Regression and Its Types

Types of Regression:

Linear Regression, Regression Trees, Non-Linear Regression, and Polynomial Regression are different types and extensions of regression techniques used for predicting continuous values in supervised machine learning. Let's study all these types in detail.

- **Linear Regression**
- **Regression Trees**
- **Non-Linear Regression**
- **Polynomial Regression**

- **Linear Regression:** Linear regression is a fundamental statistical technique used for modeling the relationship between a dependent variable and one or more independent variables. It is widely employed in various fields, including economics, finance, biology, and machine learning. Let us learn more about them.

  a. **Theoretical Foundations**

  Linear regression assumes a linear relationship between the independent variables X and the dependent variable Y. The model can be represented as:

  $Y=\beta 0+\beta 1X1+\beta 2X2+\ldots+\beta nXn+\varepsilon$

  Here:

  - Y is the dependent variable.
  - $\beta 0$ is the y-intercept.
  - $\beta 1,\beta 2,\ldots,\beta n$ are the coefficients of the independent variables $X1,X2,\ldots,Xn$.
  - $X1,X2,\ldots,Xn$ are the independent variables.
  - ε is the error term.

  The goal of linear regression is to find the values of $\beta 0,\beta 1,\ldots,\beta n$ that minimize the sum of squared errors (SSE), represented as:

  $$SSE=\sum_{i=1}^{N}(Y_i-(\beta_0+\beta_1X_{i1}+\beta_2X_{i2}+\ldots+\beta_nX_{in}))^2$$

  Where, N is the number of observations.

b. **Intuition behind Linear Regression**

The intuition behind linear regression lies in fitting a line to the data points that minimizes the overall distance between the observed values and the values predicted by the model. The coefficients ($\beta 0, \beta 1, ..., \beta n$) determine the slope and intercept of this line.

The error term (ε) accounts for the variability in the data that the model cannot explain. The goal is to minimize this error by adjusting the coefficients during the model training process.

c. **Mathematical Equations**

The coefficients $\beta 0, \beta 1, ..., \beta n$ can be calculated using the following equations:

$$\beta_1 = \frac{\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{N}(X_i - \bar{X})^2}$$

$$\beta_0 = \bar{Y} - \beta_1 \bar{X}$$

where $\bar{X}$ and $\bar{Y}$ are the mean values of the independent and dependent variables, respectively.

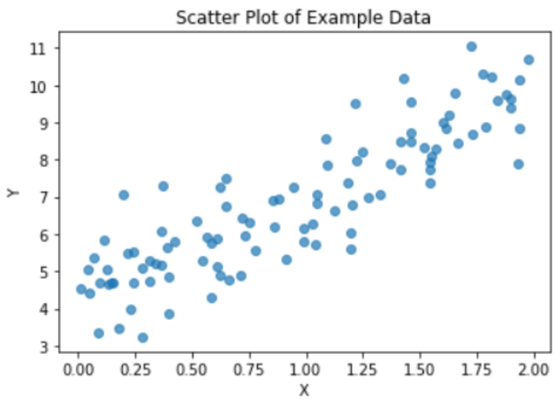
d. **Examples and Code**

Let's consider a simple example using Python to demonstrate linear regression. We'll use the popular **scikit-learn** library for this purpose.

```
# Import necessary libraries
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generate example data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
Y = 4 + 3 * X + np.random.randn(100, 1)

# Visualize the data
plt.scatter(X, Y, alpha=0.7)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Scatter Plot of Example Data')
plt.show()

# Fit the linear regression model
```

```
model = LinearRegression()
model.fit(X, Y)

# Print the coefficients
print(f'Intercept (beta0): {model.intercept_[0]}')
print(f'Coefficient (beta1): {model.coef_[0][0]}')

# Predict new values
new_X = np.array([[1.5]])
predicted_Y = model.predict(new_X)
  print(f'Predicted Y for X=1.5: {predicted_Y[0][0]}')
```

Output:



```
Intercept (beta0): 4.215096157546747
Coefficient (beta1): 2.7701133864384833
Predicted Y for X=1.5: 8.370266237204472
```

This Python script creates and demonstrates a simple linear regression model using scikit-learn.

This Python script employs scikit-learn to showcase a simple linear regression model. It begins by importing necessary libraries, including NumPy for numerical operations, scikit-learn's LinearRegression module for modeling, and Matplotlib for data visualization. The script then generates example data, creating random X values and calculating corresponding Y values based on a linear relationship with added noise. The data is visualized through a scatter plot to illustrate the relationship between X and Y.

Following this, the script proceeds to fit a linear regression model to the generated data. The model's intercept and coefficient are printed to the console, providing insight into the fitted line's equation.
To demonstrate the model's predictive capabilities, a new X value (1.5) is chosen, and the corresponding Y value is predicted using the trained model. This prediction is then printed to the console.

- **Regression trees:** Regression trees are powerful predictive models that can capture complex relationships in data. Let us learn more about them. At its core, a regression tree is a decision tree designed for predicting continuous outcomes. The tree structure comprises nodes representing decisions, branches denoting possible outcomes, and leaves holding predicted values. The process of constructing a regression tree involves recursively partitioning the data based on features to minimize the variance of the target variable within each partition.

a. **Mathematical Equation**

The recursive partitioning in regression trees is guided by impurity measures, commonly mean squared error (MSE) for regression tasks. Given a node *m* with observations *Dm*, the MSE is calculated as:

$$\mathrm{MSE}(D_m) = \frac{1}{|D_m|} \sum_{i \in D_m} (y_i - \bar{y}_m)^2$$

Here, *yi* represents the target variable for observation *i* in node *m*, *y¯m* is the mean of the target variable in node *m*. The goal is to select the feature and split point that minimize the weighted sum of MSE for the child nodes.

b. **Intuition Behind Regression Trees**

The allure of regression trees lies in their simplicity and interpretability. They mimic human decision-making by breaking down complex problems into a series of **binary** choices. At each node, the tree selects the feature that best splits the data, with the goal of minimizing the variance of the target variable within each branch. This recursive process continues until a stopping criterion is met, resulting in a tree structure that maps features to predicted values.

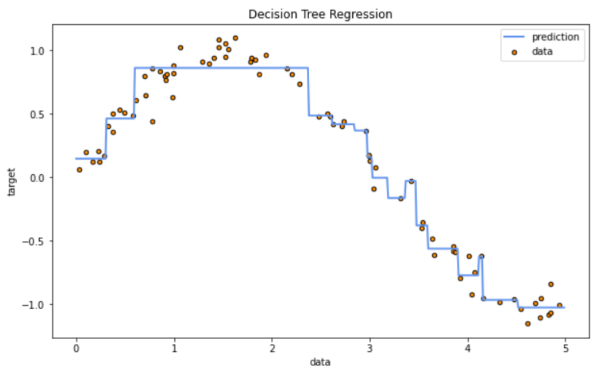
c. **Example Code:**

Let's consider a simple example where we want to predict the price of houses based on features like square footage and the number of bedrooms. The decision tree would make binary splits based on these features, creating a hierarchical structure. For instance, the first split might be based on square footage, with subsequent splits based on bedrooms, creating a tree that segments the data into subsets with similar target variable values.

```
# Import necessary libraries
from sklearn.tree import DecisionTreeRegressor
import numpy as np
import matplotlib.pyplot as plt

# Generate example data
np.random.seed(42)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + np.random.normal(0, 0.1,
X.shape[0])

# Fit regression tree
reg_tree = DecisionTreeRegressor(max_depth=4)
reg_tree.fit(X, y)

# Predict for new data
X_new = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_pred = reg_tree.predict(X_new)

# Plot the results
plt.figure(figsize=(10, 6))
plt.scatter(X, y, s=20, edgecolor="black",
c="darkorange", label="data")
plt.plot(X_new, y_pred, color="cornflowerblue",
label="prediction", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

Output:

In this example, we generate synthetic data with a sinusoidal relationship and fit a regression tree to predict the target variable. The resulting plot shows how the tree partitions the feature space to make predictions.



This Python code utilizes scikit-learn to implement decision tree regression for a one-dimensional dataset. After importing the necessary libraries, synthetic data is generated, and a decision tree regressor with a maximum depth of 4 is trained on the data. The code then predicts target values for a new set of inputs and visualizes the results by plotting the original data points and the decision tree's predictions. The x-axis represents the input feature, the y-axis signifies the target variable, and the plot is titled "Decision Tree Regression." This concise script offers a clear visual representation of the model's performance in capturing underlying patterns in the data.

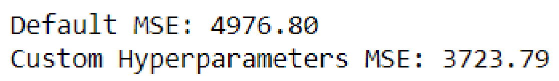
Hyperparameter Tuning

Regression trees come with hyperparameters that influence their structure, such as the maximum depth of the tree, the minimum number of samples required to split a node, and the minimum number of samples required to be at a leaf node. Tuning these hyperparameters is crucial to prevent overfitting or underfitting the model.

```
# Import necessary libraries
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

# Load the Diabetes dataset
diabetes = load_diabetes()
X = diabetes.data
y = diabetes.target

# Split the dataset 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)

# Example 1: Default hyperparameters
dt_default = DecisionTreeRegressor(random_state=42)
dt_default.fit(X_train, y_train)
y_pred_default = dt_default.predict(X_test)
```

```
mse_default = mean_squared_error(y_test,
y_pred_default)
print(f"Default MSE: {mse_default:.2f}")

# Example 2: Using different hyperparameters
dt_custom = DecisionTreeRegressor(
    max_depth=5,           # Maximum depth of the
tree
    min_samples_split=5,   # To split an internal
node
    min_samples_leaf=2,    # To be at a leaf node
    random_state=42
)
dt_custom.fit(X_train, y_train)
y_pred_custom = dt_custom.predict(X_test)
mse_custom = mean_squared_error(y_test,
y_pred_custom)
print(f"Custom Hyperparameters MSE:
{mse_custom:.2f}")
```

Output:



```
Default MSE: 4976.80
Custom Hyperparameters MSE: 3723.79
```

d. **Advantages and Challenges**

Advantages of Regression Tress:

- **Non-linearity:** Regression trees can model non-linear relationships in data, making them versatile for various datasets.
- **Interpretability:** The hierarchical structure of decision trees provides a clear and interpretable decision-making process, making them valuable for understanding the factors influencing predictions.
- **Robustness:** Regression trees are robust to outliers, as they make decisions based on binary splits rather than relying on the magnitude of the data points.

Challenges and Mitigations

- **Overfitting:** Regression trees can become overly complex and capture noise in the data, leading to overfitting. Pruning techniques, limiting tree depth, or imposing constraints on leaf nodes can mitigate this issue.
- **Instability:** Small changes in the data can result in different tree structures. Ensemble methods like Random Forests aggregate multiple trees to improve stability and generalization.

  Regression trees find applications in various domains. In finance, they can predict stock prices based on market indicators. In healthcare, they may be used to estimate patient recovery times based on medical history. Understanding the context of your data is crucial for effective application.

- **Non-Linear Regression:** Non-linear regression is employed when the relationship between the independent and dependent variables cannot be adequately captured by a linear model. While linear regression assumes a straight-line relationship, non-linear regression accommodates curves, exponential growth, and other intricate patterns in the data. This flexibility makes it suitable for a wide array of real-world scenarios.

  a. Theoretical Foundations

  In linear regression, the model is expressed as:

  $Y=\beta 0+\beta 1X1+\beta 2X2+\ldots+\beta nXn+\varepsilon$

  Non-linear regression extends this framework by allowing for more complex functional forms. The model can take the form:

  $Y=f(X1,X2,\ldots,Xn;\beta 0,\beta 1,\ldots,\beta k)+\varepsilon$

  Here, *f* represents a non-linear function of the independent variables and coefficients.

b. Intuition Behind Non-Linear Regression

The intuition behind non-linear regression lies in its ability to capture intricate patterns and trends in the data that cannot be adequately represented by a straight line. By introducing non-linear functions, the model can flexibly adapt to a wide range of relationships, including exponential growth, logarithmic decay, and more.

c. Mathematical Equations

Non-linear regression introduces a non-linear function into the model. This function can take various forms, such as polynomials, exponentials, logarithms, or combinations thereof. A common representation is:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \varepsilon$$

Additionally, non-linear regression can involve more complex functions such as:

$$Y = \beta_0 + \beta_1 \cdot e^{\beta_2 X} + \beta_3 \cdot \log(X) + \varepsilon$$

The choice of the non-linear function depends on the underlying patterns in the data.

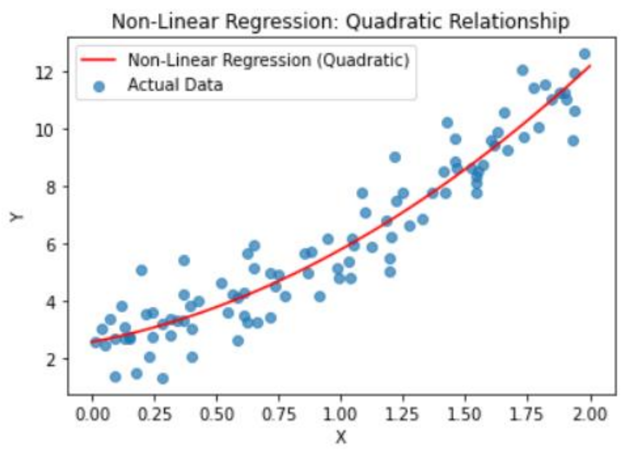
d. Python Implementation with Scikit-Learn

Non-linear regression can be implemented using the **scikit-learn** library in Python. Let's consider fitting a non-linear regression model to a dataset with a quadratic relationship.

```
# Import necessary libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
import matplotlib.pyplot as plt

# Generate example data for a quadratic relationship
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 2 + 3 * X + np.random.randn(100, 1) + X**2
```

```
# Transform features to include quadratic term
poly_features = PolynomialFeatures(degree=2,
include_bias=False)
X_poly = poly_features.fit_transform(X)

# Fit linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)

# Visualize the results
X_new = np.linspace(0, 2, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)

plt.scatter(X, y, alpha=0.7, label='Actual Data')
plt.plot(X_new, y_new, color='red', label='Non-Linear
Regression (Quadratic)')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Non-Linear Regression: Quadratic
Relationship')
plt.legend()
plt.show()
```

This code demonstrates non-linear regression with a quadratic relationship using scikit-learn. It starts by generating a dataset with a quadratic pattern, including random noise. The features are then transformed with PolynomialFeatures, introducing a quadratic term. A LinearRegression model is applied to fit the transformed features to the target variable. The results are visualized by predicting new data points and plotting both the actual data and the non-linear regression curve, showcasing the quadratic relationship.



e. Advantages and Challenges

Advantages:

- **Increased Flexibility:** Non-linear regression allows for modeling a broader range of relationships compared to linear regression.
- **Improved Fit:** With the ability to capture complex patterns, non-linear regression often provides a better fit to the data.
- **Enhanced Predictive Power:** Non-linear regression can yield more accurate predictions when the relationship between variables is non-linear.

Challenges and Considerations

- **Overfitting:** Non-linear regression models can become overly complex and overfit the training data. Regularization techniques and proper model validation can help mitigate this risk.
- **Model Selection:** Choosing the appropriate non-linear function is crucial and often requires domain knowledge or experimentation.

  Non-linear regression finds applications in diverse fields. In finance, it can model the complex relationship between economic variables. In environmental science, it may describe the growth of ecosystems over time. Identifying the non-linear patterns in your data is essential for successful application.

- **Polynomial Regression:** **Modeling Complex Patterns:** Polynomial regression is a form of non-linear regression that extends the linear regression model by incorporating polynomial terms, enabling the modeling of more complex relationships between variables.

a. Polynomial Regression Equation

Polynomial regression involves fitting a polynomial equation to the data instead of a linear one. The equation takes the form:

$Y=\beta 0+\beta 1X+\beta 2X2+\ldots+\beta nXn+\varepsilon$

Here, *n* is the degree of the polynomial, determining the complexity of the relationship captured by the model. The degree of the polynomial is a critical parameter in polynomial regression. A higher degree allows the model to capture more intricate patterns but also increases the risk of overfitting.

b. Theoretical Foundations

The theoretical foundation of polynomial regression lies in its ability to capture non-linear patterns in the data by introducing polynomial terms. The key concept is that the relationship between the independent and dependent variables is modeled as a polynomial function of the form

$f(X)=\beta 0+\beta 1X+\beta 2X2+\ldots+\beta nXn$.

c. Intuition Behind Polynomial Regression

The intuition behind polynomial regression is straightforward. While linear regression fits a straight line to the data, polynomial regression fits a curve. This curve can adapt to the data's curvature, capturing more intricate relationships that a linear model might overlook. The degree of the polynomial determines the flexibility of the curve.

d. Mathematical Equations

The equation for a polynomial regression model of degree *n* is:

$Y=\beta 0+\beta 1X+\beta 2X2+\ldots+\beta nXn+\varepsilon$

The coefficients $\beta 0,\beta 1,\ldots,\beta n$ are estimated during the model training process, and ε represents the error term.

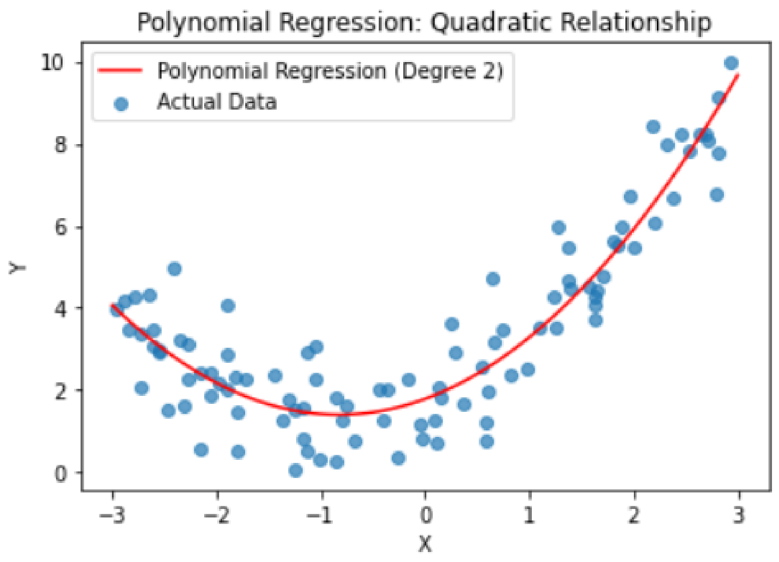
e. Practical Examples

Let's consider a practical example using Python to implement polynomial regression. In this example, we'll use the **numpy** and **matplotlib** libraries to generate synthetic data and visualize the polynomial regression model.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Generate example data
np.random.seed(42)
```

```
X = 6 * np.random.rand(100, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1)

# Transform features to include polynomial terms up to
degree 2
poly_features = PolynomialFeatures(degree=2,
include_bias=False)
X_poly = poly_features.fit_transform(X)

# Fit linear regression model
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)

# Visualize the results
X_new = np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)

plt.scatter(X, y, alpha=0.7, label='Actual Data')
plt.plot(X_new, y_new, color='red', label='Polynomial
Regression (Degree 2)')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polynomial Regression: Quadratic Relationship')
plt.legend()
plt.show()
```

Output:



In this example, the synthetic data represents a quadratic relationship, and the polynomial regression model of degree 2 effectively captures the curvature.This Python code employs scikit-learn to implement polynomial regression and visualize the results with Matplotlib. After generating example data with a quadratic relationship, it transforms the features to include polynomial terms up to degree 2. A linear regression model is then trained on the transformed features. The code generates new data points for visualization, transforms them, and predicts output values. The plot showcases the actual data points and the polynomial regression curve, providing a visual representation of the quadratic relationship captured by the model.

f. Advantages and Challenges

Advantages:

- **Flexibility:** Polynomial regression can model a wide range of non-linear relationships, offering more flexibility than linear regression.
- **Increased Accuracy:** By capturing the curvature in the data, polynomial regression often provides more accurate predictions for non-linear relationships.

Challenges and Considerations:

- **Overfitting:** High-degree polynomials can lead to overfitting, where the model fits the noise in the data rather than the underlying pattern. Regularization techniques and model validation are essential for mitigating overfitting.
- **Interpretability:** As the degree of the polynomial increases, the interpretability of the model diminishes. Understanding the practical implications of high-degree polynomials is crucial.

Polynomial regression finds applications in various fields. In physics, it can model the trajectory of a projectile. In economics, it may describe the relationship between investment and economic growth. Identifying the appropriate degree of the polynomial is vital for meaningful application.