Understanding Linear and Polynomial Regression in Python

Introduction

In this document, we will explore two fundamental regression techniques: **Linear Regression** and **Polynomial Regression**. These are powerful tools used in data science and machine learning to model relationships between variables and make predictions. We'll use the **Diabetes** dataset from `sklearn.datasets` to demonstrate these models in Python.

What is Linear Regression?

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The goal is to find the best-fitting straight line through the data points that can be used for prediction.

In simple linear regression, the model is defined as:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:

- y is the dependent variable
- x is the independent variable
- β_0 is the intercept
- β_1 is the slope (coefficient)
- ϵ is the error term

What is Polynomial Regression?

Polynomial regression is an extension of linear regression where the relationship between the independent variable and the dependent variable is modeled as an nth-degree polynomial. It is useful when the data shows a curved trend that a straight line cannot capture.

The general form of a polynomial regression equation is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

This allows the model to fit more complex, non-linear patterns in the data while still using a linear model under the hood through feature transformation.

1. Loading the Dataset and Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

Explanation:

- `pandas`, `numpy`, and `matplotlib.pyplot` are standard libraries for data manipulation and visualization.
- `load_diabetes` provides a sample dataset used for regression tasks.
- `train_test_split` is used to divide our data into training and testing sets.
- `LinearRegression` creates our linear model.
- `PolynomialFeatures` and `make_pipeline` help us build a polynomial regression model.

2. Preparing the Dataset

```
diabetes = load_diabetes()
data = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
data['target'] = diabetes.target
```

Explanation:

We load the diabetes dataset into a DataFrame called `data` and add the target variable (a measure of disease progression) as a new column.

3. Selecting a Feature

```
X = data[['bmi']]
y = data['target']
```

Explanation:

We focus on a single feature: **BMI** (Body Mass Index). `X` represents the input (independent variable), and `y` represents the target (dependent variable).

4. Splitting the Data

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

Explanation:

We split the dataset into training (80%) and testing (20%) sets. This ensures that our model is trained on one portion and evaluated on another to avoid overfitting.

5. Building a Linear Regression Model

```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_predictions = lr.predict(X_test)
```

Explanation:

We create and train a linear regression model using `LinearRegression`. After fitting it on training data, we use it to predict the test data.

6. Evaluating the Linear Model

```
r2 = r2_score(y_test, y_predictions)
mse = mean_squared_error(y_test, y_predictions)
mae = mean_absolute_error(y_test, y_predictions)
intercept = lr.intercept_

print("The R2 score is:", r2)
print("The Mean Squared Error is:", mse)
print("The Mean Absolute Error is:", mae)
print("The Intercept is:", intercept)
```

Explanation:

- **R2 Score** measures how well the model explains the variation in the data.
- **MSE** and **MAE** are error metrics. Lower values indicate better model performance.

7. Visualizing Linear Regression

```
plt.scatter(X, y)
plt.xlabel('Persons BMI')
```

```
plt.ylabel('Persons Diabetes level')
plt.show()
```

Explanation:

This scatter plot helps visualize the relationship between BMI and diabetes level.

8. Building a Polynomial Regression Model

```
poly_model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
poly_model.fit(X_train, y_train)
poly_y_predictions = poly_model.predict(X_test)
```

Explanation:

Polynomial regression models non-linear relationships. `PolynomialFeatures` transforms the input feature to include powers of the feature (e.g., BMI^2).

9. Evaluating the Polynomial Model

```
pol_r2 = r2_score(y_test, poly_y_predictions)
pol_mse = mean_squared_error(y_test, poly_y_predictions)
pol_mae = mean_absolute_error(y_test, poly_y_predictions)

print("The R2 score is:", pol_r2)
print("The Mean Squared Error is:", pol_mse)
print("The Mean Absolute Error is:", pol_mae)
```

Explanation:

We use the same evaluation metrics to assess how well the polynomial model performs in comparison to the linear model.

Summary

- **Linear regression** fits a straight line to the data.
- **Polynomial regression** allows for a curved line, better capturing non-linear trends.
- Model performance is compared using R2, MSE, and MAE.
- Visualizing and evaluating models helps choose the right approach for your dataset.

Both regression techniques are crucial tools in the data scientist's toolkit, and Python's `scikit-learn` makes implementing them straightforward and efficient.