MACHINE LEARNING

LAB WORK 5

Name: Arjun Unnikrishnan USN: 22BTRAD004

Logistic Regression Algorithm

Logistic Regression is a statistical method commonly used for binary classification problems, where the goal is to predict the probability of an instance belonging to one of two classes. Despite its name, logistic regression is a classification algorithm, not a regression one. The algorithm models the relationship between a dependent binary variable and one or more independent variables by estimating probabilities using the logistic function. The logistic function, also known as the sigmoid function, maps any real-valued number into a range of 0 to 1, making it suitable for representing probabilities. Logistic Regression works by fitting a linear equation to the input features and then applying the logistic function to the result. The model is trained using optimization techniques like gradient descent to adjust the parameters (weights and bias) iteratively, minimizing the difference between predicted probabilities and actual class labels in the training data. Logistic Regression is widely employed in various fields, including medicine, finance, and machine learning, due to its simplicity, interpretability, and effectiveness in binary classification tasks.

Code:

```
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 fit(self, X, y):

        # Initialize weights and bias

        num_samples, num_features = X.shape

        self.weights = np.zeros(num_features)

        self.bias = 0

        # Gradient Descent

        for _ in range(self.num_iterations):

            # Calculate predictions

            linear_model = np.dot(X, self.weights) + self.bias

            predictions = self.sigmoid(linear_model)

            # Compute gradients
```

```
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)
            # Update weights and bias
            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 [1 if x >= 0.5 else 0 for x in predictions]
# Example usage:
# Assuming X_train and y_train are your training data
X_train = np.array([[2, 3], [1, 2], [3, 4]])
y_train = np.array([0, 0, 1])
# Create and train the logistic regression model
model = LogisticRegression(learning_rate=0.01, num_iterations=1000)
model.fit(X_train, y_train)
# Assuming X_test is your test data
X_test = np.array([[4, 5], [1, 1]])
# Make predictions
predictions = model.predict(X_test)
print("Predictions:", predictions)
```

Output:

Name: Arjun Unnikrishnan

USN: 22BTRAD004

Lab 5

Implementing Logical Regression Algorithm

In [1]:
```
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 fit(self, X, y):
        # Initialize weights and bias
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0
        # Gradient Descent
        for _ in range(self.num_iterations):
            # Calculate predictions
            linear_model = np.dot(X, self.weights) + self.bias
            predictions = self.sigmoid(linear_model)
            # Compute gradients
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)
            # Update weights and bias
            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 [1 if x >= 0.5 else 0 for x in predictions]
# Example usage:
# Assuming X_train and y_train are your training data
X_train = np.array([[2, 3], [1, 2], [3, 4]])
y_train = np.array([0, 0, 1])
# Create and train the logistic regression model
model = LogisticRegression(learning_rate=0.01, num_iterations=1000)
model.fit(X_train, y_train)
# Assuming X_test is your test data
X_test = np.array([[4, 5], [1, 1]])
# Make predictions
predictions = model.predict(X_test)
print("Predictions:", predictions)
```

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

GitHub Link: https://github.com/arj1-1n/ML