Homework: Logistic Regression with Gradient Descent in Python

Overview:

In this hands-on assignment, you will classify images of handwritten digits (specifically digits 0 and 1) using logistic regression. You will:

1. Train a logistic regression model using scikit-learn and evaluate its performance.
2. Implement logistic regression from scratch using gradient descent and binary cross-entropy loss.
3. Visualize your model's behavior using informative plots, including confusion matrix and misclassified examples.

Dataset:

We will use the MNIST dataset (from Keras) which includes 28x28 grayscale images of handwritten digits. You will filter this dataset to include only digits 0 and 1 for binary classification.

```
from tensorflow.keras.datasets import mnist

from sklearn.preprocessing import StandardScaler

import numpy as np

# Load dataset

(X_train_full, y_train_full), (X_test_full, y_test_full) =
mnist.load_data()


# Filter digits 0 and 1

mask_train = (y_train_full == 0) | (y_train_full == 1)

mask_test = (y_test_full == 0) | (y_test_full == 1)


X_train = X_train_full[mask_train].reshape(-1, 28*28) / 255.0

y_train = y_train_full[mask_train]


X_test = X_test_full[mask_test].reshape(-1, 28*28) / 255.0

y_test = y_test_full[mask_test]
```

Then, split your data into train and test sets and standardize the features using `StandardScaler`.

Part 1: Sanity Check with scikit-learn

Use scikit-learn's `LogisticRegression` model to:

- Fit the training data
- Predict the labels on the test set
- Calculate and print the accuracy and log loss

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, log_loss

# Standardize data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train scikit-learn logistic regression
clf = LogisticRegression()
clf.fit(X_train_scaled, y_train)
y_pred = clf.predict(X_test_scaled)
y_prob = clf.predict_proba(X_test_scaled)[:, 1]

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Log Loss:", log_loss(y_test, y_prob))
```

Part 2: Implement Logistic Regression from Scratch

You will now implement logistic regression using NumPy.

Tasks:

- Implement the sigmoid function
- Implement binary cross-entropy loss
- Use gradient descent to update weights over 100 epochs
- Track and store loss values over iterations for both training and test sets
- Compare results after **5 epochs** and **100 epochs**

```
def sigmoid(z):
    return
# Standardize again and add intercept
X_train_manual = scaler.transform(X_train)
X_manual = np.hstack([np.ones((X_train_manual.shape[0], 1)),
X_train_manual])
y_manual = y_train.reshape(-1, 1)
# Initialize weights
W = np.zeros((X_manual.shape[1], 1))
# Set hyperparameters
lr =
epochs =
train_loss_history = []
test_loss_history = []

# Gradient descent loop
for i in range(epochs):
    z =
    y_hat = sigmoid(z)
    # Compute loss (binary cross-entropy)
    loss = -np.mean(y_manual * np.log(y_hat + 1e-10) + (1 - y_manual) *
np.log(1 - y_hat + 1e-10))
```

```python
    train_loss_history.append(loss)


# Compute test loss

y_hat_test = sigmoid(X_test_manual @ W)

test_loss = -np.mean(y_test.reshape(-1, 1) * np.log(y_hat_test + 1e-10) +
(1 - y_test.reshape(-1, 1)) * np.log(1 - y_hat_test + 1e-10))

test_loss_history.append(test_loss)


    # Compute gradient

    grad =

    W -= lr * grad

    if (i + 1) % 10 == 0:

        print(f"Epoch {i + 1}, Loss: {loss:.4f}")
```

Part 3: Evaluation and Visualization

You will now evaluate your custom implementation and visualize results.

Tasks:

1. **Accuracy and Log Loss**: Evaluate your manual model after 5 and 100 epochs.
2. **Plot 1: Loss vs Iteration** — Plot binary cross-entropy loss over training epochs for both training and test sets.
3. **Plot 2: Confusion Matrix** — Use scikit-learn to generate a confusion matrix for predictions.
4. **Plot 3: Important Pixels** — Visualize model weights (excluding bias) as a 28x28 image.
5. **Plot 4: Misclassified Examples** — Display up to 10 misclassified images with predicted label and confidence.
6. **Compare with scikit-learn model** — Print scikit-learn accuracy again and show its misclassified examples.

```
# Prepare test data with intercept

X_test_manual = np.hstack([np.ones((X_test_scaled.shape[0], 1)),
X_test_scaled])

# Predict and evaluate

y_prob_manual = sigmoid(X_test_manual @ W)

y_pred_manual = (y_prob_manual >= 0.5).astype(int)


print("Accuracy:", accuracy_score(y_test, y_pred_manual))

print("Log Loss:", log_loss(y_test, y_prob_manual))


# Plot 1: Loss vs Iteration


# Plot 2: Confusion Matrix


# Plot 3: Important Pixels


# Plot 4: Misclassified Examples

# Compare with scikit-learn

print("scikit-learn accuracy:", accuracy_score(y_test, y_pred))
```

Submission Instructions:

- Submit a PDF that includes all plots (loss, confusion matrix, important pixels, misclassified examples).
- Also submit a .py file with your code.
- Name your files as follows: `<id>.pdf` and `<id>.py` (e.g., `123456789.pdf`, `123456789.py`).