Machine Learning Lab (AI332L)

Class:	BS Artificial Intelligence	**Semester:**	5th (Fall 2023)
Instructor:	Dr. M. Ehatisham-ul-Haq	**Email:**	ehtisham@mail.au.edu.pk

| Lab 06 |

Gradient Descent in Machine Learning with Python

Lab Objective:

In this lab tutorial, we will explore the concept of Gradient Descent, a powerful optimization algorithm used in machine learning to find the optimal parameters of a model.

Setting up Python with Google Colab

1. Go to [Google Colab](https://colab.research.google.com).
2. Create a new notebook by clicking on **File** > **New Notebook**.
3. You're now ready to start writing and executing Python code in the notebook!

Building Regression Model with Gradient Descent

1. Load and Explore: Boston Housing Price Dataset

We can use the Boston Housing Prices dataset available in scikit-learn for solving the multivariable regression problem with Gradient Descent.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler

# Step 1: Load and Explore the Dataset
boston = load_boston()
X = boston.data
y = boston.target

# Step 2: Data Preprocessing (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Visualize the dataset
plt.scatter(X_scaled[:, 5], y)  # Example: Using one feature for visualization
plt.xlabel('Feature')
plt.ylabel('Price')
```

```
plt.title('Boston Housing Prices Dataset')
plt.show()
```

2. Implement Gradient Descent

```
# Step 1: Define Cost Function and Gradient

# Define cost function (Mean Squared Error)
def cost_function(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    error = predictions - y
    return np.sum(error**2) / (2 * m)

# Define gradient function
def gradient(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    error = predictions - y
    return X.T.dot(error) / m

# Step 2: Implement Gradient Descent

def gradient_descent(X, y, theta, learning_rate, iterations):
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        theta = theta - learning_rate * gradient(X, y, theta)
        cost_history[i] = cost_function(X, y, theta)

    return theta, cost_history

# Initialize parameters
X_b = np.c_[np.ones((len(X_scaled), 1)), X_scaled]  # Add bias term
theta = np.random.randn(X_b.shape[1], 1)  # Random initialization for parameters
(including bias)
learning_rate = 0.01
iterations = 1000

# Apply Gradient Descent
theta_final, cost_history = gradient_descent(X_b, y, theta, learning_rate,
iterations)
print(f'Optimal theta: {theta_final}')

# Step 3: Visualize Convergence

plt.plot(range(1, iterations+1), cost_history, color='blue')
plt.rcParams["figure.figsize"] = (10, 6)
```

```
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost (J)')
plt.title('Convergence of Gradient Descent')
plt.show()
```

Building Classification Model with Gradient Descent using Logistic Regression

We will use the Iris dataset for this classification task. The Iris dataset is a widely used dataset for classification, and it is available in **scikit-learn**.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Data Preprocessing (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Initialize and Train the Model using SGD
sgd_classifier = SGDClassifier(loss='log', max_iter=1000, random_state=42)
sgd_classifier.fit(X_train, y_train)

# Step 5: Predict
y_pred_train = sgd_classifier.predict(X_train)
y_pred_test = sgd_classifier.predict(X_test)

# Step 6: Evaluate Model Performance
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f'Training Accuracy: {train_accuracy}')
print(f'Test Accuracy: {test_accuracy}')
```

One-vs.-One Classification:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Data Preprocessing (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Initialize and Train the Model using OvO strategy
ovo_classifier = OneVsOneClassifier(SGDClassifier(loss='log', max_iter=1000,
random_state=42))
ovo_classifier.fit(X_train, y_train)

# Step 5: Predict
y_pred_train_ovo = ovo_classifier.predict(X_train)
y_pred_test_ovo = ovo_classifier.predict(X_test)

# Step 6: Evaluate Model Performance
train_accuracy_ovo = accuracy_score(y_train, y_pred_train_ovo)
test_accuracy_ovo = accuracy_score(y_test, y_pred_test_ovo)

print(f'Training Accuracy (OvO): {train_accuracy_ovo}')
print(f'Test Accuracy (OvO): {test_accuracy_ovo}')
```

One-vs.-Rest Classification:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import SGDClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

```
# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Data Preprocessing (Standardization)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Step 4: Initialize and Train the Model using OvA strategy
ova_classifier = OneVsRestClassifier(SGDClassifier(loss='log', max_iter=1000,
random_state=42))
ova_classifier.fit(X_train, y_train)

# Step 5: Predict
y_pred_train_ova = ova_classifier.predict(X_train)
y_pred_test_ova = ova_classifier.predict(X_test)

# Step 6: Evaluate Model Performance
train_accuracy_ova = accuracy_score(y_train, y_pred_train_ova)
test_accuracy_ova = accuracy_score(y_test, y_pred_test_ova)

print(f'Training Accuracy (OvA): {train_accuracy_ova}')
print(f'Test Accuracy (OvA): {test_accuracy_ova}')
```

Lab Task(s):

1. Write the code to predict the value of a House based on the input features of your own choice. You must use the optimized theta parameters generated by the Gradient Descent.
2. Use any of the publically available datasets for the Multivariate Regression problem. Use `SGDRegressor` and `LinearRegressor` to train and test the models
3. Compare the results (i.e., theta parameter values) obtained in part (2) with those obtained using SGDRegressor and LinearRegressor.
4. Finally, evaluate the model performance for the Regression cases and analyze the results.
5. Take any multiclass dataset and use the OVO and OVA classification methods to train and test the model. Compare the performance of both strategies for classification.