Indian Institute of Technology Dharwad
CS214: Artificial Intelligence Laboratory
Lab 8 – Regression with single input variable

Problem Statement

You are given the dataset `AirQuality.csv`, which contains various air pollution indicators. Your task is to predict **CO concentration (mg/m^3)** using regression models.

Dataset Details

- **Input Variables:** Different air pollution sensor readings.
  - PT08.S1(CO): Tin oxide sensor response (nominally CO targeted)
  - NMHC: Non-Methanic HydroCarbons concentration (microg/m^3)
  - C6H6(GT): Benzene concentration (microg/m^3)
  - PT08.S2(NMHC): Titania sensor response (nominally NMHC targeted)
  - NOx: NOx concentration (ppb)
  - PT08.S3(NOx): Tungsten oxide sensor response (nominally NOx targeted)
  - NO2(GT): NO2 concentration (microg/m^3)
  - PT08.S4(NO2): Tungsten oxide sensor response (nominally NO2 targeted)
  - PT08.S5(O3): Indium oxide sensor response (nominally O3 targeted)
  - T: Temperature (°C)
  - RH: Relative Humidity
  - AH: Absolute Humidity
- **Output Variable:** CO concentration (mg/m^3).
- Note: Lab 8 is on Regression with single input variable. For all the below tasks, only 1 input feature is used from the dataset, i.e., **PT08.S1(CO)**.

Tasks

1. **Split the data into 60% training data, 20% validation data, and 20% test data.**

2. Simple Linear Regression

1. Build a simple linear regression model to predict CO concentration using **PT08.S1(CO)** as the independent variable.
2. **Plot:** Best fit line on the training data (x-axis: PT08.S1(CO), y-axis: CO concentration).
3. Compute **RMSE** on training, validation, and test data.
4. **Scatter Plot:** Actual vs Predicted CO concentration on the test data.

3. Polynomial Curve Fitting

1. Build regression models with degrees $p = 2, 3, 4, 5$ to predict CO concentration using **PT08.S1(CO)** as the independent variable.
2. Compute RMSE on training and validation dataset.
3. **Plot:** Best fit curves for each polynomial degree.
4. Select the polynomial degree with the lowest validation RMSE and evaluate the selected model on test data.
5. Compute **RMSE** for test data.
6. **Bar Graph:** RMSE of validation data vs Degree of Polynomial.
7. **Scatter Plot:** Actual vs Predicted CO concentration on test data for the best polynomial degree.

4. Neural network

1. Select **PT08.S1(CO)** as the independent variable to predict CO concentration.
2. Build a simple neural network model using PyTorch with:
   - **Input layer:** 1 linear neuron.
   - **Hidden layer:** 1 layer with Sigmoid activation function, varying the number of neurons (8, 16, 32, 64) to analyze the impact on model performance.
   - **Output layer:** 1 linear neuron.
   - **Loss function:** Mean Squared Error (MSE).
   - **Optimizer:** Stochastic Gradient Descent (SGD).
3. Train the model with varying hidden neurons and evaluate the best architecture based on the lowest validation RMSE.
4. Compute **RMSE** on training, validation, and test data.
5. Plot training loss vs. epochs for the best model.

Note

A. Simple Linear Regression

Import the LinearRegression from `sklearn.linear_model`.
A code snippet for prediction using linear regression:

```python
from sklearn.linear_model import LinearRegression

# Initialize linear regression model
regressor = LinearRegression()

```

```
# Fit the model using training data
# x is the set of univariate or multivariate training data
# y is the corresponding dependent variable
regressor.fit(x, y)

# Predict values
y_pred = regressor.predict(x)
```

B. Polynomial Curve Fitting

Import the PolynomialFeatures from `sklearn.preprocessing`.
A code snippet for prediction using polynomial regression:

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

# Initialize polynomial feature transformation
polynomial_features = PolynomialFeatures(degree=p)

# Transform the input features (x) into polynomial terms
x_poly = polynomial_features.fit_transform(x)

# x_poly contains polynomial expansions (monomials of polynomial up to degree p
# that will be used in a linear regression model

regressor = LinearRegression()
regressor.fit(x_poly, y)

# Predict values (notice we also transform x for predictions)
y_pred = regressor.predict(x_poly)
```

C. Neural Network Implementation

```
import torch
import torch.nn as nn
import torch.optim as optim

class NeuralNet(nn.Module):
 def __init__(self, input_size, hidden1, hidden2):
  super().__init__()
   self.layers = nn.Sequential(
      nn.Linear(input_size, hidden1),
      nn.Sigmoid(),
      nn.Linear(hidden1, hidden2),
      nn.Sigmoid(),
      nn.Linear(hidden2, 1)
)

 def forward(self, x):
    return self.layers(x)

```

```
model = NeuralNet(input_size=30, hidden1=128, hidden2=32)
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

References

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford, UK: Oxford University Press, 1995.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York, NY: Springer, 2006.

[3] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, *Extreme Learning Machine for Regression and Multiclass Classification*, 2012.