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

class LogisticRegressionMiniBatch:
    def __init__(self, lr=0.01, epochs=1000, batch_size=32, tol=1e-4):
        self.lr = lr  # Learning rate
        self.epochs = epochs  # Number of epochs
        self.batch_size = batch_size  # Mini-batch size
        self.tol = tol  # Tolerance for convergence
        self.weights = None
        self.bias = None

    # Sigmoid function for logistic regression
    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    # Predict probabilities for input X
    def predict_proba(self, X):
        z = np.dot(X, self.weights) + self.bias
        return self.sigmoid(z)

    # Predict class labels (0 or 1) based on threshold of 0.5
    def predict(self, X):
        probabilities = self.predict_proba(X)
        return (probabilities >= 0.5).astype(int)

    # Log-Loss (cross-entropy loss) function
    def log_loss(self, y_true, y_pred_proba):
        return -np.mean(y_true * np.log(y_pred_proba + 1e-15) + (1 - y_true) * np.l

    # Compute the gradients for weights and bias based on the mini-batch
    def gradient(self, X_batch, y_batch):
        y_pred_proba = self.predict_proba(X_batch)
        error = y_pred_proba - y_batch
        grad_w = np.dot(X_batch.T, error) / X_batch.shape[0]
        grad_b = np.mean(error)
        return grad_w, grad_b

    # Fit the model using mini-batch gradient descent
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.random.randn(n_features)
        self.bias = np.random.randn()

        for epoch in range(self.epochs):
            # Shuffle the data before each epoch
            indices = np.random.permutation(n_samples)
            X_shuffled = X[indices]
            y_shuffled = y[indices]

            # Iterate over mini-batches
            for i in range(0, n_samples, self.batch_size):
                X_batch = X_shuffled[i:i + self.batch_size]
                y_batch = y_shuffled[i:i + self.batch_size]
```

```
                # Calculate gradients
                grad_w, grad_b = self.gradient(X_batch, y_batch)

                # Update weights and bias
                self.weights -= self.lr * grad_w
                self.bias -= self.lr * grad_b

            # Print log loss every 100 epochs
            if epoch % 100 == 0:
                y_pred_proba = self.predict_proba(X)
                loss = self.log_loss(y, y_pred_proba)
                print(f"Epoch {epoch}: Loss {loss}")

            # Check for convergence
            if np.linalg.norm(grad_w) < self.tol:
                print("Convergence reached.")
                break

        return self.weights, self.bias


# Example usage

# Generate synthetic dataset for logistic regression
np.random.seed(42)
X = np.random.randn(100, 5)  # 100 samples, 5 features
true_weights = np.array([1, -2, 3, -1, 2])  # True weights
true_bias = -0.5  # True bias
linear_combination = np.dot(X, true_weights) + true_bias
y = (linear_combination > 0).astype(int)  # Generate binary labels (0 or 1)

# Create an instance of the Logistic Regression model with Mini-batch gradient desc
model = LogisticRegressionMiniBatch(lr=0.01, epochs=1000, batch_size=32, tol=1e-3)

# Fit the model
weights, bias = model.fit(X, y)

# Predict using the model
y_pred = model.predict(X)

# Print results
print("Learned Weights:", weights)
print("Learned Bias:", bias)
print("Predicted Labels:", y_pred)
```

```
Epoch 0: Loss 1.7954775897116972
Epoch 100: Loss 0.6983430514202746
Epoch 200: Loss 0.4011096161255388
Epoch 300: Loss 0.30419907403400676
Epoch 400: Loss 0.2563243041069259
Epoch 500: Loss 0.22783314077525027
Epoch 600: Loss 0.2081548602924787
Epoch 700: Loss 0.19283362361236306
Epoch 800: Loss 0.1808208923658152
Epoch 900: Loss 0.1709733971167161
Learned Weights: [ 1.03186409 -1.60923512  2.14758115 -0.37395479  1.70440633]
Learned Bias: -0.9762304535696413
Predicted Labels: [1 0 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1
1 0 1 0
 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0
 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
```

In [ ]:

```
# The Logistic Regression model utilizes the sigmoid function to predict the probab
# of class 1 given the input features X. The logistic loss function (log-loss) is u
# to evaluate how accurately the model predicts the class labels.
#
# During training, Mini-batch Gradient Descent is employed: the training data is sh
# before each epoch to ensure randomness, and the dataset is divided into mini-batc
# size `batch_size`. For each mini-batch, the gradients of the loss function with r
# to the weights w and bias b are computed, and the parameters are updated using th
# rate. This update process moves the weights and bias in the direction that minimi
# log-loss.
#
# The algorithm checks for convergence by monitoring if the gradients for weights w
# small enough (below a set tolerance), which stops the training early if this cond
# is met.
#
# In this implementation, a synthetic dataset is randomly generated, with known tru
# and bias used to produce the labels y. The model is then trained on this data, an
# weights and bias are printed, along with the predicted class labels for the input
# During the training process, the log-loss is also printed every 100 epochs, demon
# Mini-batch Gradient Descent can be utilized to train a logistic regression model,
# parameters based on small random subsets of the data (mini-batches).
```