**Linear Regression Overview**

**Linear Regression** is a statistical method used for modeling the relationship between a dependent variable and one or more independent variables. It's one of the simplest and most widely used regression techniques.

**1. Theory and Concepts**

**1.1. Introduction to Linear Regression**

* **Simple Linear Regression**: This involves a single independent variable and models the relationship as a straight line.

**Mathematical Form**:

y=β0+β1x+ϵy = \beta\_0 + \beta\_1 x + \epsilony=β0​+β1​x+ϵ

Where:

* + yyy is the dependent variable.
  + β0\beta\_0β0​ is the intercept.
  + β1\beta\_1β1​ is the slope of the line.
  + xxx is the independent variable.
  + ϵ\epsilonϵ is the error term.

**Example**: Suppose you want to predict the price of a house based on its size. You would use size as the independent variable and price as the dependent variable.

* **Multiple Linear Regression**: This involves more than one independent variable. The relationship is modeled as a hyperplane.

**Mathematical Form**:

y=β0+β1x1+β2x2+⋯+βpxp+ϵy = \beta\_0 + \beta\_1 x\_1 + \beta\_2 x\_2 + \cdots + \beta\_p x\_p + \epsilony=β0​+β1​x1​+β2​x2​+⋯+βp​xp​+ϵ

Where:

* + x1,x2,…,xpx\_1, x\_2, \ldots, x\_px1​,x2​,…,xp​ are independent variables.
  + β1,β2,…,βp\beta\_1, \beta\_2, \ldots, \beta\_pβ1​,β2​,…,βp​ are their coefficients.

**Example**: Predicting house prices based on size, location, number of bedrooms, etc.

**1.2. Assumptions of Linear Regression**

* **Linearity**: The relationship between the dependent and independent variables is linear.
* **Independence**: The residuals (errors) are independent of each other.
* **Homoscedasticity**: The residuals have constant variance across all levels of the independent variables.
* **Normality**: The residuals are normally distributed.

**1.3. Interpretation of Coefficients**

* **Understanding Model Coefficients**: Each coefficient represents the change in the dependent variable for a one-unit change in the corresponding independent variable, holding other variables constant.
* **Interpreting Statistical Significance**: Statistical significance is typically assessed using p-values. A coefficient is considered statistically significant if its p-value is below a certain threshold (e.g., 0.05).

**2. Implementation with Scikit-learn**

**2.1. Fitting a Linear Regression Model**

* **Preparing Data**: Split your data into features (X) and target (y). Optionally, you can also split the data into training and testing sets.

**Example**:

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import pandas as pd

# Load data

data = pd.read\_csv('data.csv')

X = data[['feature1', 'feature2']] # Independent variables

y = data['target'] # Dependent variable

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

* **Training and Evaluating the Model**:

**Example**:

model = LinearRegression()

model.fit(X\_train, y\_train) # Train the model

# Predict and evaluate

y\_pred = model.predict(X\_test)

**2.2. Model Evaluation Metrics**

* **R-squared**: Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.

**Example**:

from sklearn.metrics import r2\_score

r2 = r2\_score(y\_test, y\_pred)

print(f'R-squared: {r2}')

* **Adjusted R-squared**: Adjusted for the number of predictors in the model. It provides a more accurate measure when comparing models with different numbers of predictors.

**Example**:

n = len(y\_test) # Number of observations

p = X\_test.shape[1] # Number of predictors

adj\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

print(f'Adjusted R-squared: {adj\_r2}')

* **Mean Squared Error (MSE)**: Measures the average of the squares of the errors—i.e., the average squared difference between predicted and actual values.

**Example**:

from sklearn.metrics import mean\_squared\_error

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

* **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in a set of predictions, without considering their direction.

**Example**:

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f'Mean Absolute Error: {mae}')

**2.3. Handling Multicollinearity**

* **Detecting Multicollinearity**: Use metrics like Variance Inflation Factor (VIF) to assess multicollinearity.

**Example**:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

import numpy as np

X = np.c\_[np.ones(X\_train.shape[0]), X\_train] # Add constant term for VIF

vif = [variance\_inflation\_factor(X, i) for i in range(X.shape[1])]

print(f'VIF: {vif}')

* **Techniques to Address Multicollinearity**:
  + **Remove highly correlated predictors**.
  + **Use regularization techniques** like Ridge or Lasso regression.

**3. Model Evaluation**

**3.1. Residual Analysis**

* **Plotting Residuals**: Residual plots can help diagnose issues with the model.

**Example**:

residuals = y\_test - y\_pred

plt.scatter(y\_pred, residuals)

plt.xlabel('Predicted values')

plt.ylabel('Residuals')

plt.title('Residual Plot')

plt.show()

* **Analyzing Residual Patterns**: Look for patterns in residuals to check for homoscedasticity and other issues.

**3.2. Cross-Validation**

* **K-Fold Cross-Validation**: Splits the data into K folds, trains the model on K-1 folds, and tests it on the remaining fold. Repeats K times.

**Example**:

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(model, X, y, cv=5, scoring='r2')

print(f'Cross-Validation Scores: {scores}')

print(f'Mean Cross-Validation Score: {scores.mean()}')

* **Leave-One-Out Cross-Validation (LOOCV)**: A special case of K-Fold where K equals the number of observations.

**Example**:

from sklearn.model\_selection import LeaveOneOut

loo = LeaveOneOut()

scores = cross\_val\_score(model, X, y, cv=loo, scoring='r2')

print(f'LOOCV Scores: {scores}')

print(f'Mean LOOCV Score: {scores.mean()}')

**3.3. Regularization Techniques**

* **Ridge Regression**: Adds a penalty proportional to the square of the magnitude of coefficients.

**Example**:

from sklearn.linear\_model import Ridge

ridge = Ridge(alpha=1.0)

ridge.fit(X\_train, y\_train)

y\_pred\_ridge = ridge.predict(X\_test)

* **Lasso Regression**: Adds a penalty proportional to the absolute value of the coefficients.

**Example**:

from sklearn.linear\_model import Lasso

lasso = Lasso(alpha=0.1)

lasso.fit(X\_train, y\_train)

y\_pred\_lasso = lasso.predict(X\_test)

**Understanding Ridge and Lasso**:

* **Ridge Regression**: Useful when you have many predictors; it helps to prevent overfitting by shrinking the coefficients.
* **Lasso Regression**: Performs variable selection by shrinking some coefficients to zero, effectively selecting a subset of predictors.

**Implementing Regularization with Scikit-learn**:

from sklearn.linear\_model import Ridge, Lasso

from sklearn.metrics import mean\_squared\_error

# Ridge Regression

ridge = Ridge(alpha=1.0)

ridge.fit(X\_train, y\_train)

ridge\_pred = ridge.predict(X\_test)

print(f'Ridge MSE: {mean\_squared\_error(y\_test, ridge\_pred)}')

# Lasso Regression

lasso = Lasso(alpha=0.1)

lasso.fit(X\_train, y\_train)

lasso\_pred = lasso.predict(X\_test)

print(f'Lasso MSE: {mean\_squared\_error(y\_test, lasso\_pred)}')

**Summary**

* **Linear Regression Basics**: Includes simple and multiple linear regression, assumptions, and coefficient interpretation.
* **Scikit-learn Implementation**: Covers model fitting, evaluation metrics, handling multicollinearity.
* **Model Evaluation**: Includes residual analysis, cross-validation, and regularization techniques like Ridge and Lasso regression.