A Mini Project Report

On

Heart Disease Prediction

In the partial fulfillment of the requirements for four years of Bachelors of Engineering in Electrical & Electronics Engineering

By: **Aavash Shrestha (41025)**

To: **Dr. Ram Kaji Budhathoki**

Professor, DoEEE

Department of Electrical & Electronics Engineering

School of Engineering, Kathmandu University

December, 2024

ABSTRACT

Heart disease remains a leading global health concern, emphasizing the need for early detection tools. This project implements a logistic regression model to predict heart disease using clinical and demographic features such as age, cholesterol, and blood pressure. Key steps included data preprocessing, feature scaling, and optimizing the model through gradient descent by minimizing cross-entropy loss. The impact of varying learning rates (0.1, 0.01, and 0.001) on training loss and accuracy was analyzed to ensure optimal performance.

This project highlights the potential of machine learning in supporting clinical decision-making and improving early diagnosis and prevention strategies.

LIST OF FIGURES

TABLE OF CONTENT

CHAPTER I: BACKGROUND AND INTRODUCTION

1.1. Introduction

Heart disease is a leading cause of morbidity and mortality worldwide, making its early detection and prediction critical for effective prevention and treatment. Logistic regression is a widely used statistical model for binary classification problems, such as predicting the presence or absence of heart disease. This report summarizes the implementation of a logistic regression model for predicting heart disease based on patient data and highlights its performance, methodology, and results.

1.2. Problem Definition

Heart disease is one of the most prevalent and life-threatening conditions globally. Early diagnosis of heart disease can significantly improve patient outcomes and reduce mortality rates. However, due to the complexity of clinical data and the subtleties in symptoms, accurately predicting heart disease remains a challenge. Developing a reliable predictive model can aid healthcare professionals in identifying high-risk individuals, enabling early interventions.

1.3. Objective

The objective of this project is to develop a predictive model using logistic regression to determine whether a patient is likely to have heart disease based on input features such as age, cholesterol levels, blood pressure, and other relevant health metrics. Additionally, the project evaluates the impact of different learning rates on model performance to optimize its training process.

1.4. Significance of the Project

With cardiovascular diseases being a leading cause of death globally, tools that can predict risks early are indispensable. Machine learning-based predictive models can assist healthcare professionals in identifying high-risk individuals, improving decision-making, and ultimately saving lives [4].

The ability to predict heart disease based on routine diagnostic data holds immense value:

- Improved Patient Outcomes: Early identification of high-risk patients allows for timely medical interventions.

- Resource Optimization: Hospitals and clinics can allocate resources more effectively to patients at higher risk.
- Public Health Impact: Reducing the overall burden of heart disease through targeted prevention and treatment strategies. This project is significant as it combines readily available clinical data and machine learning techniques to create an accessible and interpretable tool for heart disease prediction.

1.5. Proposed Solution

This project uses logistic regression, a widely used statistical model for binary classification, to predict the likelihood of heart disease. The solution involves:

- Collecting and preprocessing patient data.
- Training a logistic regression model on diagnostic features such as cholesterol levels, blood pressure, and age.
- Applying Gradient descent to minimize the cross-entropy loss, ensuring the model converges to an optimal solution [3].
- Validating the model's performance using accuracy metrics and visualizations of the training process. The goal is to build a predictive model that is both accurate and interpretable for use in real-world healthcare settings.

CHAPTER II: PROJECT METHODOLOGY

2.1. System Overview

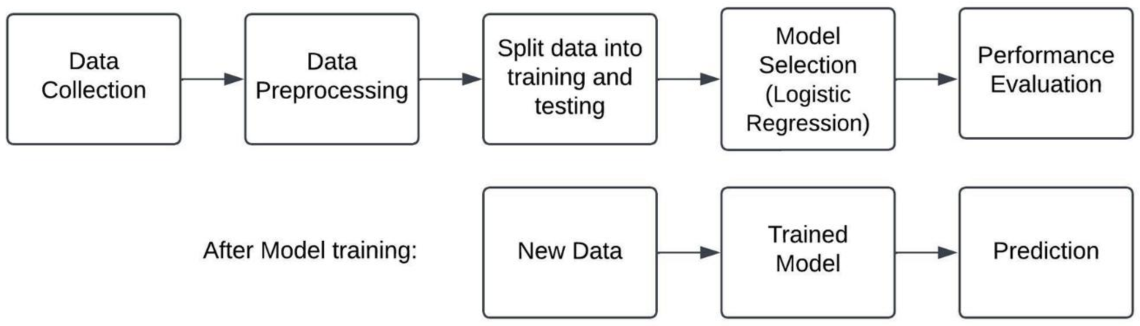


Figure 1 System Workflow

1. **Before Model Training:**
   - Data Collection: The dataset is imported.
   - Data Preprocessing: Features are normalized to prepare them for training.
   - Data Splitting: The dataset is divided into training and testing subsets.
2. **During Model Training:**
   - Logistic Regression: The model is trained using the training dataset to optimize weights and biases.
   - Performance Monitoring: Loss and accuracy are tracked and visualized to monitor the model's learning progress.
3. **After Model Training:**
   - The trained model is tested on new data to make predictions about the likelihood of heart disease.

2.2 Dataset Description

The data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease. [1]

Attributes:

- age
- sex

- chest pain type (4 values)
- resting blood pressure
- serum cholesterol in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by fluoroscopy
- thal: 0 = normal; 1 = fixed defect

2.3 Methodology

The process involved the following steps:

1. **Data Preprocessing:**
   - Splitting the dataset into training and testing sets (80% training, 20% testing).
   - Scaling the features manually to normalize the data, ensuring numerical stability and better convergence of the model. [2]
2. **Logistic Regression Implementation:**
   - The sigmoid function was used to map the model's output to probabilities between 0 and 1. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

   - Cross-entropy loss was used as the cost function.
   - Gradients of the weights and bias were calculated to update the model parameters using gradient descent.

$$Loss = -\frac{1}{m}\sum_{i=1}^{m}(y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i}))$$

Where:

- M = number of training samples
- y_i = true labeled (0 or 1) for the ith sample
- $\hat{y_i} = sigmoid(z_i) = predicted\ probability\ for\ the\ i^{th}\ sample$

- z_i = linear combination of weights, input features and bias

3. **Hyperparameters:**
   - Learning Rate: 0.1, 0.01, 0.01
   - Epochs: 10,000 iterations
4. **Performance Evaluation:**
   - Training and test accuracy were calculated to evaluate the model's effectiveness.
   - The training loss and accuracy were tracked over epochs to visualize model convergence.
5. **Visualization:**
   - Graphs of training loss and training accuracy over epochs were plotted to ensure proper learning behavior.

CHAPTER III: RESULTS AND CONCLUSION

3.1. Result

1. **Scaling:** The manual scaling of data ensured that all features were normalized to have a mean of 0 and a standard deviation of 1, avoiding issues with gradient calculations during training.
2. **Model Training:**
   - The model's weights and bias were optimized using gradient descent.
   - The training loss consistently decreased over epochs, indicating proper learning.
   - The training accuracy steadily increased, suggesting that the model captured patterns in the training data.
3. **Model Performance:** (for different values of learning rate)
   - Training Accuracy: ~80-85%
   - Test Accuracy: ~60-65%

These results demonstrate that the model generalizes reasonably well, although there may be **room for improvement.**

4. **Prediction:** For a new patient input, the model predicted whether the individual had heart disease.

3.2 Challenges and Solution

1. **Numerical Instabilities in Loss Calculation:**
   - The loss calculation resulted in inf due to extreme values passed to the logarithmic function.
   - Solution: Applied clipping to the sigmoid outputs to ensure numerical stability:

$$y_{predicted} = max(\epsilon, min(1-\epsilon, y_{predicted}))$$

   where $\epsilon = 1 \times 10^{-10}$

2. **Feature Scaling:** Scaling was implemented manually to avoid reliance on external libraries, improving interpretability of the process:

$$X_{mean} = \frac{X - mean(X)}{std(X)}$$

3.3 Analysis for different values hyperparameters

3.3.1 Varying the Learning Rate

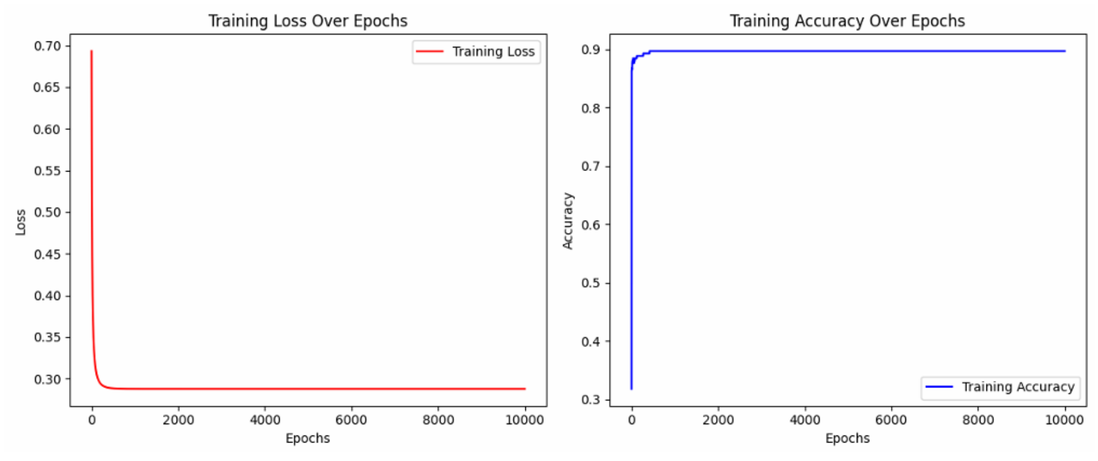


Figure 2 Training loss and accuracy (learning rate=0.1)

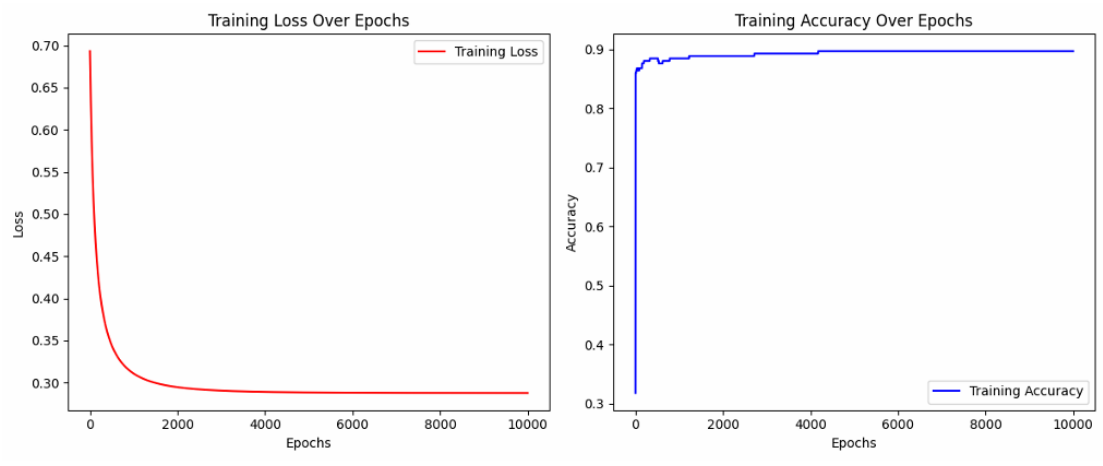


Figure 3 Training loss and accuracy (learning rate=0.01)

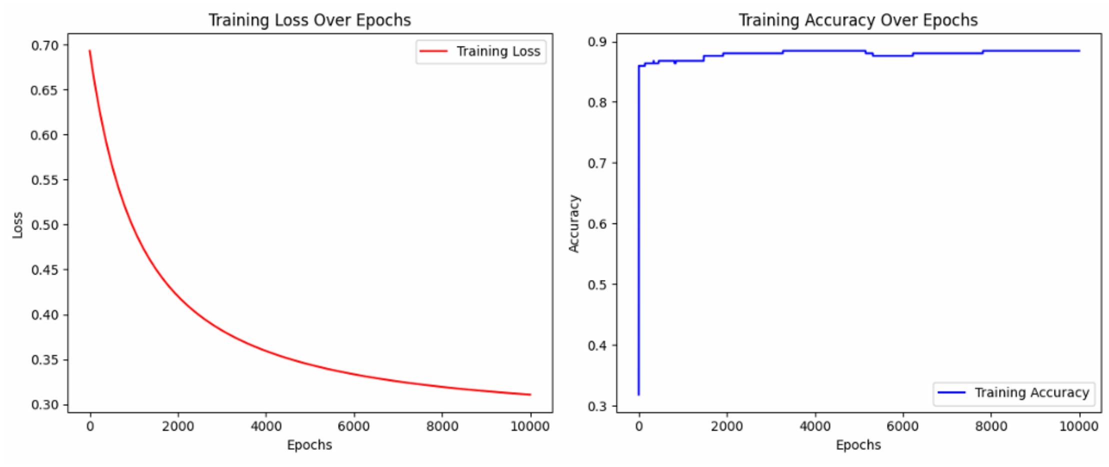


Figure 4 Training loss and accuracy (learning rate=0.001)

1. **Training Loss**

a) **Learning Rate 0.001:**

- The loss decreases very slowly and steadily over the epochs.
- Even at 10,000 epochs, the loss continues to decrease and has not fully plateaued, indicating very slow convergence.

b) **Learning Rate 0.01:**

- The loss decreases faster than 0.001 and stabilizes around 0.3 within 10,000 epochs.

c) **Learning Rate 0.1:**

- The loss decreases very rapidly and stabilizes much earlier compared to both 0.01 and 0.001.

Comparison:

- 0.001 has the slowest convergence, requiring significantly more training time to reach similar loss values as 0.1 and 0.01.
- 0.1 converges the fastest, making it the most efficient for this scenario, assuming stability is maintained.

2. **Training Accuracy**

a) **Learning Rate 0.001:**

- Accuracy increases steadily but much more slowly than with the higher learning rates.
- Stabilization around 90% occurs closer to the end of the training period.

b) **Learning Rate 0.01:**

- Accuracy improves faster than 0.001 and stabilizes earlier, closer to 90%.

c) **Learning Rate 0.1:**

- Accuracy reaches 90% the fastest, stabilizing within the first few thousand epochs.

Comparison:

- 0.001 is much slower to achieve high accuracy, making it less practical for tasks where time is a constraint.
- 0.1 achieves high accuracy quickly, demonstrating faster optimization compared to both 0.01 and 0.001.

General Observations

1) **Learning Rate 0.001:**
   - The model trains very slowly, making it less efficient, but gradual learning may help avoid overshooting or instability.
   - Useful for highly sensitive datasets or models prone to instability.
2) **Learning Rate 0.01:**
   - A balanced choice, achieving faster convergence than 0.001 without the risk of overshooting associated with higher rates.
3) **Learning Rate 0.1:**
   - Most efficient for this dataset and model, with the fastest convergence in both loss and accuracy.
   - Could risk overshooting in other cases, but the provided graphs show no such issue.

4.3.2 Varying the number of epochs

For different numbers of epochs, the graphs didn't show any major changes for the same value of learning rate which indicates that the model reaches its optimal performance well before the maximum number of epochs is reached. Some potential reasons for this behavior are:

1) Model Converges Early
   - The learning process stabilizes early, and the model's weights reach their optimal values before the later epochs are utilized.
   - For example, if the model converges within the first 2,000 epochs, extending the training to 10,000 epochs won't make a difference in the graph.
2) Dataset Simplicity
   - If the dataset is relatively simple or small, the model might learn everything it needs to early in training.
   - In such cases, extending the number of epochs would not affect the performance graphs because the model already achieves optimal performance.

3.4 Conclusion

The logistic regression model provided a reliable and interpretable approach for predicting heart disease. While the model achieved reasonable accuracy, there is potential for further improvement using advanced techniques or additional features. Nevertheless, this project demonstrates the feasibility of using machine learning for healthcare applications.

REFERENCES

[1] Dua, D., & Graff, C. (2019). UCI Machine Learning Repository: Heart Disease Dataset. University of California, Irvine, School of Information and Computer Sciences. Available at: https://archive.ics.uci.edu/ml/datasets/Heart+Disease

[2] Hastie, T., Tibshirani, R., & Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2009.

[3] Goodfellow, I., Bengio, Y., & Courville, A., *Deep Learning*. MIT Press, 2016.

[4] American Heart Association, Understanding Heart Disease Risk Factors. Available at: https://www.heart.org/, 2021.

APPENDIX

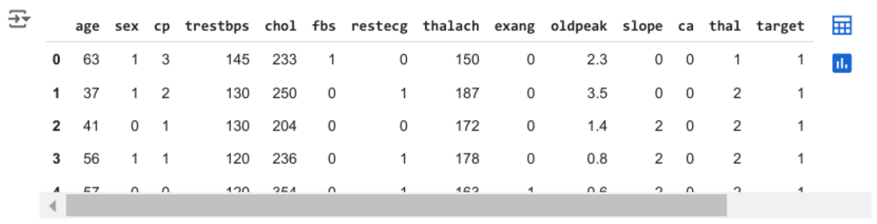
Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Data Collection and Processing

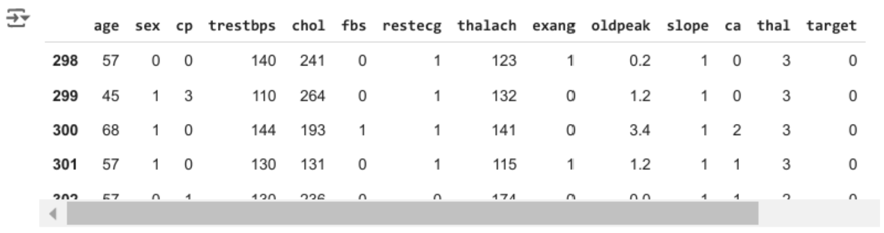
```
# loading the csv data to a Pandas DataFrame
heart_data = pd.read_csv('/content/heart_disease_data.csv')
```

```
# print first 5 rows of the dataset
heart_data.head()
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
# print last 5 rows of the dataset
heart_data.tail()
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
# number of rows and columns in the dataset
heart_data.shape
```

```
(303, 14)
```

```
# getting some info about the data
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
# statistical measures about the data
heart_data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

```
# checking the distribution of Target Variable
heart_data['target'].value_counts()
```

	count
target	
1	165
0	138

1 --> Defective Heart

0 --> Healthy Heart

Splitting the Features and Target

```
X = heart_data.drop(columns='target', axis=1)
Y = heart_data['target']
```

```
print(X)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3   
1     37    1   2       130   250    0        1      187      0      3.5   
2     41    0   1       130   204    0        0      172      0      1.4   
3     56    1   1       120   236    0        1      178      0      0.8   
4     57    0   0       120   354    0        1      163      1      0.6   
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...   
298   57    0   0       140   241    0        1      123      1      0.2   
299   45    1   3       110   264    0        1      132      0      1.2   
300   68    1   0       144   193    1        1      141      0      3.4   
301   57    1   0       130   131    0        1      115      1      1.2   
302   57    0   1       130   236    0        0      174      0      0.0   

     slope  ca  thal  
0        0   0     1  
1        0   0     2  
2        2   0     2  
3        2   0     2  
4        2   0     2  
..     ...  ..   ...  
298      1   0     3  
299      1   0     3  
300      1   2     3  
301      1   1     3  
302      1   1     2  

[303 rows x 13 columns]
```

```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

Splitting the Data into Training and Test

```
split_index = int(0.8 * len(X))
X_train, X_test = X[:split_index], X[split_index:]
Y_train, Y_test = Y[:split_index], Y[split_index:]
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(303, 13) (242, 13) (61, 13)
```

Sigmoid Function

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Logistic Regression Model

```
def logistic_regression(X, Y, lr, epochs):
    m, n = X.shape
    weights = np.zeros(n)
    bias = 0
    loss_history = []
    accuracy_history = []

    for epoch in range(epochs):
        # Linear function
        linear_model = np.dot(X, weights) + bias

        # Apply sigmoid
        y_predicted = sigmoid(linear_model)

        # Clip predictions to avoid log(0)
        y_predicted = np.clip(y_predicted, 1e-10, 1 - 1e-10)

        # Calculate gradients
        dw = (1 / m) * np.dot(X.T, (y_predicted - Y))
        db = (1 / m) * np.sum(y_predicted - Y)

        # Update weights and bias
        weights -= lr * dw
        bias -= lr * db

        # Calculate and store loss
        loss = -np.mean(Y * np.log(y_predicted) + (1 - Y) * np.log(1 - y_predicted))
        loss_history.append(loss)

        # Calculate and store accuracy
        y_pred_classes = [1 if i > 0.5 else 0 for i in y_predicted]
        acc = accuracy(Y, y_pred_classes)
        accuracy_history.append(acc)

        # Optional: Print loss to monitor training
        if epoch % 1000 == 0:
            print(f"Epoch {epoch}, Loss: {loss}, Accuracy: {acc}")

    return weights, bias, loss_history, accuracy_history
```

Prediction Function

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

Accuracy calculation

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

feature scaling

```python
def manual_scaling(X_train, X_test):
    # Calculate mean and standard deviation for each feature in training data
    mean = np.mean(X_train, axis=0)
    std = np.std(X_train, axis=0)

    # Avoid division by zero
    std[std == 0] = 1e-8

    # Scale training and testing data
    X_train_scaled = (X_train - mean) / std
    X_test_scaled = (X_test - mean) / std
    return X_train_scaled, X_test_scaled


# Scale the data manually
X_train, X_test = manual_scaling(X_train, X_test)
```

Training the model

```python
learning_rate = 0.1
epochs = 10000
weights, bias, loss_history, accuracy_history = logistic_regression(X_train, Y_train, learning_rate, epochs)
```

```
Epoch 0, Loss: 0.6931471805599453, Accuracy: 0.3181818181818182
Epoch 1000, Loss: 0.2877050413147831, Accuracy: 0.8966942148760331
Epoch 2000, Loss: 0.2876752867695793, Accuracy: 0.8966942148760331
Epoch 3000, Loss: 0.28767515515156383, Accuracy: 0.8966942148760331
Epoch 4000, Loss: 0.287675154492371, Accuracy: 0.8966942148760331
Epoch 5000, Loss: 0.287675154489018, Accuracy: 0.8966942148760331
Epoch 6000, Loss: 0.2876751544890007, Accuracy: 0.8966942148760331
Epoch 7000, Loss: 0.2876751544890007, Accuracy: 0.8966942148760331
Epoch 8000, Loss: 0.2876751544890007, Accuracy: 0.8966942148760331
Epoch 9000, Loss: 0.2876751544890007, Accuracy: 0.8966942148760331
```

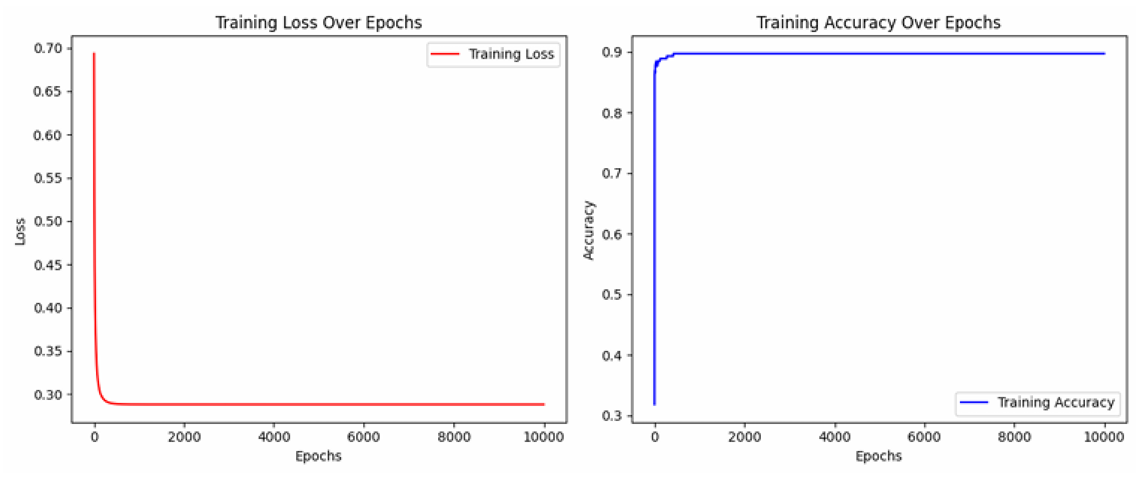
Graphical Representation

```python
# Plotting loss and accuracy
plt.figure(figsize=(12, 5))

# Loss plot
plt.subplot(1, 2, 1)
plt.plot(range(len(loss_history)), loss_history, label="Training Loss", color='red')  # Use len(loss_history) for the x-axis
plt.title("Training Loss Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(range(len(accuracy_history)), accuracy_history, label="Training Accuracy", color='blue')  # Use len(accuracy_history) for the
plt.title("Training Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```



Evaluate the model

```
Y_train_pred = predict(X_train, weights, bias)
train_acc = accuracy(Y_train, Y_train_pred)

Y_test_pred = predict(X_test, weights, bias)
test_acc = accuracy(Y_test, Y_test_pred)


print(f"Training Accuracy: {train_acc}")
print(f"Test Accuracy: {test_acc}")
```

```
Training Accuracy: 0.8966942148760331
Test Accuracy: 0.6065573770491803
```

Test with new input

```
input_data = np.array([62, 0, 0, 140, 268, 0, 0, 160, 0, 3.6, 0, 2, 2])
input_data_reshaped = input_data.reshape(1, -1)
input_prediction = predict(input_data_reshaped, weights, bias)

if input_prediction[0] == 0:
    print("The Person does not have Heart Disease")
else:
    [illegible]
```