Name: Navaneeth Talluri

Student ID: 23102357

Module Name: Machine Learning and Neural Networks

Subject Code: 7PAM2021-0105-2024

GitHub Repository:

https://github.com/NavaneethTalluri/Linear-Regression-.git

Linear Regression & Its Applications

1. Introduction

Linear regression is the simplest statistical technique used in machine learning and data analysis. It Pandeyishes a relationship between a dependent variable (target) and one or more independent variables (features). Due to its simplicity and interpretability, linear regression is widely used for predictive modeling across various domains (Sugiyama, 2015).

This document covers:

- Theoretical foundations of linear regression
- Python implementation using scikit-learn
- Real-world applications
- Model evaluation techniques

2. Fundamentals of Linear Regression

Basic Concept

Linear regression fits the best linear line (or hyperplane for multiple variables) between input and output variables. The mathematical representation is:

$$y=\beta0+\beta1x1+\beta2x2+\cdots+\beta nxn+\epsilon$$

Where:

yy: Target (dependent) variable

x1,x2,…,xn$x_1,x_2,\dots,x_n$: Features (independent variables)

β0β_0: Intercept term

β1,β2,…,βn$\beta_1,\beta_2,\dots,\beta_n$: Feature coefficients

ϵϵ: Error term (random noise)

Types of Linear Regression

Simple Linear Regression: Single independent variable

$$y=\beta 0+\beta 1x+\epsilon y=\beta_0+\beta_1 x+\epsilon$$

Multiple Linear Regression: Multiple independent variables

(Used Latex for equations)

Key Assumptions

- For reliable results, these assumptions must hold:
- Linearity: Linear relationship between features and target
- Independence: Observations are independent of each other
- Homoscedasticity: Constant residual variance across predictions
- Normality: Residuals are normally distributed

3. Applications

Real-World Uses

- Real Estate: Predicting house prices based on features like area and bedrooms
- Sales Forecasting: Estimating future sales from marketing spend
- Healthcare: Analyzing relationships like BMI vs. heart disease risk
- Finance: Predicting stock prices and assessing investment risks
- Engineering: Correlating product quality with manufacturing parameters

Limitations

Assumes linear relationships (real data often isn't linear)

Sensitive to outliers

Problems with multicollinearity (correlated features)

4. Python Implementation

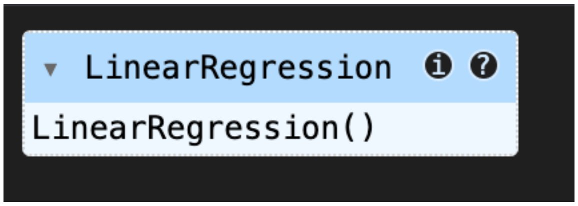
Step 1: Load Data

```
from sklearn.datasets import fetch_california_housing

import pandas as pd
```

```
# Load dataset

housing = fetch_california_housing()

X = housing.data

y = housing.target


# Convert to DataFrame

housing_df = pd.DataFrame(X, columns=housing.feature_names)

housing_df['Price'] = y

housing_df.head()
```

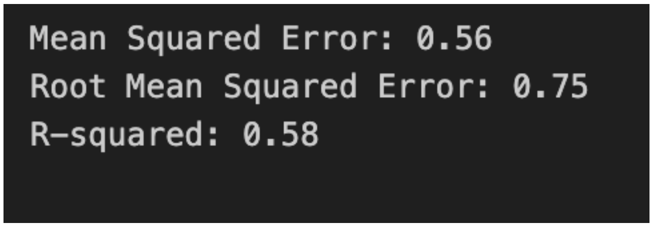
Step 2: Train Model

```
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model

model = LinearRegression()

model.fit(X_train, y_train)
```



```
LinearRegression()
```

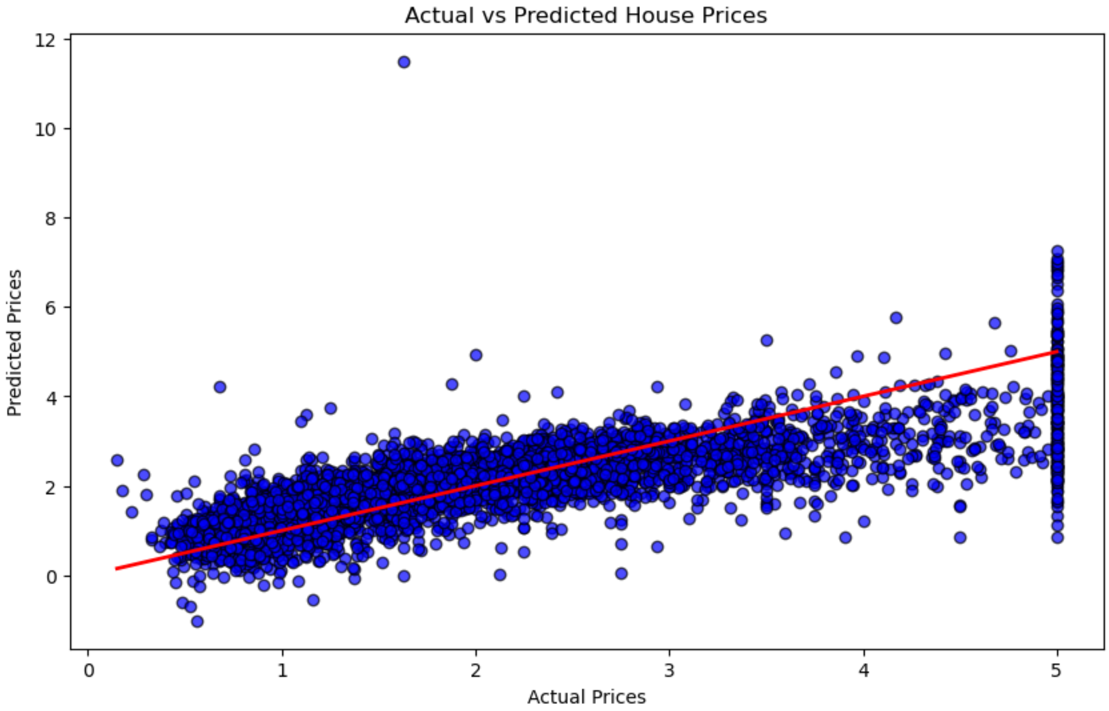
```
# Evaluate

y_pred = model.predict(X_test)

print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")

print(f"R²: {r2_score(y_test, y_pred):.2f}")
```



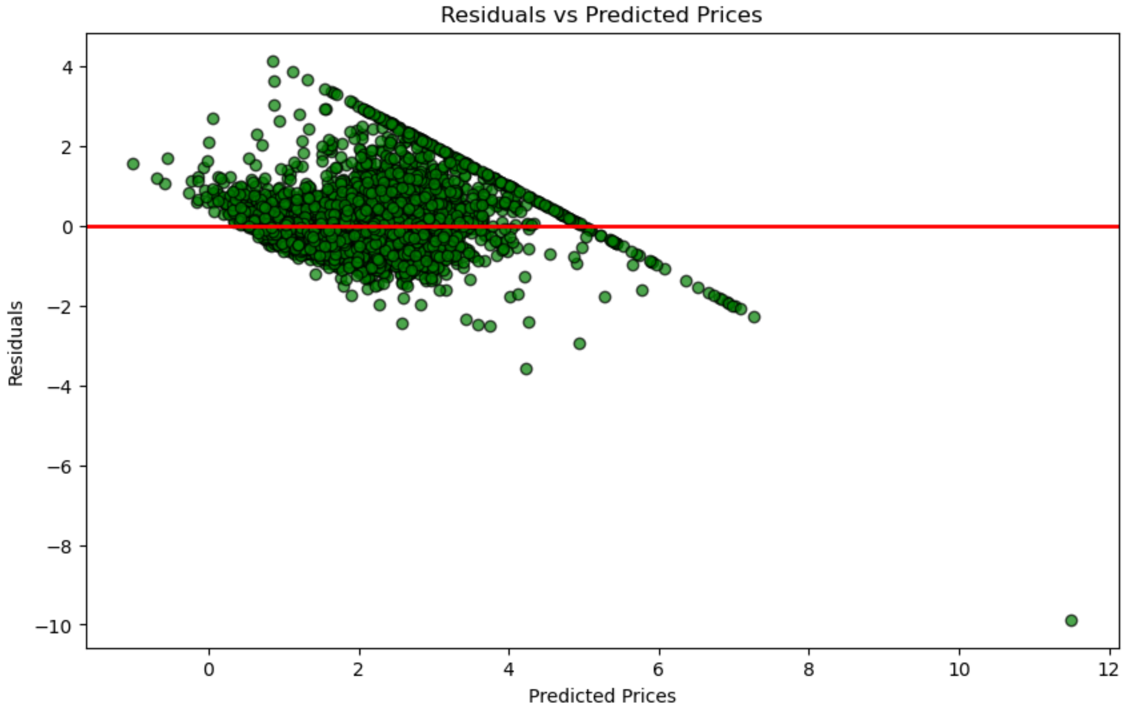
```
Mean Squared Error: 0.56
Root Mean Squared Error: 0.75
R-squared: 0.58
```

Step 3: Visualize Results

```
import matplotlib.pyplot as plt

plt.scatter(y_test, y_pred)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')

plt.xlabel('True Prices')

plt.ylabel('Predicted Prices')

plt.title('Price Predictions')

plt.show()
```



Visualize for Actual Vs Predicted House prices



Visualize for Residuals vs Predicted Prices

Coefficient:

Coefficient for MedInc: 0.45

Coefficient for HouseAge: 0.01

Coefficient for AveRooms: -0.12

Coefficient for AveBedrms: 0.78

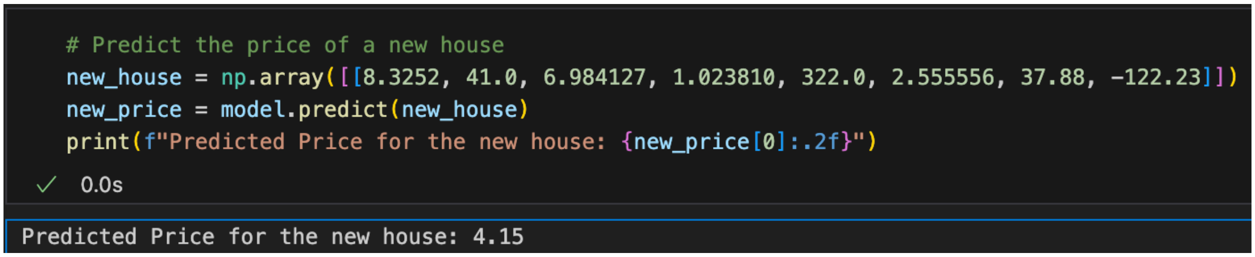
Coefficient for Population: -0.00

Coefficient for AveOccup: -0.00

Coefficient for Latitude: -0.42

Coefficient for Longitude: -0.43

Prediction Price for New House:



```
# Predict the price of a new house
new_house = np.array([[8.3252, 41.0, 6.984127, 1.023810, 322.0, 2.555556, 37.88, -122.23]])
new_price = model.predict(new_house)
print(f"Predicted Price for the new house: {new_price[0]:.2f}")
```

✓ 0.0s

```
Predicted Price for the new house: 4.15
```

5. Evaluation & Conclusion

Key Metrics

Mean Squared Error (MSE): Average squared prediction errors

R-squared: Proportion of variance explained by model

Conclusion

Linear regression remains a fundamental tool for predictive modeling across industries. While powerful, practitioners must verify assumptions and preprocess data carefully to ensure reliable results (Ahmad & Pandey, 2025).

6. References

1. Ahmad, Absar & Pandey, Manjari. (2025). Application of Linear and Non-Linear Regression in Research.
2. Scikit-learn documentation: https://scikit-learn.org Retrieved on March 22, 2025.
3. Sugiyama, M. (2015). Introduction to Statistical Machine Learning.
4. California Housing Dataset: UCI Machine Learning Repository Retrieved on March 24, 2025.

Appendix:

```
import numpy as np
import pandas as pd
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 sklearn.datasets import fetch_california_housing

# Step 2: Load and prepare the dataset
# Use California housing dataset as Boston dataset is outdated
california = fetch_california_housing()

# Convert dataset to Pandas DataFrame for simplicity
df = pd.DataFrame(california.data, columns=california.feature_names)
df['PRICE'] = california.target # Target column (house prices)

# Step 3: Split the dataset into features and target variable
X = df.drop('PRICE', axis=1) # Features
y = df['PRICE'] # Target variable

# Step 4: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Initialize and train the linear regression model
```

```
model = LinearRegression()
model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = model.predict(X_test)

# Step 7: Model evaluation
mse = mean_squared_error(y_test, y_pred) # Mean Squared Error
rmse = np.sqrt(mse) # Root Mean Squared Error
r2 = r2_score(y_test, y_pred) # R-squared score

# Print the evaluation metrics
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R-squared: {r2:.2f}")

# Step 8: Visualize the results
# Plot the actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', edgecolors='k', alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)
plt.title('Actual vs Predicted House Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.show()

# Step 9: Visualize the residuals
residuals = y_test - y_pred

plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='green', edgecolors='k', alpha=0.7)
plt.axhline(y=0, color='red', linewidth=2)
plt.title('Residuals vs Predicted Prices')
plt.xlabel('Predicted Prices')
plt.ylabel('Residuals')
plt.show()

# Step 10: Interpretation of the coefficients
coefficients = model.coef_
feature_names = X.columns

# Print the coefficients
for feature, coef in zip(feature_names, coefficients):
print(f"Coefficient for {feature}: {coef:.2f}")

# Step 11: Predict the price of a new house (example)
new_house = np.array([[8.3252, 41.0, 6.984127, 1.023810, 322.0, 2.555556, 37.88, -122.23]])
new_price = model.predict(new_house)
print(f"Predicted Price for the new house: {new_price[0]:.2f}")
```