Linear Regression

Types of Linear Regression

1. Simple Linear Regression: Involves one independent variable.
2. Multiple Linear Regression: Involves more than one independent variable.

Assumptions of Linear Regression

1. **Linearity**: The relationship between the independent and dependent variables is linear.
2. **Independence**: Observations are independent of each other.
3. **Homoscedasticity**: Constant variance of errors.
4. **Normality**: The residuals (errors) are normally distributed.
5. **No multicollinearity**: Independent variables are not highly correlated.

Simple Linear Regression

Model

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

- y: Dependent variable
- x: Independent variable
- β_0: Intercept
- β_1: Slope
- ϵ: Error term

Estimating Coefficients

- **Ordinary Least Squares (OLS)** method is used to estimate β0 and β1.
- The goal is to minimize the sum of the squared residuals:

$$\text{RSS} = \sum (y_i - (\beta_0 + \beta_1 x_i))^2$$

Interpretation of Coefficients

- **Intercept (β0)**: The value of y when x=0x = 0x=0.
- **Slope (β1)**: The change in y for a one-unit change in xxx.

Goodness of Fit

- **R-squared (R^2)**: Proportion of the variance in the dependent variable that is predictable from the independent variable(s).

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

Code for Simple Linear Regression

Using statsmodel

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Generating sample data
np.random.seed(0)
X = np.random.rand(100)
y = 2 * X + np.random.normal(0, 0.1, 100)

# Adding a constant (intercept) to the model
X = sm.add_constant(X)

# Fitting the model
model = sm.OLS(y, X).fit()

# Model summary
print(model.summary())
```

```
# Plotting the results
plt.scatter(X[:, 1], y, label='Data')
plt.plot(X[:, 1], model.predict(X), color='red', label='Fitted line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Using scikit-learn

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generating sample data
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 * X + np.random.normal(0, 0.1, 100)

# Fitting the model
model = LinearRegression()
model.fit(X, y)

# Model coefficients
print('Intercept:', model.intercept_)
print('Slope:', model.coef_)

# Plotting the results
plt.scatter(X, y, label='Data')
plt.plot(X, model.predict(X), color='red', label='Fitted line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Multiple Linear Regression

Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

- y: Dependent variable
- $x_1, x_2, \ldots, x_n$: Independent variables
- $\beta_0, \beta_1, \ldots, \beta_n$: Coefficients
- ϵ: Error term

Estimating Coefficients

- Similar to simple linear regression, OLS is used to estimate the coefficients.

Interpretation of Coefficients

- **Intercept (β0\beta_0β0)**: The value of yyy when all xix_ixi are 0.
- **Coefficient (βi\beta_iβi)**: The change in yyy for a one-unit change in xix_ixi, holding other variables constant.

Goodness of Fit

- **Adjusted R-squared**: Adjusted for the number of predictors in the model. It penalizes for adding non-significant predictors.

Python Code for Multiple Linear Regression

Using statsmodels

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Generating sample data
np.random.seed(0)
X1 = np.random.rand(100)
X2 = np.random.rand(100)
y = 3 * X1 + 2 * X2 + np.random.normal(0, 0.1, 100)

# Creating a DataFrame
df = pd.DataFrame({'X1': X1, 'X2': X2, 'y': y})

# Adding a constant (intercept) to the model
X = sm.add_constant(df[['X1', 'X2']])

# Fitting the model
model = sm.OLS(df['y'], X).fit()

# Model summary
print(model.summary())
```

Using scikit-learn

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

# Generating sample data
np.random.seed(0)
X1 = np.random.rand(100)
X2 = np.random.rand(100)
y = 3 * X1 + 2 * X2 + np.random.normal(0, 0.1, 100)

# Creating the feature matrix and target vector
X = np.column_stack((X1, X2))
y = y

# Fitting the model
model = LinearRegression()
model.fit(X, y)

# Model coefficients
print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)
```

```
# Making predictions
y_pred = model.predict(X)

# Evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score
print('Mean Squared Error:', mean_squared_error(y, y_pred))
print('R-squared:', r2_score(y, y_pred))
```

Logistic Regression

Logistic regression is a statistical method used for binary classification problems, where the outcome variable is categorical with two possible outcomes. It models the probability that a given input point belongs to a certain class.

Types of Logistic Regression

1. **Binary Logistic Regression**: The outcome variable has two categories.
2. **Multinomial Logistic Regression**: The outcome variable has more than two categories.
3. **Ordinal Logistic Regression**: The outcome variable has ordered categories.

Assumptions of Logistic Regression

1. **Linearity of the logit**: The logit (log-odds) of the outcome is a linear combination of the predictor variables.
2. **Independence of errors**: Observations are independent of each other.
3. **Absence of multicollinearity**: Independent variables are not highly correlated.
4. **Large sample size**: Logistic regression requires large sample sizes for reliable results.

Binary Logistic Regression

Estimating Coefficients

- **Maximum Likelihood Estimation (MLE)** is used to estimate the coefficients.

Interpretation of Coefficients

- **Intercept (β0)**: Log-odds of the outcome when all predictors are zero.
- **Coefficient (βi)**: Change in the log-odds of the outcome for a one-unit change in xix_ixi.

Goodness of Fit

- **Likelihood Ratio Test**: Compares the fit of two nested models.
- **Hosmer-Lemeshow Test**: Tests the goodness of fit for logistic regression models.
- **Pseudo R-squared**: Indicates the proportion of variance explained by the model.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

- p: Probability of the event occurring (e.g., $y = 1$)
- $x_1, x_2, \ldots, x_n$: Independent variables
- $\beta_0, \beta_1, \ldots, \beta_n$: Coefficients

Code for Binary Logistic Regression

Using `statsmodels`

```
import numpy as np
import pandas as pd
import statsmodels.api as sm

# Generating sample data
np.random.seed(0)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] + np.random.normal(0, 0.1, 100) > 1).astype(int)

# Adding a constant (intercept) to the model
X = sm.add_constant(X)

# Fitting the model
model = sm.Logit(y, X).fit()

# Model summary
print(model.summary())
```

Using `scikit-learn`

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Generating sample data
np.random.seed(0)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] + np.random.normal(0, 0.1, 100) > 1).astype(int)

# Fitting the model
model = LogisticRegression()
model.fit(X, y)

# Model coefficients
print('Intercept:', model.intercept_)
print('Coefficients:', model.coef_)

# Making predictions
y_pred = model.predict(X)

# Evaluation metrics
print(confusion_matrix(y, y_pred))
print(classification_report(y, y_pred))
```

Practical Steps

1. Data Collection: Gather relevant data.
2. Exploratory Data Analysis (EDA): Understand the data through summary statistics and visualizations.
3. Model Specification: Define the model structure and select variables.
4. Model Fitting: Use statistical software to fit the model.
5. Model Evaluation: Use diagnostic measures to check assumptions and evaluate performance.
6. Prediction: Use the model to make predictions on new data.