Linear Regression Homework

Overview

In this hands-on assignment, we will use linear regression to predict the progression of diabetes in patients. The goal is to understand how linear regression works and how it can be implemented using gradient descent.

We will first train a linear regression model using **scikit-learn**, a popular Python machine learning library. This model will serve as a reference implementation. Then, you will implement linear regression manually using gradient descent and compare your results to the scikit-learn model.

By the end of this assignment, you should be able to:

1. Load and prepare a real dataset.
2. Train a linear regression model using scikit-learn.
3. Implement linear regression using gradient descent.
4. Compare your implementation with scikit-learn.
5. Visualize the training process and the final model performance.

Dataset

In this assignment, we will use the diabetes dataset provided by scikit-learn. The dataset contains information from 442 diabetes patients. For each patient, there are 10 input variables: age, sex, body mass index, average blood pressure, and six blood serum measurements. The blood serum measurements are Total Cholesterol (TC), Low Density Lipoprotein (LDL), High Density Lipoprotein (HDL), TC/HDL, Low Tension Glaucoma (LTG), and Glucose. The target value is a quantitative measure of disease progression one year after the baseline measurements.

More information about the dataset can be found at: `https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html`

The following code loads the diabetes dataset and splits it into training and testing sets. The last 20 samples are used as the test set, and the remaining samples are used for training.

```
diabetes = datasets.load_diabetes() # Load diabetes dataset
diabetes_X = diabetes.data # Input matrix of dimensions 442 x 10
diabetes_y = diabetes.target # Target values

# Split the data into training and testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the target values into training and testing sets
diabetes_y_train = diabetes_y[:-20]
```

```
diabetes_y_test = diabetes_y[-20:]
```

Feature Standardization

Before training, we standardize the features to have zero mean and unit variance using StandardScaler. This ensures all features contribute equally to the model and improves the stability and speed of gradient descent.

We use StandardScaler from scikit-learn:

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

From this point onward, we use the standardized dataset (X_scaled) for both training and evaluation.

Reference Model Using scikit-learn

Before implementing linear regression manually, we first train a linear regression model using scikit-learn. This will give us reference values for the model coefficients and mean squared error. The linear regression model in scikit-learn can be trained with just a call to function fit on the model since scikit- learn is a machine learning library. You can see the documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html).

```
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
mean_squared_error = metrics.mean_squared_error(diabetes_y_test, di
abetes_y_pred)
print("Mean squared error: %.2f" % mean_squared_error)
print("="*80)
```

The mean squared error obtained using scikit-learn will serve as a reference for your own implementation. Your result does not need to be exactly identical, but it should be reasonably close (say within 1%). You should also compare the coefficients learned by your implementation with the coefficients obtained from scikit-learn.

Implementing Linear Regression with Gradient Descent

Now, it's time to implement linear regression ourselves. Here's a template for you to get started.

```
# train
X = diabetes_X_train_scaled
y = diabetes_y_train

# train: init
W = ...
b = ...
learning_rate = ...
epochs = ...

# train: gradient descent
# Note: Save the mean squared error at each iteration in a list (e.g.,
mse_history)
# for later visualization (MSE vs. iteration plot).
mse_history = []

for i in range(epochs):
    # calculate predictions
    # TODO

    # calculate error and cost (mean squared error)
    # TODO
    # mse_history.append(current_mse)

    # calculate gradients
    # TODO

    # update parameters
    # TODO

    # diagnostic output
    if i % 1000 == 0:
        print("Epoch %d: %f" % (i, current_mse))
```

Visualizing Your Model's Performance

Once you have completed your gradient descent implementation and tested it against the results from scikit-learn, you will now visualize your model's behavior and performance.

Below is a plotting template. You must create three plots to evaluate the learning process and your trained model.

1. **Plot MSE vs. Iteration**
    - Plot the training loss (mean squared error) over the course of training.

- Use the values you stored in mse_history.
- This helps you visualize whether the model is learning and if the error is decreasing.

2. **Plot True vs. Predicted Values**
    - Create a scatter plot comparing the model's predicted values (on the test set) with the true target values.
    - A dashed diagonal line ($y = x$) can help show how close your predictions are to the ideal output.
3. **Plot Feature Importance**
    - Visualize the absolute value of each learned coefficient.
    - This shows which features were most important in the model's decision process.
    - You must use the weights from your manual gradient descent implementation (W), not scikit-learn's.

Submission Instructions

Submit a PDF file that includes all required plots: MSE vs. iterations, true vs. predicted values, and feature importance. In addition, submit a `.py` file containing your full Python code.

Name your files according to your student ID number, using the following format:
`<id>.pdf` and `<id>.py` (i.e., `123456789.pdf` and `123456789.py`).