HW2 Report: Logistic Regression for Wine Classification

Course Name: AI and Machine Learning
Course Code: SDM274

Submitted By:

- **Name:** `Han Lubing`
- **Student ID:** `12210262`
- **Email:** `12210262@mail.sustech.edu.cn`

Date of Submission: `October 19, 2025`

Abstract

This report documents the implementation and evaluation of a logistic regression model for binary classification, built from scratch in Python. The project aims to classify wine samples into one of two categories based on their chemical properties, using the well-known Wine Dataset. The methodology involved preprocessing the dataset to create a binary classification problem, implementing a `LogisticRegression` class, and training it using two distinct gradient descent optimization methods: Mini-batch Gradient Descent (MBGD) and Stochastic Gradient Descent (SGD). The model's performance was rigorously evaluated on a held-out test set using four standard metrics: accuracy, precision, recall, and F1 score. Key outcomes show a significant performance difference between the two optimization methods, with the SGD-trained model achieving an F1 score of 97.9%, compared to 90.9% for the MBGD model. The findings demonstrate that for this dataset, SGD provides a more effective and robust training approach. This project serves as a practical exercise in understanding and implementing fundamental machine learning algorithms and evaluation principles.

Keywords: `Logistic Regression`, `Binary Classification`, `Gradient Descent`, `Wine Dataset`, `Model Evaluation`, `Machine Learning`

1. Introduction

1.1. Project Background & Motivation

The field of machine learning is built upon a foundation of core algorithms that solve common predictive tasks. Among the most fundamental of these is logistic regression, a powerful and interpretable method for binary classification. The motivation for this project is to gain a deep, practical understanding of logistic regression not just by using a pre-built library, but by implementing the algorithm from the ground up. This includes coding the model structure, the gradient descent optimization loop, and the evaluation metrics. The Wine Dataset, a classic benchmark from the UCI Machine Learning Repository, provides an ideal, real-world context for this task.

1.2. Summary of Your Project

This project involves building a complete binary classification pipeline in Python. First, the multi-class Wine Dataset is loaded and transformed into a two-class problem. A custom `LogisticRegression` class is implemented, capable of being trained via both Mini-batch and

Stochastic Gradient Descent. The data is then split into training and testing sets, and models are trained using both optimization methods. Finally, the performance of each model is measured and compared on the test set to determine the more effective approach.

2. Problem Description and Project Objectives

2.1 Problem Description
The core problem is to create a binary classification model that can distinguish between two different classes of wine based on a set of 13 chemical features. The original Wine Dataset contains 178 wine samples with three classes, but we remove one class to create a binary classification problem. This involves handling a real-world dataset, implementing a machine learning model from scratch, training it effectively, and evaluating its predictive performance in a quantitative manner.

2.2 Primary Objective:
The specific aims of this project are as follows:

- To implement a complete Logistic Regression model in Python using the NumPy library.
- To process the multi-class Wine Dataset, removing one class to frame it as a binary classification task.
- To implement and train the model using two different optimization strategies: Mini-batch Gradient Descent and Stochastic Gradient Descent.
- To write code that evaluates the classification performance of the trained models using the metrics of Accuracy, Precision, Recall, and F1 Score.
- To analyze and compare the results of the two training methods and draw conclusions about their effectiveness for this problem.

3. Design & Methodology

3.0. Theoretical Background

Logistic Regression Overview:
Logistic regression is a statistical method for binary classification that models the probability of an event occurring. Unlike linear regression which outputs continuous values, logistic regression outputs probabilities between 0 and 1 using the sigmoid function.

Mathematical Foundation:

1. **Sigmoid Function**: The core of logistic regression is the sigmoid (logistic) function that maps any real number to a value between 0 and 1:
   $\sigma(z) = \frac{1}{1+e^{-z}}$
   where $z = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{w}^T\mathbf{x}$
2. **Hypothesis Function**: The predicted probability is given by:
   $h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
3. **Cross-Entropy Loss Function**: For binary classification, we use the cross-entropy loss:
   $J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h(\mathbf{x}^{(i)})) + (1-y^{(i)})\log(1-h(\mathbf{x}^{(i)}))]$
   where m is the number of training examples, $y^{(i)}$ is the true label, and $h(\mathbf{x}^{(i)})$ is the predicted probability.
4. **Gradient of the Loss**: The gradient of the cross-entropy loss with respect to weights is:

$$\nabla J(\mathbf{w}) = \frac{1}{m}\mathbf{X}^T(\mathbf{h}(\mathbf{x}) - \mathbf{y})$$

5. **Weight Update Rule**: Using gradient descent, weights are updated as:

   $$\mathbf{w} = \mathbf{w} - \alpha \nabla J(\mathbf{w})$$

   where α is the learning rate.

Optimization Methods:

- **Mini-batch Gradient Descent**: Updates weights using a subset of training data (batch size > 1)
- **Stochastic Gradient Descent**: Updates weights using one training example at a time (batch size = 1)

3.1. System Architecture / Overall Design

The project follows a standard machine learning workflow, which can be broken down into several sequential stages:

1. **Data Loading & Preprocessing**: The `wine.data` file containing 178 wine samples with 13 chemical features is loaded. Samples belonging to class `3` are filtered out to create a binary classification problem, and the labels for classes `1` and `2` are remapped to `0` and `1` respectively.
2. **Data Splitting**: The dataset is randomly shuffled and split into a 70% training set and a 30% test set as required by the assignment.
3. **Model Initialization**: An instance of the `LogisticRegression` class is created.
4. **Model Training**: The `fit` method is called on the training data with either 'mini_batch' or 'stochastic' update methods. Inside this method, the model iterates for a set number of epochs (5000 in our implementation). In each epoch, it processes the data in batches (batch size 16 for mini-batch, or one-by-one for SGD), calculates the predicted output using the sigmoid function, computes the gradient of the cross-entropy loss, and updates the model's weights using gradient descent.
5. **Prediction**: The trained model is used to make predictions on the unseen test set.
6. **Evaluation**: The predictions are compared against the true labels from the test set, and the performance metrics (Accuracy, Precision, Recall, F1) are calculated.

3.2. Implementation Details

The core of the project is the `LogisticRegression.py` script. Key components are highlighted below.

- **Data Loading:** The `load_data` function handles the initial data preparation.

```python
def load_data(file_path):
    data = np.loadtxt(file_path, delimiter=',')
    data = data[data[:, 0] != 3] # Remove class '3'
    X = data[:, 1:]
    y = data[:, 0]
    y = np.where(y == 1, 0, 1) # Convert classes 1,2 to 0,1
    return X, y.reshape(-1, 1)
```

- **Data Splitting:** The `split_data` function randomly shuffles the dataset and splits it into training and test sets with a 7:3 ratio as required by the assignment.

```python
def split_data(X, y, test_size=0.3):
    indice = list(range(X.shape[0]))
    np.random.shuffle(indice)
    split_idx = int(X.shape[0] * (1 - test_size))
    X_train, X_test = X[indice[:split_idx]], X[indice[split_idx:]]
    y_train, y_test = y[indice[:split_idx]], y[indice[split_idx:]]
    return X_train, X_test, y_train, y_test
```

- **Model Training:** The `fit` method orchestrates the training process, implementing the mathematical formulas described in the theoretical background. The `gradient` method computes the gradient of the cross-entropy loss using the formula $\nabla J(\mathbf{w}) = \frac{1}{m}\mathbf{X}^T(\mathbf{h}(\mathbf{x}) - \mathbf{y})$, which is the core of the learning process. The learning rate α is set to 0.0001 and the model trains for 5000 epochs.

```python
class LogisticRegression:
    def sigmoid(self, z):
        z = np.clip(z, -500, 500)  # Prevent overflow
        return 1 / (1 + np.exp(-z))  # Implements σ(z) = 1/(1+e^(-z))

    def fit(self, X, y, update_method='mini_batch'):
        self.init_weights(X.shape[1])

        for i in range(self.epoch):
            if update_method == 'stochastic':
                self.batch_size = 1
            for X_batch, y_batch in self.iter_mini_batch(X, y):
                y_pred = self.predict(X_batch)  # Implements h(x) = σ(w^T x)
                gradient = self.gradient(X_batch, y_batch, y_pred)
                self.weights -= self.lr * gradient  # Implements w = w -
α∇J(w)
        return self.weights

    def gradient(self, X, y, y_pred):
        X_ = self.preprocess(X)
        return X_.T @ (y_pred - y.reshape(y_pred.shape))/X.shape[0]  #
Implements ∇J(w) = (1/m)X^T(h(x) - y)
```

- **Model Evaluation:** The `evaluate` function takes the true labels and the model's predictions and calculates the four key performance metrics using the following mathematical formulas:

Mathematical Definitions:

1. **Accuracy**: The proportion of correctly classified samples:
   $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$
2. **Precision**: The proportion of positive predictions that are actually positive:
   $\text{Precision} = \frac{TP}{TP+FP}$
3. **Recall**: The proportion of actual positives that are correctly identified:
   $\text{Recall} = \frac{TP}{TP+FN}$
4. **F1 Score**: The harmonic mean of precision and recall:
   $\text{F1 Score} = 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$

Where:

- TP = True Positives (correctly predicted positive cases)
- TN = True Negatives (correctly predicted negative cases)
- FP = False Positives (incorrectly predicted positive cases)
- FN = False Negatives (incorrectly predicted negative cases)

```python
def evaluate(y_true, y_pred):
    y_pred = (y_pred >= 0.5).astype(int)
    acurracy = np.sum(y_true == y_pred) / y_true.shape[0]
    precision = np.sum((y_true == 1) & (y_pred == 1)) / np.sum(y_pred == 1)
    recall = np.sum((y_true == 1) & (y_pred == 1)) / np.sum(y_true == 1)
    F1_score = 2 * (precision * recall) / (precision + recall)
    return acurracy, precision, recall, F1_score
```

4. Testing & Results

4.1. Testing

The model's performance was tested on a held-out portion of the dataset, comprising 30% of the original data as required by the assignment. The training was performed on the other 70%. This ensures that the evaluation is done on data that the model has not seen during training, providing an unbiased estimate of its generalization ability. Both Mini-batch and Stochastic Gradient Descent methods were implemented and compared as specified in the requirements.

4.2. Results and Analysis

The script was executed to train two separate models—one with Mini-batch Gradient Descent and one with Stochastic Gradient Descent. The performance of each on the test set is as follows:

- **Table: Model Performance Results**

Metric	Mini-batch GD	Stochastic GD
Accuracy	0.8974	**0.9744**
Precision	0.9524	**0.9583**
Recall	0.8696	**1.0000**
F1 Score	0.9091	**0.9787**

- **Analysis:** The results clearly indicate that the **Stochastic Gradient Descent (SGD) model significantly outperformed the Mini-batch model**.
    - **Overall Performance**: The SGD model achieved a much higher Accuracy (97.4% vs 89.7%) and, most importantly, a much higher F1 Score (97.9% vs 90.9%). The F1 score is often the best metric for overall performance as it balances Precision and Recall.
    - **Precision-Recall Trade-off**: Both models performed exceptionally well, but SGD showed superior performance across all metrics. The Mini-batch model achieved high precision (95.2%) and good recall (87.0%), while the SGD model achieved even higher precision (95.8%) and perfect recall (100.0%), meaning it identified every true positive sample without any false negatives.

   - **Conclusion of Analysis**: For this task, the SGD model is clearly superior. Its ability to achieve perfect recall while maintaining high precision makes it an extremely effective classifier. The sample-by-sample updates of SGD likely helped it explore the solution space more thoroughly and converge to a better solution than the Mini-batch method.

5. Conclusion

This project successfully achieved its objectives. A logistic regression model was implemented from scratch and used to solve a binary classification problem on the Wine Dataset. The comparison between Mini-batch and Stochastic Gradient Descent revealed that SGD was the more effective optimization strategy, yielding a model with an accuracy of 97.4% and an F1 score of 97.9%.

The primary learning from this project was the practical demonstration of how different optimization strategies can lead to significantly different results, with SGD achieving superior performance across all metrics. The project also highlighted the importance of understanding the trade-offs between evaluation metrics like precision and recall, though in this case, SGD achieved both high precision and perfect recall.

Potential Optimizations:
The model's performance could be further improved. Two highly recommended next steps would be:

1. **Feature Scaling**: Applying standardization to the features to bring them all to a similar scale would likely speed up convergence and improve final accuracy.
2. **Regularization**: Adding L2 regularization would help prevent the model from overfitting and could improve its generalization performance, as well as help with numerical stability.