Threshold 10^-3

In [50]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import time

def sigmoid(z):
    """Sigmoid function."""
    return 1 / (1 + np.exp(-z))

def logistic_regression_gradient_descent(X, y, learning_rate=1e-5, max_iterations=1
    """logistic regression model using gradient descent."""
    # Initialize weights with random values
    np.random.seed(0)
    num_features = X.shape[1]
    w = np.random.randn(num_features + 1)  # Include intercept term

    # Add intercept term to features
    X = np.hstack((np.ones((X.shape[0], 1)), X))

    # Initialize variables for error metrics
    errors = []

    # Perform gradient descent
    start_time = time.time()
    for i in range(max_iterations):
        # Compute predictions
        z = np.dot(X, w)
        y_pred = sigmoid(z)

        # Compute gradients
        gradient = np.dot(X.T, y - y_pred)

        # Update weights
        w += learning_rate * gradient

        # Compute cross-entropy error
        cross_entropy_error = -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_
        errors.append(cross_entropy_error)

        # Check gradient magnitude for termination
        if np.all(np.abs(gradient) < gradient_threshold):
            break

    # Compute classification error on training set
    y_pred_binary = np.round(y_pred)
    classification_error = np.mean(y_pred_binary != y)

    end_time = time.time()
    training_time = end_time - start_time
```

```
    return w, errors[-1], classification_error, training_time


# Load the Cleveland dataset
url_train = "cleveland-train.csv"
url_test = "cleveland-test.csv"

# Load the training dataset
df_train = pd.read_csv(url_train)
X_train = df_train.iloc[:, :-1].values
y_train = df_train.iloc[:, -1].values

# Load the test dataset
df_test = pd.read_csv(url_test)
X_test = df_test.iloc[:, :-1].values
y_test = df_test.iloc[:, -1].values

# Run experiments
for max_iterations in [10000, 100000, 1000000]:
    # Train logistic regression model using gradient descent
    w, cross_entropy_error, classification_error, training_time = logistic_regressi

    # Evaluate model on test set
    X_test_with_intercept = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
    z_test = np.dot(X_test_with_intercept, w)
    y_pred_test = sigmoid(z_test)
    y_pred_test_binary = np.round(y_pred_test)
    test_classification_error = np.mean(y_pred_test_binary != y_test)

    # Print results
    print(f"Cross-entropy error on training set: {cross_entropy_error}")
    print(f"Classification error on training set: {classification_error}")
    print(f"Classification error on test set: {test_classification_error}")
    print(f"Training time: {training_time} seconds")
```

```
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:37: Runt
imeWarning: divide by zero encountered in log
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:37: Runt
imeWarning: invalid value encountered in multiply
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:8: Runti
meWarning: overflow encountered in exp

Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.1633641719818115 seconds
Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.1137442588806152 seconds
Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.08378267288208 seconds
```

Threshold 10^-6

In [51]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import time

def sigmoid(z):
    """Sigmoid function."""
    return 1 / (1 + np.exp(-z))

def logistic_regression_gradient_descent(X, y, learning_rate=1e-5, max_iterations=1
    """logistic regression model using gradient descent."""
    # Initialize weights with random values
    np.random.seed(0)
    num_features = X.shape[1]
    w = np.random.randn(num_features + 1)  # Include intercept term

    # Add intercept term to features
    X = np.hstack((np.ones((X.shape[0], 1)), X))

    # Initialize variables for error metrics
    errors = []

    # Perform gradient descent
    start_time = time.time()
    for i in range(max_iterations):
        # Compute predictions
        z = np.dot(X, w)
        y_pred = sigmoid(z)

        # Compute gradients
        gradient = np.dot(X.T, y - y_pred)

        # Update weights
        w += learning_rate * gradient

        # Compute cross-entropy error
        cross_entropy_error = -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_
        errors.append(cross_entropy_error)

        # Check gradient magnitude for termination
        if np.all(np.abs(gradient) < gradient_threshold):
            break

    # Compute classification error on training set
    y_pred_binary = np.round(y_pred)
    classification_error = np.mean(y_pred_binary != y)

    end_time = time.time()
    training_time = end_time - start_time

    return w, errors[-1], classification_error, training_time
```

```
# Load the Cleveland dataset
url_train = "cleveland-train.csv"
url_test = "cleveland-test.csv"

# Load the training dataset
df_train = pd.read_csv(url_train)
X_train = df_train.iloc[:, :-1].values
y_train = df_train.iloc[:, -1].values

# Load the test dataset
df_test = pd.read_csv(url_test)
X_test = df_test.iloc[:, :-1].values
y_test = df_test.iloc[:, -1].values

# Run experiments
for max_iterations in [10000, 100000, 1000000]:
    # Train logistic regression model using gradient descent
    w, cross_entropy_error, classification_error, training_time = logistic_regressi

    # Evaluate model on test set
    X_test_with_intercept = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
    z_test = np.dot(X_test_with_intercept, w)
    y_pred_test = sigmoid(z_test)
    y_pred_test_binary = np.round(y_pred_test)
    test_classification_error = np.mean(y_pred_test_binary != y_test)

    # Print results
    print(f"Cross-entropy error on training set: {cross_entropy_error}")
    print(f"Classification error on training set: {classification_error}")
    print(f"Classification error on test set: {test_classification_error}")
    print(f"Training time: {training_time} seconds")
```

```
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:37: Runt
imeWarning: divide by zero encountered in log
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:37: Runt
imeWarning: invalid value encountered in multiply
C:\Users\johns\anaconda3\envs\cs688\lib\site-packages\ipykernel_launcher.py:8: Runti
meWarning: overflow encountered in exp

Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.178243637084961 seconds
Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.232536792755127 seconds
Cross-entropy error on training set: nan
Classification error on training set: 1.0
Classification error on test set: 1.0
Training time: 1.0653712749481201 seconds
```

In [ ]: