06 Regression

01 Univariate Linear Regression

- Linear regression is a supervised learning algorithm used to predict a continuous target variable based on one or more input features

$\hat{y} = \theta_0 + \theta_1 x$

- (\hat{y}): Predicted value
- (\theta_0): Intercept (bias term)
- (\theta_1): Coefficient (slope) for the feature (x)
- (x): Input feature

Linear Regression from Scratch

- The relationship between the input features and the target variable is modeled as a linear equation

01 Generating the Dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Create a simple dataset
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

02 Loss Function

- The loss function for linear regression is the Mean Squared Error (MSE)

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$

- (m) is the number of training examples
- (\hat{y}^{(i)}) is the predicted value
- (y^{(i)}) is the actual value

03 Gradient Descent

- We use gradient descent to minimize the MSE by iteratively updating the model parameters
- **Gradient Descent Update Rule**

$$\theta_j = \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) \cdot x_j^{(i)}$$

- (\alpha) is the learning rate
- (\theta_j) are the model parameters

```
# Initialize parameters
theta = np.random.randn(2, 1)  # For theta_0 (bias) and theta_1 (slope)
alpha = 0.01
iterations = 1000
m = len(X_train)

# Add bias term (x0 = 1) to the training data
X_b_train = np.c_[np.ones((m, 1)), X_train]

# Gradient Descent
for iteration in range(iterations):
    gradients = (2/m) * X_b_train.T.dot(X_b_train.dot(theta) - y_train)
    theta = theta - alpha * gradients

print(f"Optimized parameters (theta_0, theta_1): {theta.ravel()}")
```

04 Predictions on Test Data

```
# Add bias term to the test data
X_b_test = np.c_[np.ones((len(X_test), 1)), X_test]

# Predict
y_pred = X_b_test.dot(theta)
```

05 Evaluating the Model

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on test set: {mse}")
```

06 Plotting the Results

- We can visualize the linear regression line along with the training data

```
plt.scatter(X_test, y_test, color='blue', label='Test data')
plt.plot(X_test, y_pred, color='red', label='Regression line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

02 Multivariable Linear Regression

- Multivariable (or multiple) linear regression is an extension of simple linear regression where the target variable depends on more than one input feature
- It models the relationship between the dependent variable (target) and multiple independent variables (features) as a linear equation

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- (n) is the number of features
- (\hat{y}): Predicted value
- (\theta_0): Intercept (bias term)
- (\theta_1, \theta_2, \dots, \theta_n): Coefficients (weights) for each feature
- (x_1, x_2, \dots, x_n): Input features

In matrix form, this can be written as
$\hat{y} = X\theta$

- (X) is the feature matrix (including a column of ones for the bias term)
- (\theta) is the vector of parameters (including the bias term)

Implementing Multivariable Linear Regression

- Let's implement multivariable linear regression from scratch using a synthetic dataset

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Generate a synthetic dataset
np.random.seed(42)
X1 = 2 * np.random.rand(100, 1)  # Feature 1: size in square feet
X2 = np.random.rand(100, 1)  # Feature 2: number of bedrooms
y = 4 + 3 * X1 + 2 * X2 + np.random.randn(100, 1)  # Target variable:
house price

# Combine features into a single matrix
X = np.c_[X1, X2]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize parameters
theta = np.random.randn(3, 1)  # For theta_0 (bias), theta_1 (size),
theta_2 (bedrooms)
alpha = 0.01
iterations = 1000
m = len(X_train)

# Add bias term (x0 = 1) to the training data
X_b_train = np.c_[np.ones((m, 1)), X_train]

# Gradient Descent
for iteration in range(iterations):
    gradients = (2/m) * X_b_train.T.dot(X_b_train.dot(theta) - y_train)
    theta = theta - alpha * gradients

print(f"Optimized parameters (theta_0, theta_1, theta_2):
{theta.ravel()}")
```

```
# Add bias term to the test data
X_b_test = np.c_[np.ones((len(X_test), 1)), X_test]

# Predict
y_pred = X_b_test.dot(theta)
```

```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on test set: {mse}")
```

Use in Practice

- In practice, you can use libraries like `scikit-learn` to implement multivariable linear regression more efficiently

```
from sklearn.linear_model import LinearRegression

# Create the model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on test set: {mse}")
```