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 04 |

Building, Analyzing, and Testing Different Types of Regression Models

Lab Objective:

This lab tutorial will guide you on analyzing, building, and testing regression models in Python. We will use the popular **scikit-learn** library for this purpose.

Setting up Python with Google Colab

1. Go to Google Colab.
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 and Testing Linear Regression Models in Python

1. Load and Explore: Diabetes Dataset

Let us use the "Diabetes" dataset available in **scikit-learn**, which contains ten baseline variables, six blood serum measurements, age, sex, body mass index, average blood pressure, and six blood serum measurements for 442 diabetes patients.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_diabetes

# Load the dataset
diabetes = load_diabetes()
data = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
data['TARGET'] = diabetes.target

# Explore the dataset
print(data.head())
print(data.info())
print(data.describe())
```

2. Dataset Preprocessing

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split data into features (X) and target variable (y)
X = data.drop('TARGET', axis=1)
y = data['TARGET']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

3. Building Regression Models

3.1. Linear Regression

```
from sklearn.linear_model import LinearRegression

# Initialize the model
model_lr = LinearRegression()

# Train the model
model_lr.fit(X_train, y_train)
```

3.2. Ridge Regression

```
from sklearn.linear_model import Ridge

# Initialize the model
model_ridge = Ridge(alpha=1.0)

# Train the model
model_ridge.fit(X_train, y_train)
```

3.3. Lasso Regression

```
from sklearn.linear_model import Lasso

# Initialize the model
model_lasso = Lasso(alpha=1.0)

# Train the model
model_lasso.fit(X_train, y_train)
```

4. Building Linear Regression Models

```
from sklearn.metrics import mean_squared_error

# Predict using the models
y_pred_lr = model_lr.predict(X_test)
y_pred_ridge = model_ridge.predict(X_test)
y_pred_lasso = model_lasso.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

print(f'MSE Linear Regression: {mse_lr}')
print(f'MSE Ridge Regression: {mse_ridge}')
print(f'MSE Lasso Regression: {mse_lasso}')
```

POINTS TO PONDER:

1) **Differentiate between Linear, Ridge, and Lasso Regression.**
2) **Analyze the MSE values to determine which model performed best and why.**
3) **Observe the results with different hyperparameters for Ridge and Lasso regression.**
4) **Explore what method is being used at the backend for computing the values of theta parameters.**

5. Building Non-Linear Regression Model

Nonlinear regression models are used when the relationship between the independent and dependent variables is not linear. One popular nonlinear regression model is the Polynomial Regression. In the next step, we will perform Polynomial Regression on the "Diabetes" dataset.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Define the degree of the polynomial
degree = 2

# Create a pipeline with Polynomial Features and Linear Regression
model_poly = make_pipeline(PolynomialFeatures(degree), LinearRegression())

# Train the model
model_poly.fit(X_train, y_train)

# Predict using the model
y_pred_poly = model_poly.predict(X_test)
```

```
# Calculate Mean Squared Error (MSE) for Polynomial Regression
mse_poly = mean_squared_error(y_test, y_pred_poly)

print(f'MSE Polynomial Regression (Degree {degree}): {mse_poly}')
```

POINTS TO PONDER:

What is the working mechanism behind the above code?

The use of a pipeline with Polynomial Features and Linear Regression allows us to seamlessly combine the process of transforming the features into polynomial features and fitting a linear regression model in a single step. Here is a breakdown of why we use this approach:

Step-1 - Polynomial Features: Polynomial regression is essentially a linear regression on transformed features. It transforms the original features into higher-degree polynomial features. For example, if we have a single feature, `x`, and we want to perform polynomial regression with degree 2, it will transform the feature into `x` and `$x^2$`.This transformation allows the model to capture nonlinear relationships between the features and the target variable.

Step-2 - Linear Regression: After transforming the features using Polynomial Features, we are left with a set of new features, which may be of higher degree. However, the relationship between these transformed features and the target variable is still linear.

S**tep-3 - Pipeline:** A pipeline in **scikit-learn** allows us to chain multiple processing steps together. In this case, we first apply the Polynomial Features transformation, followed by fitting a Linear Regression model. The pipeline ensures that the same preprocessing steps are applied to both the training and testing data.

This approach simplifies the modeling process and allows us to utilize the existing tools for linear regression, including evaluation metrics like Mean Squared Error.

Can we directly use Polynomial Regression without using the Pipeline?

Yes, you can apply Polynomial Regression directly without using a pipeline. In scikit-learn, you can use the PolynomialFeatures class to transform your features, and then apply a Linear Regression model on the transformed features. This approach is also valid and provides you with more flexibility if you want to explore the intermediate steps or apply additional customizations.

Here is an example of how you can do it:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Define the degree of the polynomial
degree = 2

# Create polynomial features
```

```
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X_train)

# Train a linear regression model on the polynomial features
model_poly = LinearRegression()
model_poly.fit(X_poly, y_train)

# Predict using the model
X_test_poly = poly_features.transform(X_test)
y_pred_poly = model_poly.predict(X_test_poly)
```

You can access these coefficients using **model_poly.coef_** and **model_poly.intercept_.** Remember that the equation becomes more complex for higher-degree polynomials and involves multiple coefficients for each feature.

Lab Task:

Use any of the publically available datasets for the Multivariate Regression problem. The dataset must include a mix of numerical and categorical/ordinal features. Perform the following steps on the data.

1) Load and Explore the Dataset

a) Check for missing values, if any, and handle them appropriately.
b) Generate summary statistics for the dataset.
c) Split the data into features (X) and target variable (y).
d) Encode the categorical/ordinal variables using relevant encoding techniques.
e) Find and plot the correlation between different variables.

2) Build and Train Regression Models

a) Choose different regression models (e.g., Linear Regression, Ridge Regression, Lasso Regression, and Polynomial Regression). Train them using the features and target variable.
b) Train these models with Feature Normalization and Standardization as well.

3) Evaluate Model Performance

a) Predict the target variable using the trained models and calculate the Mean Squared Error (MSE).
b) Compare the performance of the chosen regression models with and without feature scaling.
c) Visualize the predictions against the actual values for better understanding.
d) Take a random test sample and predict its value.
e) Prepare a 1-2 page report on the analysis of results for different regression models.