

Regression and Classification

A **regression problem** is a type of supervised learning where the output variable (target) is **continuous** in nature. The objective is to predict a **real-valued** quantity based on input variables.


Example: Predicting a person's weight based on height and age.

Characteristics of Regression:

- Output is a real number (e.g., price, temperature, age).
- Useful for forecasting and estimating numerical values.
- Can handle both linear and nonlinear relationships.

The most common regression algorithm is **Linear Regression**. It models the relationship between dependent and independent variables as a straight line. It assumes linearity and constant variance of errors.

Evaluation Metrics for Regression

Metric	Formula	Description	
Mean Absolute Error (MAE)	$(\frac{1}{n} \sum$	$y_i - \hat{y}_i$	
Mean Squared Error (MSE)	$\frac{1}{n} \sum (y_i - \hat{y}_i)^2$	Penalizes larger errors more than MAE.	
Root Mean Squared Error (RMSE)	$\sqrt{\text{MSE}}$	Square root of MSE; same units as the target.	
R ² Score (Coefficient of Determination)	$1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$	Measures how much of the variance is explained by the model.	


A **classification problem** is a supervised learning task where the output variable (target) is **categorical** in nature. The goal is to assign each input to one of several predefined classes or categories.

Example: Predicting whether an email is spam or not (Spam / Not Spam).

Characteristics of Classification:

- Output is a discrete label (e.g., Yes/No, Disease/No Disease).
- Can be **binary**, **multiclass**, or **multilabel**.
- Involves decision boundaries between classes.

Common Classification Algorithms

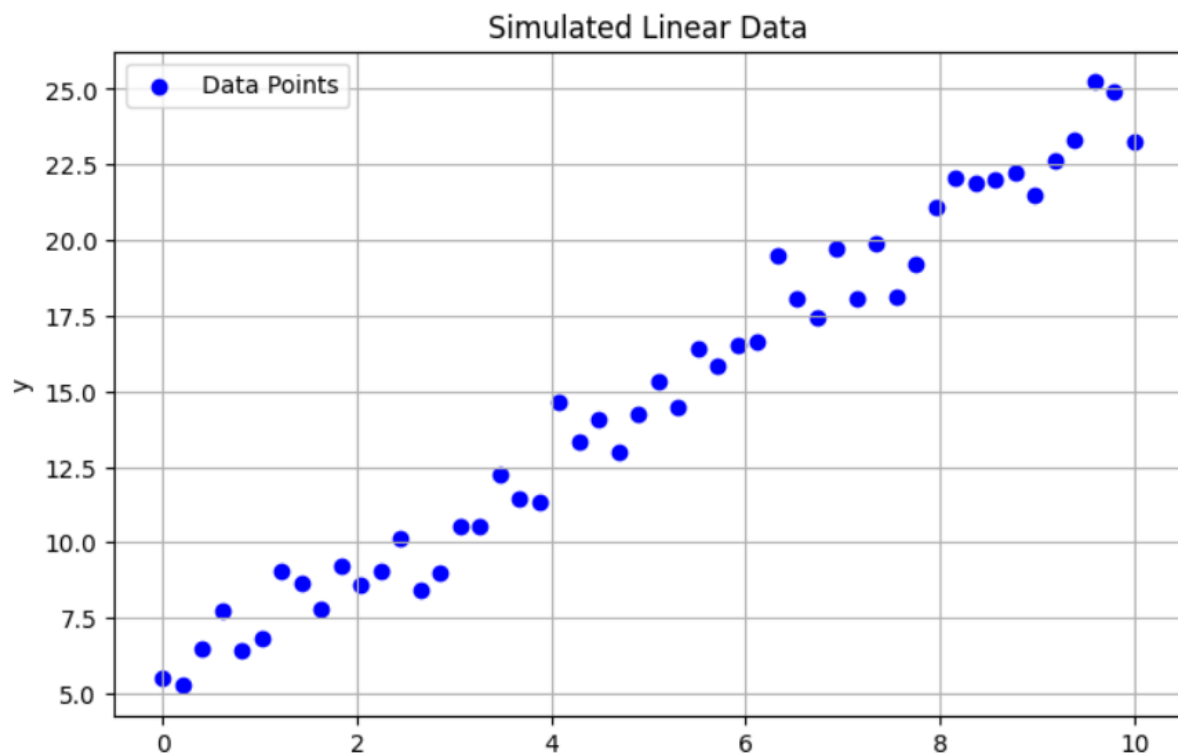
Algorithm	Description	
Logistic Regression	Despite the name, it is used for classification. Outputs probabilities using the sigmoid function.	
K-Nearest Neighbors (KNN)	Classifies a point based on the majority class of its k-nearest neighbors.	
Decision Tree Classifier	Builds a tree by splitting features based on information gain or Gini index.	
Random Forest Classifier	Ensemble of decision trees; robust against overfitting.	
Support Vector Machines (SVM)	Finds the optimal hyperplane that maximizes margin between classes.	
Naive Bayes	Based on Bayes' theorem assuming feature independence. Efficient for text classification.	
Gradient Boosting Classifier (XGBoost, LightGBM)	Sequentially builds classifiers that correct the previous ones.	
Neural Networks (MLP, CNN, etc.)	Deep models for complex classification like image, speech, and text recognition.	

Linear Regression

```
# Import required libraries
import numpy as np
import matplotlib.pyplot as plt

# Let's simulate some linear data:  $y = 2x + 5$  (with a bit of noise)
np.random.seed(42) # For reproducibility
x = np.linspace(0, 10, 50) # 50 values from 0 to 10
noise = np.random.normal(0, 1, size=x.shape) # Random noise
y = 2 * x + 5 + noise # Actual y values with noise

# Plot the data
plt.figure(figsize=(8, 5))
plt.scatter(x, y, color='blue', label='Data Points')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simulated Linear Data')
plt.legend()
plt.grid(True)
plt.show()
```



Linear Regression is a fundamental statistical method used to model the **relationship between two variables** — an **independent variable** x and a **dependent variable** y — by fitting a **straight line** through the data points.

The goal is to find the **best-fit line**:

$$y = ax + b$$

Where:

- a is the **slope** of the line (how much y changes with x)
- b is the **intercept** (the value of y when $x = 0$)

Objective of Linear Regression

To **minimize the error** between the actual values y_i and the predicted values \hat{y}_i .

This error is often called the **residual**:

$$\text{Residual} = y_i - \hat{y}_i$$

To measure the overall error across all points, we use the **sum of squared errors (SSE)**:

$$SSE = \sum_{i=1}^n (y_i - (ax_i + b))^2$$

This is where **Least Squares** comes in.



Least Squares Method

The **Least Squares** method finds the values of a and b that **minimize the sum of squared errors**.

We want to minimize:

$$J(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

To find the minimum, we take the **partial derivatives** of $J(a, b)$ with respect to a and b , set them to zero, and solve.



Deriving the Formula

Let:

- $\bar{x} = \frac{1}{n} \sum x_i$ (mean of x)
- $\bar{y} = \frac{1}{n} \sum y_i$ (mean of y)

1. Slope (a)

$$a = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

This is the **covariance of x and y** divided by the **variance of x**.

2. Intercept (b)

$$b = \bar{y} - a\bar{x}$$

```

import numpy as np
import matplotlib.pyplot as plt

# Simulate some linear data: y = 2x + 5 (with a bit of noise)
np.random.seed(42) # For reproducibility
x = np.linspace(0, 10, 50) # 50 values from 0 to 10
noise = np.random.normal(0, 1, size=x.shape) # Random noise
y = 2 * x + 5 + noise # Actual y values with noise

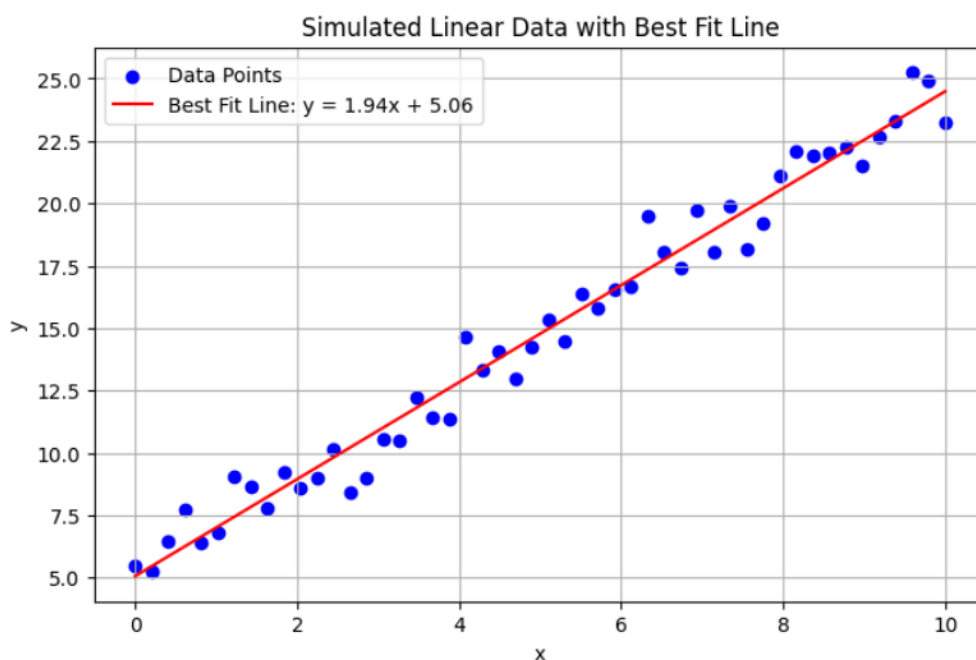
# Compute least squares regression (slope and intercept)
def compute_least_squares(x, y):
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    a = np.sum((x - x_mean) * (y - y_mean)) / np.sum((x - x_mean)**2)
    b = y_mean - a * x_mean
    return a, b

# Get slope and intercept
a, b = compute_least_squares(x, y)

# Predicted y values for the best fit line
y_pred = a * x + b

# Plot the data and the best fit line
plt.figure(figsize=(8, 5))
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, y_pred, color='red', label=f'Best Fit Line: y = {a:.2f}x + {b:.2f}')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simulated Linear Data with Best Fit Line')
plt.legend()
plt.grid(True)
plt.show()

```



Note: Apart from the Least Squares method, common optimization techniques for minimizing the cost function include Gradient Descent (batch, stochastic, mini-batch), Normal Equation for closed-form solutions, Newton's Method using second-order derivatives, Conjugate Gradient for large systems, L-BFGS as a memory-efficient quasi-Newton method, and Robust Regression approaches like L1 loss or Huber loss to handle outliers.

Introduction to Scikit-learn

Scikit-learn (also written as `scikit-learn` or `sklearn`) is one of the most popular and powerful machine learning libraries in Python. It provides simple and efficient tools for data mining, data analysis, and machine learning, and is built on top of NumPy, SciPy, and matplotlib.

◆ 1. What is Scikit-learn?

Scikit-learn is an open-source Python library designed for:

- Supervised learning (classification, regression)
- Unsupervised learning (clustering, dimensionality reduction)
- Model selection (grid search, cross-validation)
- Data preprocessing (scaling, normalization, encoding)

It is widely used in academia and industry due to its ease of use and powerful features.

◆ 2. Key Features of Scikit-learn

Feature	Description
Consistent API	All estimators follow a simple and consistent interface (<code>fit</code> , <code>predict</code> , <code>transform</code>)
Wide range of ML algorithms	Supports most standard machine learning models
Built-in utilities	For model evaluation, tuning, preprocessing, and pipelines
Integration with NumPy/Pandas	Works seamlessly with NumPy arrays and Pandas DataFrames
Documentation	Comprehensive and beginner-friendly

◆ 3. Scikit-learn Workflow Overview

Step-by-step:

1. Import the dataset
2. Split into training and testing sets
3. Preprocess the data (optional)
4. Select and initialize the model
5. Train the model (`fit`)
6. Make predictions (`predict`)
7. Evaluate the model

◆ 4. Popular Machine Learning Algorithms in Scikit-learn

■ Classification:

- Logistic Regression
- K-Nearest Neighbors (KNN)
- Decision Trees

■ Regression:

- Linear Regression
- Ridge and Lasso Regression
- Decision Tree Regression

◆ 5. Preprocessing Tools

🔧 Data Cleaning and Transformation

- `StandardScaler`, `MinMaxScaler`: Scaling features
- `OneHotEncoder`, `LabelEncoder`: Categorical encoding
- `Imputer` (now `SimpleImputer`): Handling missing values
- `PolynomialFeatures`: Creating interaction terms

◆ 6. Model Selection & Evaluation

✅ Techniques:

- Train/Test split: `train_test_split`
- Cross-validation: `cross_val_score`, `KFold`
- Grid Search: `GridSearchCV`, `RandomizedSearchCV`
- Metrics: accuracy, precision, recall, F1-score, RMSE

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

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

model = LogisticRegression()
model.fit(X_train, y_train)



y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

◆ 7. Pipeline in Scikit-learn

A pipeline chains multiple preprocessing steps and a final estimator into one unit.

Example:

python

 Copy  Edit

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression())
])

pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

◆ 8. Real-world Applications of Scikit-learn

- Spam detection using Naive Bayes
- Stock market prediction using regression models
- Customer segmentation using clustering
- Face recognition using PCA + classification
- Recommendation systems with collaborative filtering

◆ 9. Advantages and Limitations

✓ Advantages:

- Easy to use
- Well-documented
- Large community support
- Compatible with other Python libraries

✗ Limitations:

- Not ideal for **deep learning** (use TensorFlow, PyTorch instead)
- Limited to **in-memory** data (not suitable for very large datasets)
- Doesn't support GPU acceleration

◆ 10. Getting Started

```
bash
```

[Copy](#) [Edit](#)

```
pip install scikit-learn
```

Import:

```
python
```

[Copy](#) [Edit](#)

```
import sklearn
```

You can explore more from the [official documentation](#).

Project: Employee Salary Prediction using Multiple Linear Regression