Department of Electrical and Computer Engineering
Pak-Austria Fachhochschule: Institute of Applied Sciences & Technology

a

Lab Manual
Lab-III

Subject: Machine Learning

Subject Teacher: Dr. Abid Ali

Lab Supervisor: Miss. Sana Saleem

Linear Regression Model Implementation and Its Output Predictions:

Objectives

- Implement linear regression to predict outcomes using a real-world dataset.
- Explore and preprocess datasets, including handling missing values and outliers.
- Apply data visualization techniques to understand data distributions and correlations.
- Train and evaluate a linear regression model using performance metrics.
- Use gradient descent to optimize model parameters and understand the cost function.
- Analyze model performance using metrics such as accuracy, precision, recall, and F1-score.
- Visualize the importance of features in the model and analyze training loss over time.

Pre-Requisite

- Read data from CSV files, clean missing values, and detect outliers using statistical techniques like Z-scores.
- Create visualizations such as histograms, correlation matrices, and scatter plots to explore dataset features.
- Implement linear regression using Python libraries and train-test splits, applying feature scaling techniques.
- Evaluate model performance using metrics like Mean Squared Error (MSE), R-squared, accuracy, precision, recall, and F1-score.
- Understand and implement gradient descent for optimizing linear regression parameters and visualize the cost function over epochs.
- Analyze feature importance using model coefficients and visualize their contribution to predictions.
- Develop and document a working machine learning solution for linear regression and understand the relevance of each step from data preprocessing to model evaluation.

Lab Practice 1: Implementation of Linear Regression Model with one variable

```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy import stats

# Load the Diabetes dataset
url = "diabetes[1].csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI',
           'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, names=columns)

# Display first few rows of the dataset
print(df.head())

# Basic statistics of the dataset
print(df.describe())

# Check for missing values
print(df.isnull().sum())

df = df.apply(pd.to_numeric, errors='coerce')
df = df.dropna()

# Outlier analysis using Z-score for the selected feature 'Glucose'
z_scores = np.abs(stats.zscore(df['Glucose']))
print("Outliers (Z > 3):")
print(np.where(z_scores > 3))

# Remove outliers based on Z-score for 'Glucose'
df_clean = df[(z_scores < 3)]
print(f"Shape after removing outliers: {df_clean.shape}")

# Splitting the dataset into the selected feature 'Glucose' and
target variable 'Outcome'
X = df_clean[['Glucose']]  # Single input feature
y = df_clean['Outcome']    # Output variable
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test data
y_pred = model.predict(X_test)

# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Visualizing model performance
plt.scatter(y_test, y_pred, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()

# Visualize residuals
residuals = y_test - y_pred
sns.histplot(residuals, bins=20, kde=True, color='orange')
plt.title('Residuals Distribution')
plt.show()
```

Lab Practice 2: Implementation of Linear Regression Model with Linear Regression Practices.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from scipy import stats

url = "diabetes.csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI',
           'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, names=columns)

print(df.head())
print(df.describe())
print(df.isnull().sum())
df = df.apply(pd.to_numeric, errors='coerce')
print(df.isnull().sum())
df = df.dropna()

df = df.apply(pd.to_numeric, errors='coerce')
df = df.dropna()

# Visualize the correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()

df.hist(bins=20, figsize=(14,10), color='blue')
plt.suptitle('Feature Distributions')
plt.show()

z_scores = np.abs(stats.zscore(df))
print("Outliers (Z > 3):")
print(np.where(z_scores > 3))
df_clean = df[(z_scores < 3).all(axis=1)]
print(f"Shape after removing outliers: {df_clean.shape}")
df_clean.hist(bins=20, figsize=(14,10), color='green')
plt.suptitle('Distributions After Removing Outliers')
plt.show()

# Splitting the dataset into features and target variable
X = df_clean.drop(columns='Outcome')
y = df_clean['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Feature scaling (Standardization)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Visualizing model performance
plt.scatter(y_test, y_pred, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()

# Visualize residuals
residuals = y_test - y_pred
sns.histplot(residuals, bins=20, kde=True, color='orange')
plt.title('Residuals Distribution')
plt.show()

# Training and Loss Visualization (Epochs Simulation)
epochs = 500
train_errors = []
for i in range(epochs):
    model.fit(X_train_scaled, y_train)
    y_train_pred = model.predict(X_train_scaled)
    error = mean_squared_error(y_train, y_train_pred)
    train_errors.append(error)

# Plot training loss
plt.plot(range(epochs), train_errors, color='blue')
plt.xlabel('Epochs')
plt.ylabel('Training Error (MSE)')
plt.title('Training Error Over Epochs')
plt.show()
features = X.columns
importance = model.coef_
plt.figure(figsize=(10,6))
plt.barh(features, importance, color='teal')
plt.title('Feature Importance in Linear Regression')
plt.xlabel('Coefficient Values')
plt.show()
```

Lab Practice 3: Checking for the Accuracy, Precision, Recall and F1 Score Values (We will discuss these in the next Lab)

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score,
accuracy_score, precision_score, recall_score, f1_score
from scipy import stats

url = "diabetes.csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI',
           'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, names=columns)
df = df.apply(pd.to_numeric, errors='coerce')
df = df.dropna()
z_scores = np.abs(stats.zscore(df))
df_clean = df[(z_scores < 3).all(axis=1)]
X = df_clean.drop(columns='Outcome')
y = df_clean['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)

y_pred_binary = [1 if pred >= 0.5 else 0 for pred in y_pred]
accuracy = accuracy_score(y_test, y_pred_binary)
precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
```

```
f1 = f1_score(y_test, y_pred_binary)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

plt.scatter(y_test, y_pred, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()
residuals = y_test - y_pred
sns.histplot(residuals, bins=20, kde=True, color='orange')
plt.title('Residuals Distribution')
plt.show()
```

Lab Practice: Add Cost and Gradient Descent Function in the Lab

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

url = "diabetes[1].csv"
columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI',
           'DiabetesPedigreeFunction', 'Age', 'Outcome']
df = pd.read_csv(url, names=columns)
print(df.head())
print(df.describe())
print(df.isnull().sum())
df = df.apply(pd.to_numeric, errors='coerce')
df = df.dropna()

# Remove outliers using Z-score
```

```
from scipy import stats
z_scores = np.abs(stats.zscore(df))
df_clean = df[(z_scores < 3).all(axis=1)]
print(f"Shape after removing outliers: {df_clean.shape}")

# Splitting the dataset into features and target variable
X = df_clean.drop(columns='Outcome')
y = df_clean['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
np.random.seed(42)
m, n = X_train_scaled.shape
theta = np.random.randn(n)  # Initial weights
bias = 0.0  # Initial bias
learning_rate = 0.01
epochs = 1000
cost_history = []

# Define the cost function (MSE)
def compute_cost(X, y, theta, bias):
    m = len(y)
    predictions = np.dot(X, theta) + bias
    cost = (1 / (2 * m)) * np.sum((predictions - y) ** 2)
    return cost

# Define gradient descent function
def gradient_descent(X, y, theta, bias, learning_rate, epochs):
    m = len(y)
    cost_history = []

    for epoch in range(epochs):
        # Make predictions
        predictions = np.dot(X, theta) + bias

        # Compute gradients
        d_theta = (1 / m) * np.dot(X.T, (predictions - y))
        d_bias = (1 / m) * np.sum(predictions - y)

        # Update weights
        theta -= learning_rate * d_theta
        bias -= learning_rate * d_bias
        # Calculate cost and save it for plotting
        cost = compute_cost(X, y, theta, bias)
        cost_history.append(cost)

        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Cost = {cost}")

    return theta, bias, cost_history

theta, bias, cost_history = gradient_descent(X_train_scaled, y_train,
theta, bias, learning_rate, epochs)
y_pred_train = np.dot(X_train_scaled, theta) + bias
y_pred_test = np.dot(X_test_scaled, theta) + bias
mse_test = mean_squared_error(y_test, y_pred_test)
print(f"Test MSE: {mse_test}")
plt.plot(range(epochs), cost_history, color='blue')
plt.xlabel('Epochs')
plt.ylabel('Cost (MSE)')
plt.title('Cost Over Epochs (Gradient Descent)')
plt.show()

# Visualize predictions vs actual values
plt.scatter(y_test, y_pred_test, color='purple')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],
color='red', linewidth=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Gradient Descent: Actual vs Predicted")
plt.show()
features = X.columns
importance = theta
plt.figure(figsize=(10,6))
plt.barh(features, importance, color='teal')
plt.title('Feature Importance After Gradient Descent')
plt.xlabel('Coefficient Values')
plt.show()
```

Lab Tasks:

Home Task need to be submitted through the Teams, add all screenshots and relevant results with explanation must be added from the system.

1. You are required to implement a linear regression model on an insurance dataset, following the procedures outlined in the lab manual. The goal is to analyze the dataset, implement the model, evaluate its performance, and visualize the results. You will need to document each step, providing insights and analysis based on the findings.
2. Dataset is attached with.

insurance[1].csv