```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 initialize_parameters(self, num_features):
        self.weights = np.zeros(num_features)
        self.bias = 0

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.initialize_parameters(num_features)
        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            predictions = self.sigmoid(linear_model)
            # Gradient descent updates
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        predictions = self.sigmoid(linear_model)
        return (predictions > 0.5).astype(int)
# Example usage

X_train = np.array([[2.5, 3.5], [1.5, 2.5], [3.5, 4.5], [2.0, 2.5]])
y_train = np.array([1, 0, 1, 0])

model = LogisticRegression(learning_rate=0.01, num_iterations=1000)
model.fit(X_train, y_train)

X_test = np.array([[2.0, 3.0], [1.0, 1.5]])

predictions = model.predict(X_test)
print("Predictions:", predictions)
```

```
Predictions: [1 0]
```

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
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 relu(self, z):
        return np.maximum(0, z)

    def initialize_parameters(self, num_features):
        self.weights = np.zeros(num_features)
        self.bias = 0

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.initialize_parameters(num_features)
        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            predictions = self.relu(linear_model)

            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        predictions = self.relu(linear_model)
        return (predictions > 0).astype(int)

iris = datasets.load_iris()
X = iris.data
y = iris.target

y = (y == 0).astype(int)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression(learning_rate=0.01, num_iterations=1000)
model.fit(X_train, y_train)

predictions = model.predict(X_test)
```

```
accuracy = np.mean(predictions == y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9777777777777777
```