Logistic Regression

Student: Polina Stepanenko

Overview

In this project, logistic regression was implemented from scratch to classify breast cancer data. The model was trained using gradient descent, where weights and bias were iteratively updated based on calculated gradients to minimize the loss function. The performance was evaluated by tracking changes in loss during training and assessing accuracy on a test set. Visualization of the loss trend provided insights into the model's convergence.

Logistic Regression Steps

- **Data Preparation.**
  Load the breast cancer dataset, separate it into training and test sets, and check data types to ensure compatibility.
- **Parameter Initialization.**
  Initialize weights and bias randomly, setting them up for gradient descent optimization.
- **Forward Propagation.**
  Compute the linear combination of inputs and weights, then apply the sigmoid activation to obtain predictions.
- **Loss Calculation.**
  Define a cost function (log loss) to measure prediction accuracy, adjusted to prevent numerical issues.
- **Backward Propagation.**
  Calculate gradients of weights and bias with respect to the loss to enable parameter updates.
- **Parameter Update.**
  Apply gradient descent to iteratively adjust weights and bias based on calculated gradients, using a predefined learning rate.
- **Model Training.**
  Execute the above steps iteratively for a set number of epochs to optimize the model.
- **Evaluation and Visualization.**
  Evaluate the model on the test set for accuracy, plot the loss trend over training epochs to analyze convergence.

Implementation of Logistic Regression Steps

Parameter Initialization

Listing 1: Initialization of weights and bias.

```python
def parameters_initialization(m):
    W = np.random.randn(1, m)  # Weight vector of shape (1, m)
    b = 0  # Initialize bias to 0
    return W, b
```

Forward Propagation

Listing 2: Computation of linear combinations of input features and weights, including the bias, as well as the application of the activation function.

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))  # Sigmoid activation function

def forwardPropagate(X, W, b):
    z = np.dot(W, X.T) + b  # Linear combination
    y_hat = sigmoid(z)      # Apply sigmoid activation
    return z, y_hat
```

Cost Calculation

Listing 3: Computation of the average loss over the entire training dataset.

```python
def cost(n, y_hat, y_true):
    ep = 10E-10  # Small epsilon to prevent log(0)
    total_sum = 0

    for i in range(n):
        total_sum += y_true[i] * np.log(y_hat[0][i] + ep) + (1 - y_true[i]) * np.log(1 - y_hat[0][i]
            + ep)

    return (-1 / n) * total_sum
```

Backward Propagation

Listing 4: Calculation of gradients of the objective function with respect to the weights and bias.

```python
def backwardPropagate(n, X, y_hat, y_true):
    dW = 1/n * np.dot((y_hat - y_true), X)  # Gradient for weights
    db = 0
    for i in range(n):
        db += y_hat[0][i] - y_true[i]  # Gradient for bias

    return dW, (1/n)*db
```

Parameter Update

Listing 5: Updating the weights and bias.

```python
def update(lr, dW, db, W, b):
    W = W - lr * dW  # Update weights
    b = b - lr * db  # Update bias
    return W, b
```

Results

The graph (Figure 1) visualizes the change in model loss during the training process of the logistic regression model. It shows how the loss decreases as the number of iterations increases, indicating the model's convergence.

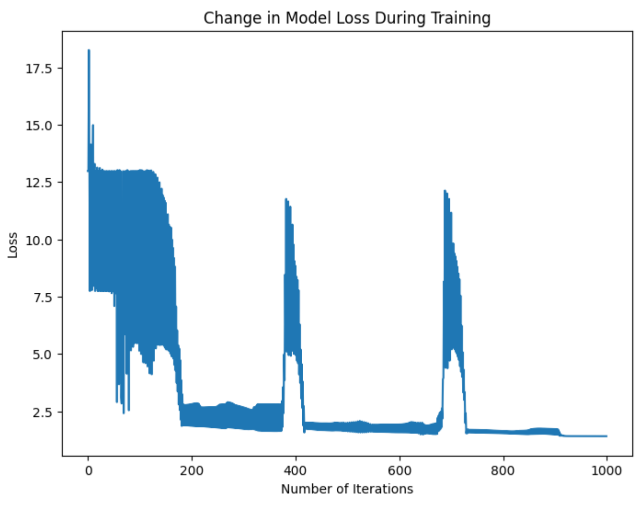

Figure 1: Change in Model Loss During Training.

Tables

To investigate the impact of learning rate and the number of iterations on the objective function value and the model's accuracy on the test set, multiple values for each parameter were explored, with both larger and smaller values than the defaults. The default values were set to 0.001 for the learning rate and 1000 for the number of iterations, and the results are shown below.

Iterations	**Learning Rate**	**Loss**	**Accuracy**	**Time (s)**
1000	0.01	1.8178	0.9122	2.03
1000	0.05	1.6360	0.9210	1.94
1000	0.001	1.2048	0.9298	2.05
1000	0.0005	1.6868	0.9122	1.86
1000	0.0001	1.4170	0.8684	1.98
1000	0.00001	1.6114	0.8245	1.86

Table 1: Impact of Learning Rate on Model Performance

Iterations	**Learning Rate**	**Loss**	**Accuracy**	**Time (s)**
200	0.001	1.6114	0.8245	0.516
500	0.001	1.5541	0.8947	1.08
750	0.001	1.5347	0.9035	1.45
1000	0.001	1.2048	0.9298	2.02
2000	0.001	1.1726	0.9298	3.39
5000	0.001	1.2035	0.9210	9.08

Table 2: Impact of Number of Iterations on Model Performance

Conclusions

In this project, we successfully implemented logistic regression from scratch to classify breast cancer data, following a structured approach that included data preparation, parameter initialization, forward propagation, loss calculation, backward propagation, and parameter updates through gradient descent.

Through experimentation with varying learning rates and iteration counts, we observed a direct relationship between these hyperparameters and model performance. The results highlighted the importance of fine-tuning the learning rate to balance between model convergence and overfitting, as well as the effect of iteration count on achieving optimal accuracy.

Our final model demonstrated significant accuracy on the test set, with the best configuration achieving over 92 % accuracy. Visualizing the loss over training epochs provided further insights into model behavior, showing a steady decrease in loss as the model converged. These findings emphasize logistic regression's effectiveness for binary classification tasks and underscore the importance of parameter optimization to maximize performance.