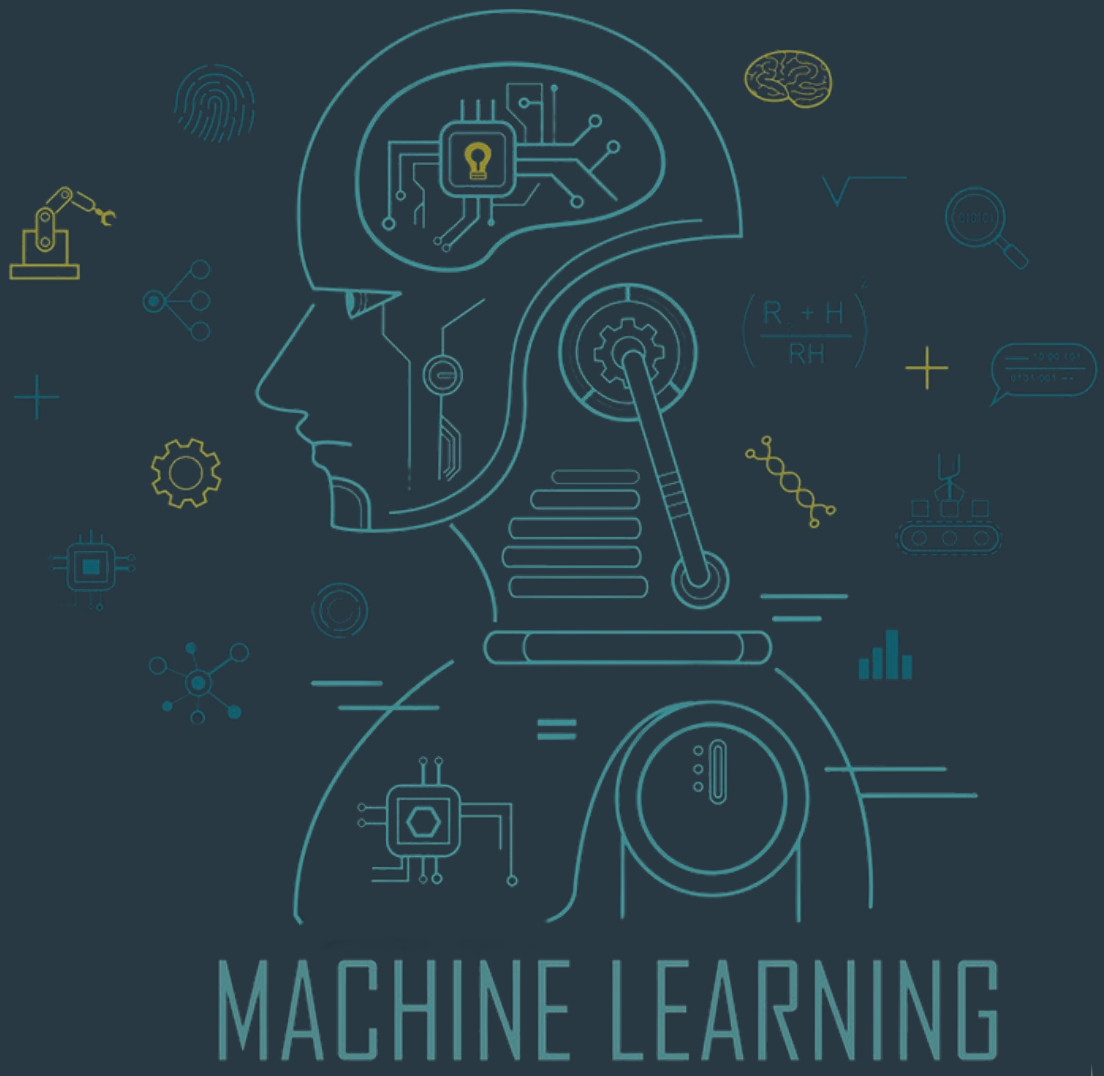
10. 11 . 2024

Project Report



Submitted by: **EFE EROL**
No: **200101066**

1. Introduction

The goal of this implementation is to use gradient descent to find the optimal values of θ that minimize the cost function.

The implementation follows standard practices for logistic regression with regularization, and the formulas used in each method are derived from the logistic regression theory.

This report provides a technical explanation of the logistic regression implementation. The class includes methods for:

- Initializing parameters
- Computing the cost function
- Calculating gradients
- Checking convergence
- Training the model
- Predicting new data

2. __init__

The constructor initializes the logistic regression model. Key parameters include:

- alpha: The learning rate used for gradient descent. A small value (e.g., 0.01) controls the step size during optimization.
- regLambda: The regularization parameter, used to penalize large values in the parameters (theta).
- epsilon: A threshold value used for determining convergence.
- maxNumIters: The maximum number of iterations for the gradient descent algorithm.

```
def __init__(self, alpha=0.01, regLambda=0.01, epsilon=0.0001, maxNumIters=10000):
    self.theta = None
    self.alpha = alpha          # learning rate
    self.regLambda = regLambda   # regularization parameter
    self.epsilon = epsilon
    self.maxNumIters = maxNumIters
```

3. computeCost

Purpose: This method computes the cost function for logistic regression. The cost function measures how well the model's predictions match the actual output values. The formula for the cost function is:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

The cost function includes both the log loss term and the regularization term.

```
def computeCost(self, theta, X, y, regLambda):

    n = len(y)  # number of rows in X

    # cost function with regularization
    costVal = (-1 / n) * (np.dot(y.T, np.log(self.sigmoid(np.dot(X, theta)))) +
    np.dot((1 - y).T, np.log(1 - self.sigmoid(np.dot(X, theta)))))
    +   (regLambda / (2 * n)) * np.sum(np.square(theta[1:]))

    return costVal  # scalar value of cost
```

4. computeGradient

The gradient is given by the formula:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \quad \text{for} \quad j \geq 1$$

This method calculates the gradient of the cost function with respect to the model parameters, which is used to update the parameters during gradient descent.

```python
def computeGradient(self, theta, X, y, regLambda):
    n = len(y)  # number of rows in X

    # gradient for logistic regression with regularization
    gradVal = (1 / n) * (np.dot(X.T, (self.sigmoid(np.dot(X, theta)) - y)))

    # no regularization for the bias term
    gradVal[1:] += (regLambda / n) * theta[1:]

    return gradVal
```

5. hasConverged

The convergence condition is:

$$\sum_{j=1}^{n}(\theta_j^{(new)} - \theta_j^{(old)})^2 < \epsilon$$

This method checks whether the model has converged by calculating the difference between the old and new values of θ.

If the sum of the squares of the differences is smaller than a predefined threshold ϵ, the model is considered to have converged.

```python
def hasConverged(self, oldTheta, newTheta):

    # the sum of the squares of (new theta - old theta) from 1 to n inside the square root
    difference = newTheta - oldTheta

    # euclidean
    distance = np.sqrt(np.sum(difference**2))

    # distance is less than epsilon -> the model has converged
    return distance < self.epsilon
```

6. fit

The fit method trains the logistic regression model using gradient descent. The method iterates through the training process by repeatedly updating the parameters θ based on the computed gradients. If the model converges (i.e., the change in θ is smaller than ϵ), the training stops. The method also prints the cost at every 500th iteration to monitor the progress.

```python
# train the logistic regression model
def fit(self, X, y):

    y = y.reshape(-1, 1)  # (n,) -> (n, 1) to ensure y is a column vector

    n, d = X.shape  # n: rows , d: columns
    X = np.c_[np.ones((n, 1)), X]  # add a column of ones for the bias term

    np.random.seed(0)
    self.theta = np.random.rand(d + 1, 1)  # initialize theta with random values

    for i in range(self.maxNumIters):

        # update theta using gradient descent
        gradient = self.computeGradient(self.theta, X, y, self.regLambda)
        newTheta = self.theta - self.alpha * gradient

        # if the model has converged, stop the optimization process
        if self.hasConverged(self.theta, newTheta):
            self.theta = newTheta
            print(f"Model converged at iteration {i}")
            break

        # if not converged, update theta
        self.theta = newTheta

        # print cost for debugging every 500 iterations
        if i % 500 == 0:
            cost = self.computeCost(self.theta, X, y, self.regLambda)
            print(f"Iteration {i}, Cost: {cost}")
```

7. predict

This method predicts the probabilities for new data points using the learned parameters θ. It applies the sigmoid function to the linear combination of the input features, generating output probabilities. If the output is greater than 0.5, the model predicts class 1; otherwise, it predicts class 0.

```
def predict(self, X):

    n = X.shape[0]  # get the number of samples

    # add a column for bias term
    X = np.c_[np.ones((n,1)), X]  # add a column of ones for the bias term

    predictedValue = self.sigmoid(np.dot(X, self.theta))

    return (predictedValue >= 0.5).astype(int)  # return 0/1 according to the threshold
```

8. sigmoid

The sigmoid function transforms the linear output into a probability. It squashes any real-valued number into the range (0, 1), making it suitable for binary classification tasks.

```
def sigmoid(self, z):

    # 1 / (1 + e^(-z))
    return 1 / (1 + np.exp(-z))
```

9. Cost Function Values during Logistic Regression Training

This section provides an analysis of the cost values printed to the console at various stages of training for the logistic regression model. These values represent the cost function's output during each iteration of the gradient descent process, showing how the model minimizes the cost function over time to improve its predictive accuracy.

At the beginning of training, the cost value is 0.4165, indicating a high level of error in the model's predictions. This is expected since the model parameters are initialized randomly, and the model has not yet adjusted to fit the data.

As the iterations increase, the cost value steadily decreases, showing that the model is learning and improving its fit to the data. For example, at iteration 500, the cost reduces significantly to 0.3208, then to 0.2523 by iteration 2000, and continues to decline throughout training.

By iteration 9500, the cost has reached 0.2106. The rate of reduction slows down, showing the model is approaching convergence. This suggests that further iterations may yield diminishing returns, as the model has nearly minimized the cost function.

The decrease in cost reflects the model's improved ability to separate the classes correctly.

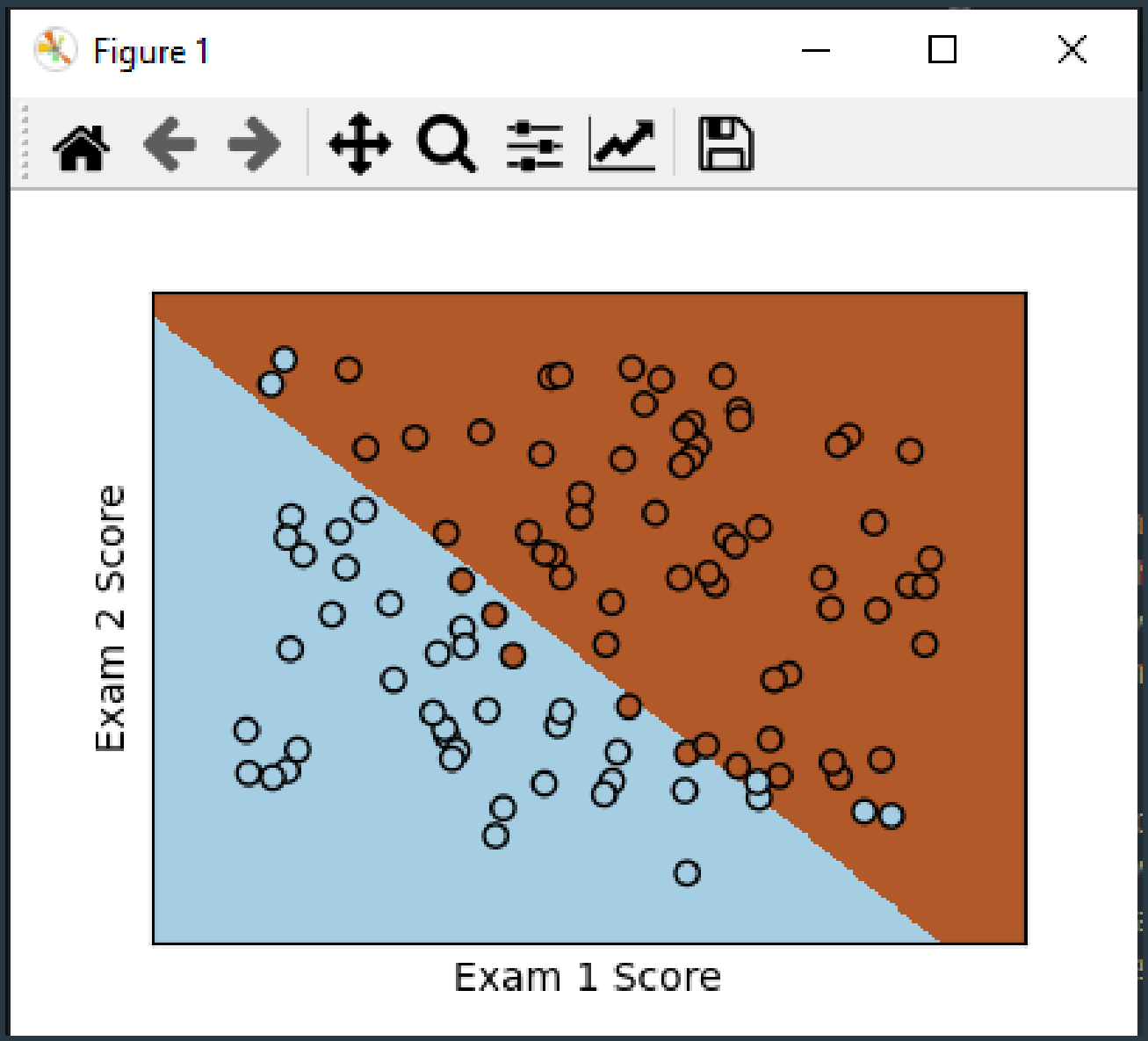
10. Cost Function Values during Logistic Regression Training

The following values were printed to the console at intervals of 500 iterations, providing insight into the model's learning progress:

```
Iteration 0, Cost: [[0.41654786]]
Iteration 500, Cost: [[0.32083517]]
Iteration 1000, Cost: [[0.28456259]]
Iteration 1500, Cost: [[0.26487358]]
Iteration 2000, Cost: [[0.25230462]]
Iteration 2500, Cost: [[0.24352772]]
Iteration 3000, Cost: [[0.23704041]]
Iteration 3500, Cost: [[0.23205304]]
Iteration 4000, Cost: [[0.22810629]]
Iteration 4500, Cost: [[0.22491292]]
Iteration 5000, Cost: [[0.22228325]]
Iteration 5500, Cost: [[0.2200866]]
Iteration 6000, Cost: [[0.21822979]]
Iteration 6500, Cost: [[0.21664451]]
Iteration 7000, Cost: [[0.21527946]]
Iteration 7500, Cost: [[0.21409536]]
Iteration 8000, Cost: [[0.21306161]]
Iteration 8500, Cost: [[0.21215401]]
Iteration 9000, Cost: [[0.21135315]]
Iteration 9500, Cost: [[0.21064332]]
```

11. Decision Boundary Visualization for Logistic Regression Model

This plot shows the decision boundary learned by the logistic regression model to classify data based on two exam scores. The diagonal line separates the two classes, with points on either side predicted as different classes. The boundary effectively divides most data points, showing the model's capability to distinguish between the classes.



10 . 11 . 2024

Thank you

for taking the time to read this report.

Submitted by: **EFE EROL**
No: **200101066**