2024). 1.1. Linear Models. Scikit-Learn (User Guide). https://scikit-learn/stable/modules/linear_model.html

```
In [ ]:
```

```
In [ ]:
```