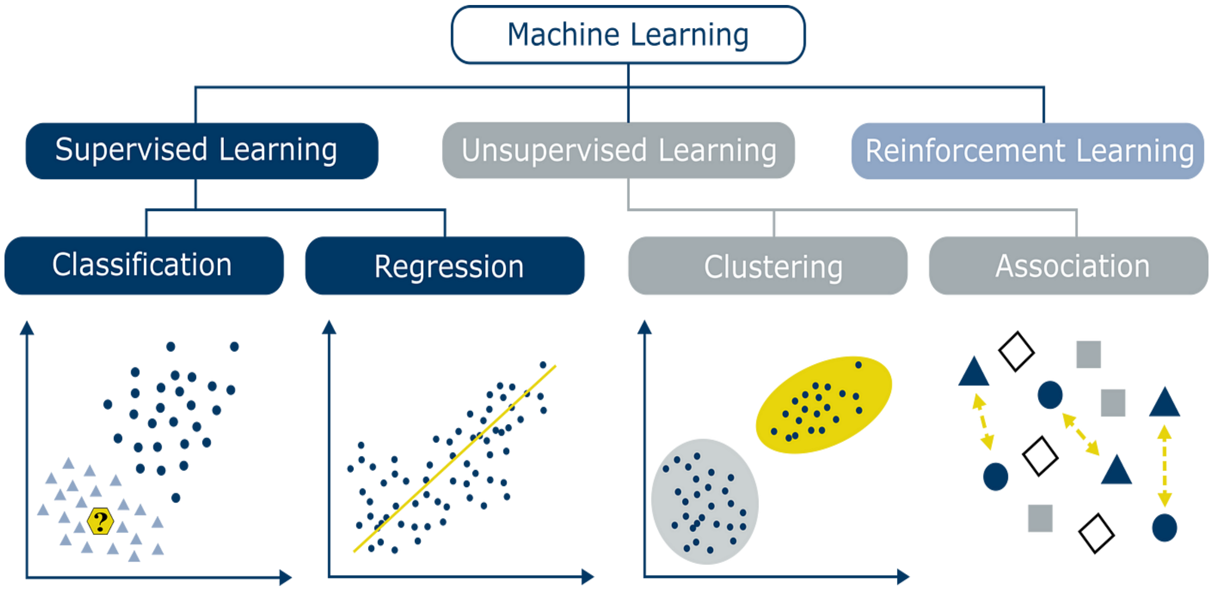
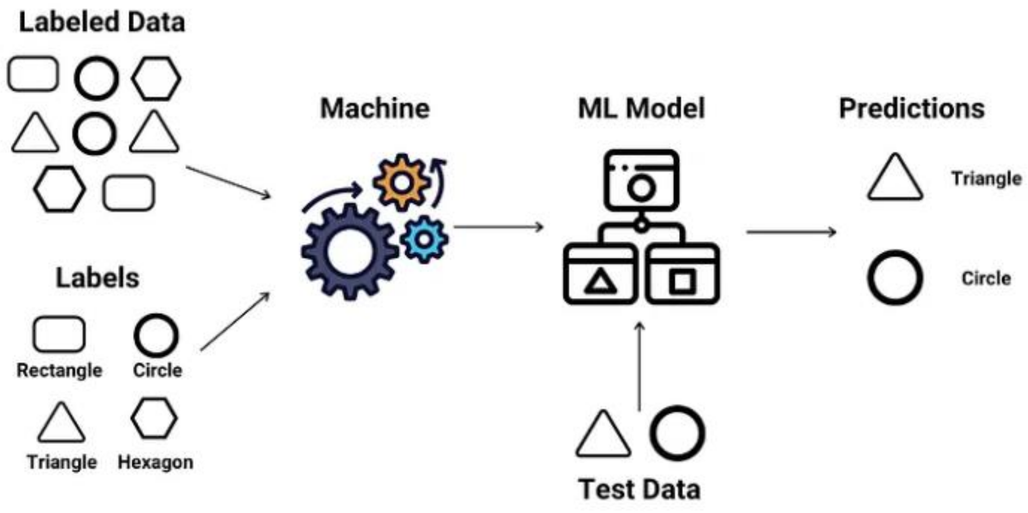
Supervised Learning-Regression

Introduction to Regression analysis



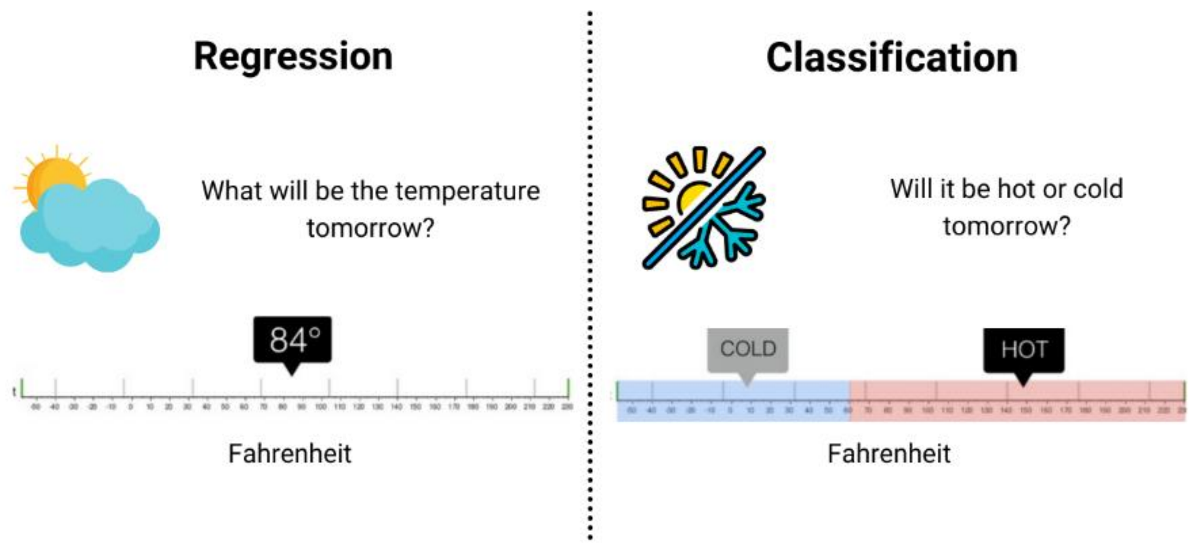
Supervised Learning

Supervised learning is a type of machine learning where an algorithm is trained on labeled data. The model learns from input-output pairs, where the correct output is already known. The goal is to predict the output for new, unseen data.



There are two main types of supervised learning:

1. **Classification**: Predicts discrete labels (e.g., spam or not spam).
2. **Regression**: Predicts continuous values (e.g., house prices).


Regression
What will be the temperature tomorrow?
84°
Fahrenheit
Classification
Will it be hot or cold tomorrow?
COLD
HOT
Fahrenheit

Classification

Classification models classify our outputs to certain categories. If the number of categories are only two then it is specially called binary classification. For greater number of categories it is called multi-class classification.

Some examples are:

- *Whether a patient has cancer or not*

Regression

Regression models are for labeling outputs with continuous values.

- Predicting house prices, or
- How long is it gonna take you to get home

are both for regression models because the results are ever changing.

The key components of supervised learning include:

1. **Input Features (X)**: These are the independent variables or attributes used by the model to make predictions. They could be anything from numerical data, such as age or income, to more complex data like images or text.
2. **Output Labels (Y)**: These are the dependent variables or target outcomes that the model is trying to predict. In classification tasks, the outputs are discrete labels (e.g., "dog" or "cat"), while in regression tasks, the outputs are continuous values (e.g., house prices).

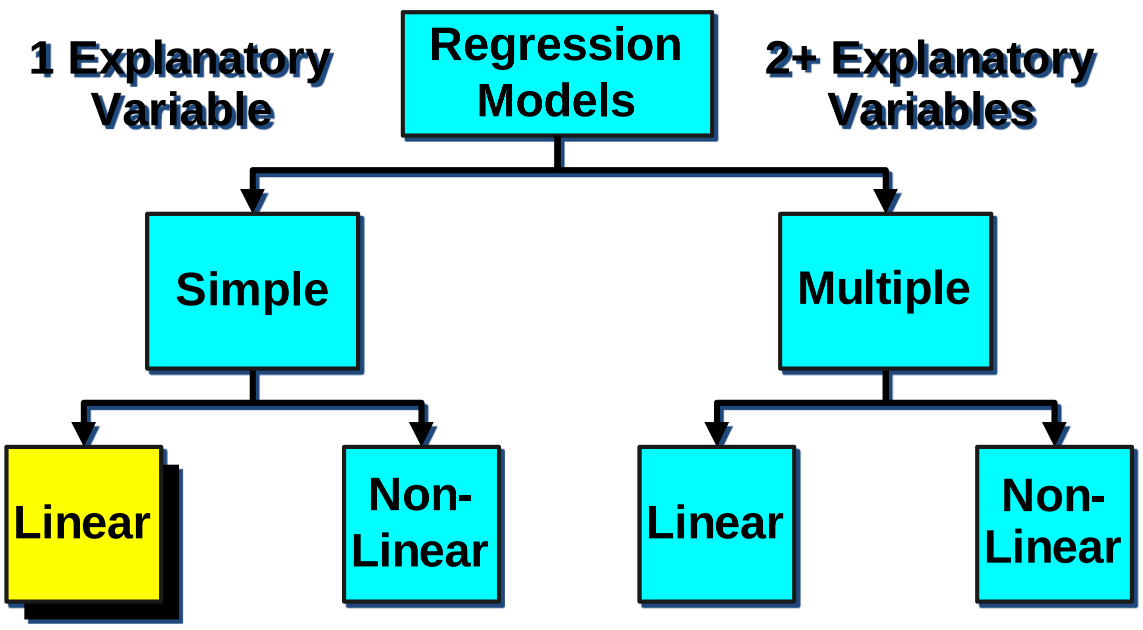
3. **Training Data**: A labeled dataset that the model uses to learn. Each example in the dataset consists of an input (X) and its corresponding correct output (Y).
4. **Learning Algorithm**: This is the method the model uses to learn the relationship between inputs and outputs.

Introduction to Regression Analysis:

Regression analysis is a key technique in supervised learning used to predict continuous outcomes. It helps in understanding the relationship between a dependent variable (the outcome) and one or more independent variables (the inputs or predictors). The primary goal of regression is to create a model that can predict the value of the dependent variable given the independent variables.

There are various types of regression techniques, with the most common being:

1. **Linear Regression**: Models the relationship between the dependent variable (Y) and one or more independent variables (X) by fitting a straight line (Y = bX + c).
2. **Multiple Regression**: Extends linear regression to multiple input variables.
3. **Polynomial Regression**: Fits a curve to the data instead of a straight line.
4. **Logistic Regression**: Though called regression, it’s used for classification tasks, predicting the probability of an event.



Simple Linear Regression:

Simple linear regression is a model that assesses the relationship between a dependent variable and an independent variable.

The simple linear model is expressed using the following equation:

y=mx+b

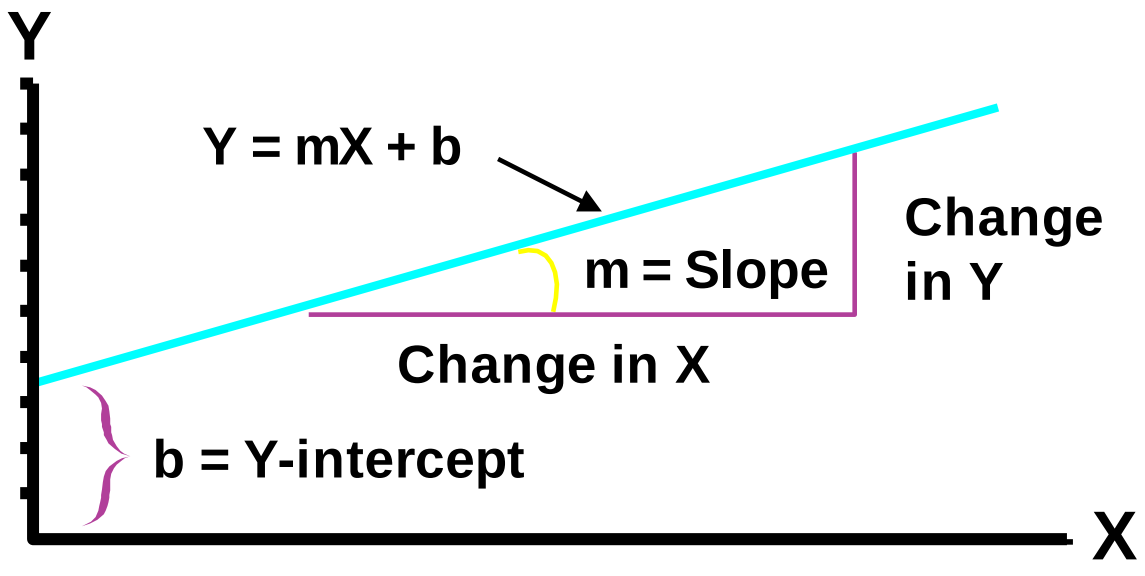
where,

y - Dependent variable

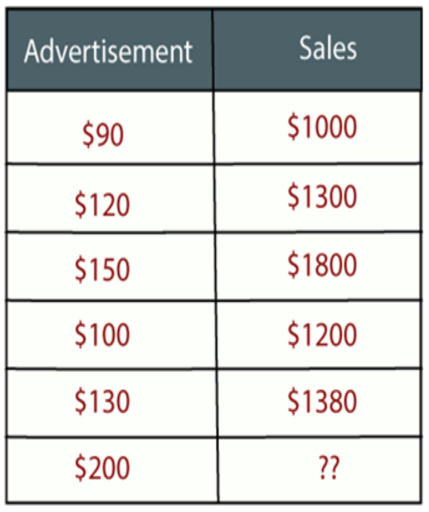
x - Independent variable

m - Slope

b - Intercept



Example: Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:



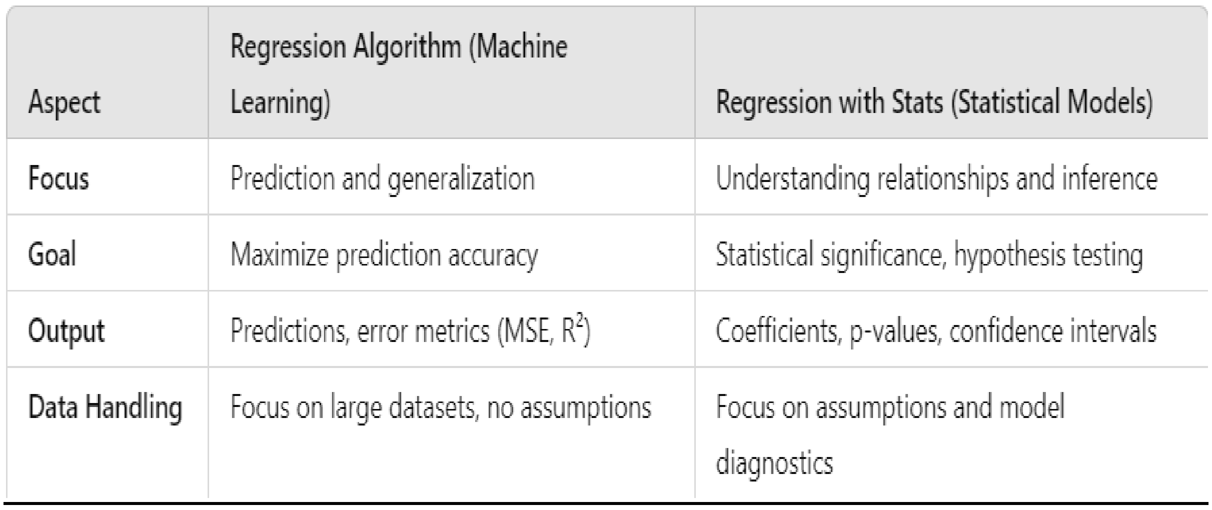
Advertisement	Sales
$90	$1000
$120	$1300
$150	$1800
$100	$1200
$130	$1380
$200	??

Supervised Learning-Regression

Measure of linear relationship, Regression with stats models

Regression algorithm (uisng machine learning) and **regression with statistical models** lies in their primary goals, approaches, and outputs. Both aim to model relationships between variables, but their use cases and how they handle data can be quite distinct.

- **Regression Algorithm (Machine Learning)**: The focus is on **prediction** and **generalization** to unseen data. It aims to create a model that can predict the target (dependent variable) based on the features (independent variables.
- **Regression with Stats (Statistical Models)**: The focus is on **understanding relationships** between variables. It's often used for explaining the nature of the relationship between independent variables (predictors) and the dependent variable (outcome).



Aspect	Regression Algorithm (Machine Learning)	Regression with Stats (Statistical Models)
Focus	Prediction and generalization	Understanding relationships and inference
Goal	Maximize prediction accuracy	Statistical significance, hypothesis testing
Output	Predictions, error metrics (MSE, R^2)	Coefficients, p-values, confidence intervals
Data Handling	Focus on large datasets, no assumptions	Focus on assumptions and model diagnostics

To measure the linear relationship between two variables and perform regression analysis using **statsmodels** in Python, you can follow these steps:

1. Measure Linear Relationship (Correlation)

Before performing regression, it's common to measure the linear relationship between two variables using **correlation**. You can use **Pearson's correlation coefficient** to measure this.

```python
import pandas as pd
import numpy as np

# Example data
data = {'X': [1, 2, 3, 4, 5], 'Y': [2, 4, 5, 4, 5]}
df = pd.DataFrame(data)

# Pearson correlation coefficient
correlation = df['X'].corr(df['Y'])
print("Pearson Correlation Coefficient:", correlation)
```

```
Pearson Correlation Coefficient: 0.7745966692414834
```

The correlation coefficient ranges from -1 to 1:

- 1 indicates a perfect positive linear relationship.
- -1 indicates a perfect negative linear relationship.
- 0 indicates no linear relationship.

2. Linear Regression Using Statsmodels

You can use the **statsmodels** library to perform linear regression. Here's an example:

Step-by-step guide:

1. Install **statsmodels** if not already installed.

```
pip install statsmodels
```

2. **Perform the regression:**

python

```python
import statsmodels.api as sm

# Define the dependent (Y) and independent (X) variables
X = df['X']  # Independent variable
Y = df['Y']  # Dependent variable

# Add a constant to the independent variable (intercept)
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(Y, X).fit()

# Print out the summary of the regression
print(model.summary())
```

- **sm.add_constant(X)** adds a constant term to the predictor variable for the intercept.
- **sm.OLS(Y, X)** creates the OLS (Ordinary Least Squares) model.
- **model.fit()** fits the model to the data.
- **model.summary()** provides a detailed statistical summary, including coefficients, R-squared value, p-values, and confidence intervals.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.600
Model:                            OLS   Adj. R-squared:                  0.467
Method:                 Least Squares   F-statistic:                     4.500
Date:                Thu, 17 Oct 2024   Prob (F-statistic):              0.124
Time:                        09:20:58   Log-Likelihood:                -5.2598
No. Observations:                   5   AIC:                             14.52
Df Residuals:                       3   BIC:                             13.74
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.2000      0.938      2.345      0.101      -0.785       5.185
X              0.6000      0.283      2.121      0.124      -0.300       1.500
==============================================================================
Omnibus:                          nan   Durbin-Watson:                   2.017
Prob(Omnibus):                    nan   Jarque-Bera (JB):                0.570
Skew:                           0.289   Prob(JB):                        0.752
Kurtosis:                       1.450   Cond. No.                         8.37
==============================================================================
```

3. Interpreting the Results

- **R-squared**: Indicates how well the independent variable explains the variability in the dependent variable.
- **p-value**: Tests the null hypothesis that the coefficient of a variable is zero (no relationship). A p-value less than 0.05 usually indicates statistical significance.
- **Coefficients**: The slope of the regression line (relationship strength) and the intercept.

Studying regression with **statsmodels** offers several advantages, especially for those who want a deeper understanding of regression analysis and statistical modeling. Here are some reasons why **statsmodels** is beneficial for studying regression:

Detailed Statistical Output

Unlike other libraries like **scikit-learn**, which focus primarily on prediction, **statsmodels** provides comprehensive statistical information about the regression model. The output includes:

- **R-squared and Adjusted R-squared**: Measures of model fit.
- **P-values**: Help determine the statistical significance of predictors.
- **Confidence Intervals**: Shows the range of values for the coefficients with a certain level of confidence.

Supervised Learning-Regression

Determining coefficient, meaning and significance of coefficients.

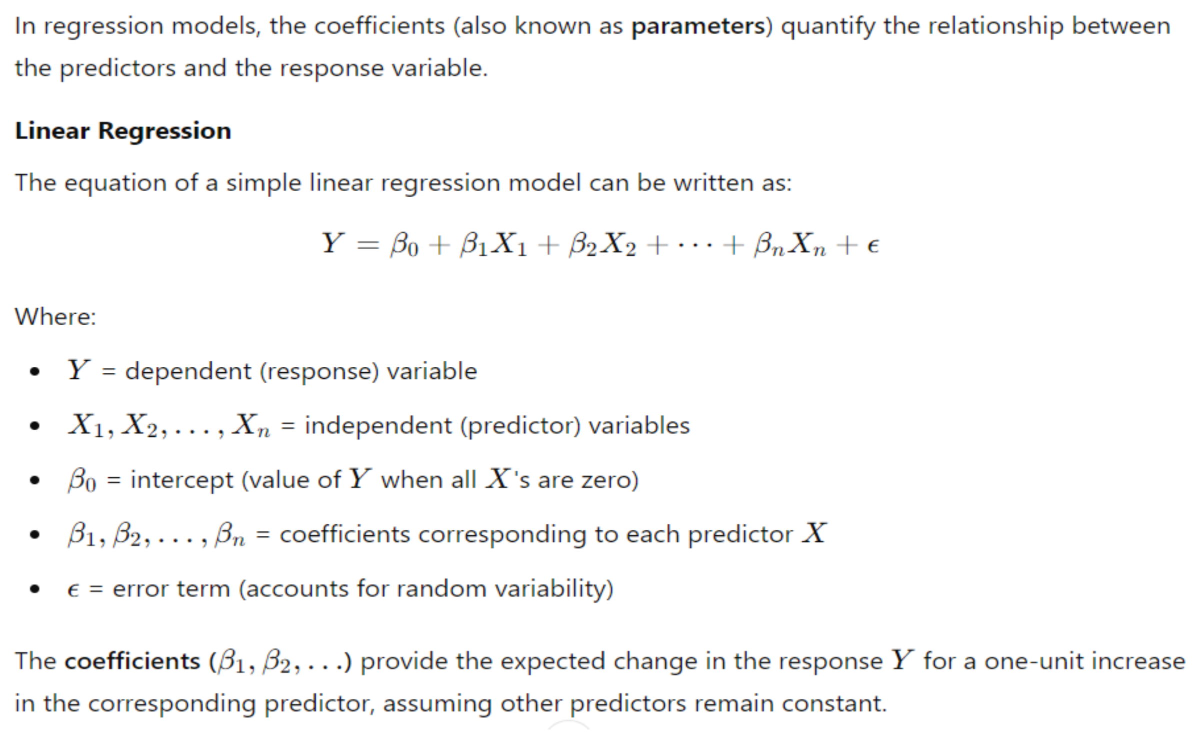
In a regression analysis, coefficients represent the relationship between independent variables (predictors) and the dependent variable (outcome). Their interpretation depends on the type of regression model:

1. Linear Regression Coefficients: These represent the change in the dependent variable for a one-unit change in the independent variable, holding other variables constant.

   For example, if the coefficient of a variable (e.g., X1) is 3, it means that for every one-unit increase in X1, the dependent variable Y increases by 3 units.

2. Logistic Regression Coefficients: These coefficients (often referred to as log-odds) represent the effect of predictors on the log-odds of the outcome. The coefficients can be transformed into odds ratios for easier interpretation in probability terms.

3. Polynomial or Non-linear Models: The coefficients may represent different powers of the independent variables and can indicate curvature or more complex relationships between predictors and the outcome.



In regression models, the coefficients (also known as **parameters**) quantify the relationship between the predictors and the response variable.

Linear Regression

The equation of a simple linear regression model can be written as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon$$

Where:

- Y = dependent (response) variable
- $X_1, X_2, \ldots, X_n$ = independent (predictor) variables
- β_0 = intercept (value of Y when all X's are zero)
- $\beta_1, \beta_2, \ldots, \beta_n$ = coefficients corresponding to each predictor X
- ϵ = error term (accounts for random variability)

The **coefficients** ($\beta_1, \beta_2, \ldots$) provide the expected change in the response Y for a one-unit increase in the corresponding predictor, assuming other predictors remain constant.

Logistic Regression

Logistic regression is used for binary outcomes (e.g., success/failure, yes/no). The model can be written as:

$$\text{logit}(P) = \ln\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

Where:

- P is the probability of the event occurring (e.g., success).
- The coefficients here represent the change in the **log-odds** of the outcome for a one-unit change in the predictor.

To make these coefficients interpretable in terms of probability, you can exponentiate them to obtain the **odds ratio**:

$$\text{Odds Ratio} = e^{\beta}$$

This represents the multiplicative change in the odds for a one-unit increase in the predictor.

The process of determining coefficients involves **estimating** them from the data. This is typically done through **optimization techniques** that minimize the error between predicted and actual values.

Ordinary Least Squares (OLS) for Linear Regression

OLS is the most common method used to estimate coefficients in linear regression. It aims to minimize the **sum of squared residuals (errors)**:

$$\text{Minimize} \sum (Y_{\text{actual}} - Y_{\text{predicted}})^2$$

The solution to this minimization problem gives the **best-fit line** that represents the relationship between predictors and the dependent variable.

Maximum Likelihood Estimation (MLE) for Logistic Regression

In logistic regression, the likelihood of the observed data is maximized. The coefficients are estimated by finding the values that maximize the likelihood of observing the data given the model.

The process of determining coefficients involves **estimating** them from the data. This is typically done through **optimization techniques** that minimize the error between predicted and actual values.

Ordinary Least Squares (OLS) for Linear Regression

OLS is the most common method used to estimate coefficients in linear regression. It aims to minimize the **sum of squared residuals (errors)**:

$$\text{Minimize} \sum (Y_{\text{actual}} - Y_{\text{predicted}})^2$$

The solution to this minimization problem gives the **best-fit line** that represents the relationship between predictors and the dependent variable.

Maximum Likelihood Estimation (MLE) for Logistic Regression

In logistic regression, the likelihood of the observed data is maximized. The coefficients are estimated by finding the values that maximize the likelihood of observing the data given the model.

Key Points of Coefficients:

- **Magnitude: Indicates the strength of the relationship between the predictor and the outcome.**
- **Sign: Indicates the direction of the relationship (positive or negative).**
- **Statistical significance: Tells whether the relationship is likely to be genuine or due to chance.**

Supervised Learning-Regression

Simple Linear Regression using sample dataset.

Simple Linear Regression is a statistical technique that models the relationship between two variables: a dependent variable (Y) and an independent variable (X). It assumes that the relationship between the variables is linear, which means that we can represent it with a straight line of the form:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Where:

- Y is the dependent variable (what you are trying to predict).
- X is the independent variable (the predictor).
- β0 is the intercept of the regression line (the value of Y when X = 0).
- β1 is the slope of the regression line (how much Y changes for a unit change in X).
- ϵ is the error term (captures any deviations from the exact linear relationship).

Methodology

1. **Hypothesis Setup**: Simple linear regression aims to find the line that best fits the data points in a two-dimensional plane.
2. **Data Collection**: Collect data where you have observations for both the dependent and independent variables.
3. **Fitting the Model**: The goal is to estimate the parameters β0 & β1 such that the sum of squared residuals (the difference between the actual value and the predicted value) is minimized.
4. **Model Evaluation**: After fitting the model, evaluate how well the model predicts by using metrics such as **R-squared** and the **mean squared error (MSE)**.
5. **Prediction**: Once the model is trained, we can use the estimated coefficients to predict new data.

Use Case: Predicting House Prices

Consider a real-world scenario where you want to predict house prices (Y) based on the size of the house (X). This is a simple linear regression problem because house price can be predicted using one feature: house size.

Dataset

For simplicity, let's use a sample dataset of house sizes (in square feet) and house prices (in thousands of dollars):

Size (sq.ft)	Price (thousands of dollars)
750	150
800	160
850	180
900	200
1000	220

- **Import Libraries**: The necessary libraries to be imported (numpy, matplotlib, and sklearn).
- **Define the Dataset**: The initial dataset of house sizes and prices
- **Model Creation**: A LinearRegression object is created, and the model is trained on the existing dataset using the .fit() method.
- **New Data for Prediction**: A new dataset (new_house_sizes) is created with house sizes of 1100, 1200, and 1300 square feet. This data will be used to make new price predictions using the .predict() method.
- **Plotting**: The plot includes the original data points, the regression line, and the predicted prices for the new house sizes (displayed in green).

```python
# Step 1: Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Step 2: Define the dataset
# House size (independent variable X)
X = np.array([750, 800, 850, 900, 1000]).reshape(-1, 1)

# House prices (dependent variable Y)
Y = np.array([150, 160, 180, 200, 220])
```

- `numpy` **(np)**: Used for handling arrays and reshaping data.
- `matplotlib.pyplot` **(plt)**: Used for plotting graphs, to visualize the relationship between house size and price.
- `LinearRegression`: The regression model from the **scikit-learn** library.

- `X`: This array contains the sizes of houses in square feet. The `reshape(-1, 1)` converts the 1D array into a 2D array as required by the regression model. Each row represents a different house size.

- `Y`: This array contains the corresponding prices (in thousands of dollars) for the house sizes in `X`.

```python
# Step 3: Create a linear regression model and fit the data
model = LinearRegression()
model.fit(X, Y)

# Step 4: Make predictions using the model on existing data
Y_pred = model.predict(X)
```

- `LinearRegression()` : Initializes a linear regression model.
- `model.fit(X, Y)` : Fits (or trains) the model using the data provided (`X` as input and `Y` as output). The model learns the relationship between house size and price, finding the best-fitting line.

- `model.predict(X)` : Predicts the house prices for the given house sizes (`X`) using the trained model.
- `Y_pred` : Stores the predicted prices for the house sizes in `X` (original data). These are the values lying on the regression line.

```python
# Step 5: New data for prediction
new_house_sizes = np.array([1100, 1200, 1300]).reshape(-1, 1)  # New house sizes (in sq.ft)

# Step 6: Predict prices for new house sizes
predicted_prices = model.predict(new_house_sizes)

# Step 7: Display results
print("Predicted Prices for New House Sizes:")
for size, price in zip(new_house_sizes, predicted_prices):
    print(f"Size: {size[0]} sq.ft -> Predicted Price: {price:.2f} thousands of dollars")
```

- `new_house_sizes` : This array contains the new house sizes (1100 sq.ft, 1200 sq.ft, and 1300 sq.ft) for which we want to predict the prices. Again, it's reshaped into a 2D array for input into the regression model.

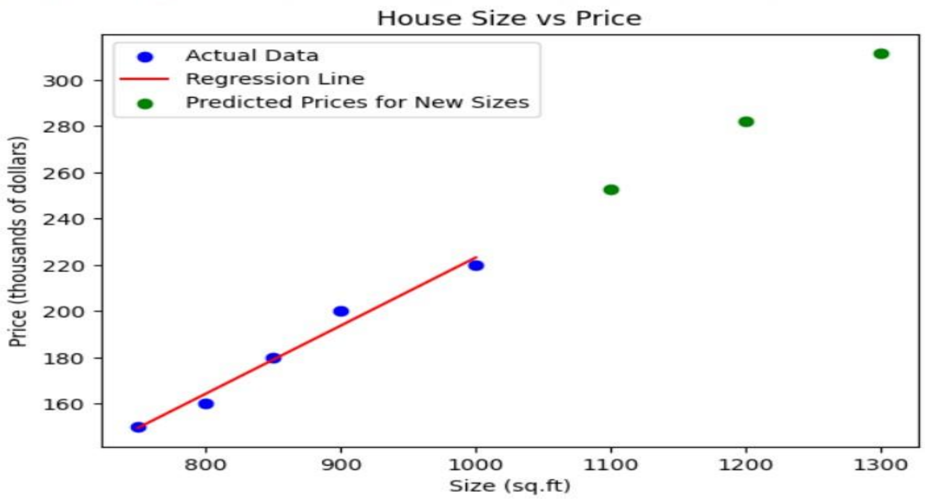
- `model.predict(new_house_sizes)` : Uses the trained model to predict the prices of houses based on the new house sizes provided.
- `predicted_prices` : Stores the predicted prices for the new house sizes.

- `for size, price in zip(new_house_sizes, predicted_prices)` : Loops over the new house sizes and their corresponding predicted prices.
- `print()` : Displays the predicted prices for each of the new house sizes, formatted to two decimal places.

```
# Step 8: Plotting the data and regression line
plt.scatter(X, Y, color="blue", label="Actual Data")
plt.plot(X, Y_pred, color="red", label="Regression Line")
plt.scatter(new_house_sizes, predicted_prices, color="green", label="Predicted Prices for New Sizes")
plt.title("House Size vs Price")
plt.xlabel("Size (sq.ft)")
plt.ylabel("Price (thousands of dollars)")
plt.legend()
plt.show()
```

- `plt.scatter(X, Y, color="blue")` : Plots the original data points (house sizes and actual prices) in blue.
- `plt.plot(X, Y_pred, color="red")` : Plots the regression line, which represents the predicted prices based on house size. It shows the best-fit linear relationship between size and price for the original data.
- `plt.scatter(new_house_sizes, predicted_prices, color="green")` : Plots the predicted prices for the new house sizes in green.
- `plt.title(), plt.xlabel(), plt.ylabel()` : Add a title and labels to the graph for clarity.
- `plt.legend()` : Adds a legend to distinguish between the actual data, the regression line, and the new predictions.
- `plt.show()` : Displays the plot.

```
Predicted Prices for New House Sizes:
Size: 1100 sq.ft -> Predicted Price: 252.70 thousands of dollars
Size: 1200 sq.ft -> Predicted Price: 282.16 thousands of dollars
Size: 1300 sq.ft -> Predicted Price: 311.62 thousands of dollars
```



Supervised Learning-Regression

Polynomial Regression using sample dataset

Polynomial Regression is an extension of linear regression where the relationship between the independent and dependent variables is modeled as an n-degree polynomial. It's useful when the data shows a non-linear relationship, but we still want to model it with linear techniques by transforming the input features.

The general form of a polynomial regression model is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \epsilon$$

where:

- y is the dependent variable.
- x is the independent variable.
- $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the polynomial terms.
- ϵ is the error term, accounting for noise in the data.

The degree of the polynomial n, determines the complexity of the model. For example:

- n=1: Linear regression (a straight line).
- n=2: Quadratic regression (a parabola).
- n=3: Cubic regression (a more flexible curve), and so on.

Why Use Polynomial Regression?

Polynomial regression is useful when data shows a non-linear trend that cannot be captured by a simple linear model. Examples include:

- Modeling the growth of populations.
- Predicting stock prices with seasonal trends.
- Tracking temperature fluctuations over time.

It provides a simple way to extend linear regression for capturing such non-linear patterns without resorting to more complex models like neural networks or non-parametric techniques.

Steps for Polynomial Regression

1. Import necessary libraries
2. Define the dataset
3. Transform the features into polynomial features
4. Fit the polynomial regression model
5. Predict new values
6. Visualize the result

Step 1: Import necessary libraries

- **numpy**: For handling array operations.
- **matplotlib.pyplot**: For plotting the graphs.
- **LinearRegression**: The linear regression model from sklearn.
- **PolynomialFeatures**: This will be used to transform the input features into polynomial features.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Step 2: Define the dataset

- **X**: The independent variable (input), representing numbers 1 through 10.
- **Y**: The dependent variable (output), following a quadratic pattern (i.e., Y=X^2)

```
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
Y = np.array([1, 4, 9, 16, 25, 36, 49, 64, 81, 100])  # Quadratic data (Y = X^2)
```

Step 3: Transform the features into polynomial features

- **PolynomialFeatures(degree=2)**: Transforms the input features into polynomial features of degree 2. For example, if X = [x1], it becomes X_poly = [1, x1, x1^2].

- **fit_transform(X)**: Transforms the input array X into polynomial features.

```
poly_features = PolynomialFeatures(degree=2)
X_poly = poly_features.fit_transform(X)
```

Step 4: Fit the polynomial regression model

- **LinearRegression()**: Initializes the linear regression model.
- **model.fit(X_poly, Y)**: Trains the linear regression model using the polynomial features of X and the dependent variable Y.

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

Step 5: Make predictions using the model on the training data

- **model.predict(X_poly)**: Uses the trained polynomial regression model to make predictions based on the original X values transformed into polynomial features.

```
Y_pred = model.predict(X_poly)
```

Step 6: New data for prediction

- **new_data**: Represents new data points (11, 12, 13) that were not part of the original dataset.
- **new_data_poly**: Transforms the new data into polynomial features.
- **new_predictions**: Predicts the Y values for the new data using the polynomial regression model.

```
new_data = np.array([11, 12, 13]).reshape(-1, 1)
new_data_poly = poly_features.transform(new_data)
new_predictions = model.predict(new_data_poly)
```

Step 7: Display the predicted prices for new data

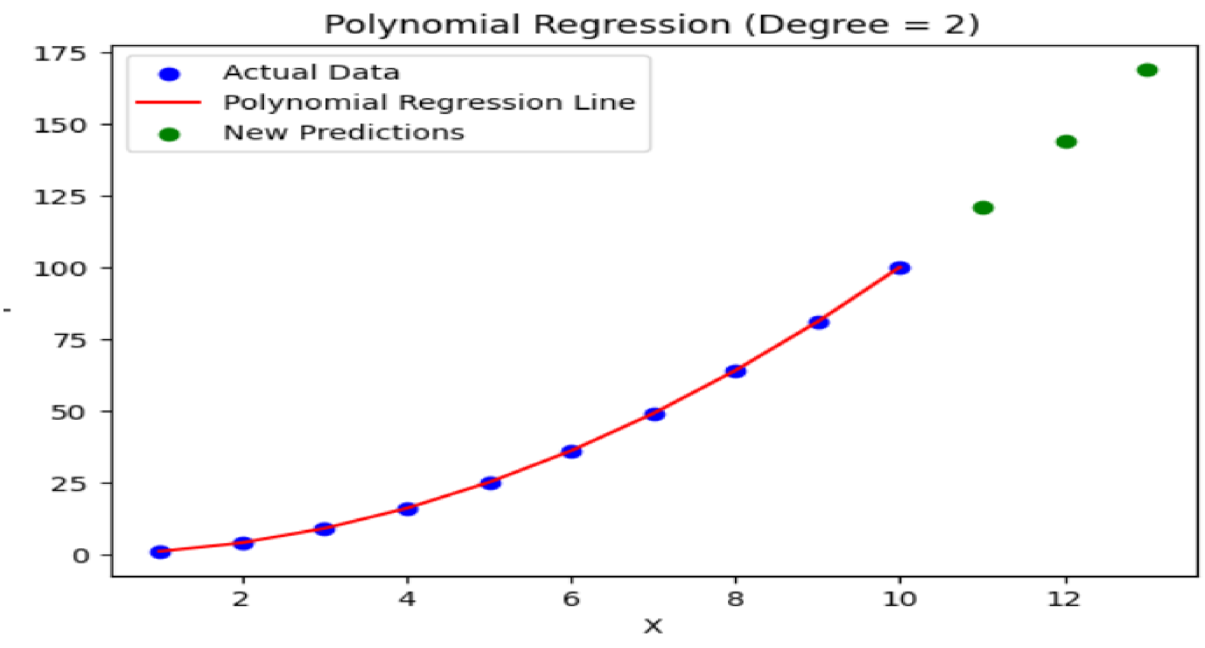
- **zip(new_data, new_predictions)**: Loops through each pair of new data points and their corresponding predicted values.
- **print()**: Displays the predictions for the new data.

```
print("Predicted values for new data:")
for data, price in zip(new_data, new_predictions):
    print(f"X: {data[0]} -> Predicted Y: {price:.2f}")
```

Step 8: Visualization

- **plt.scatter(X, Y, color="blue")**: Plots the original data points.
- **plt.plot(X, Y_pred, color="red")**: Plots the regression line (polynomial curve) based on the predicted values.
- **plt.scatter(new_data, new_predictions, color="green")**: Plots the predicted values for the new data points.
- **plt.show()**: Displays the plot.

```
plt.scatter(X, Y, color="blue", label="Actual Data")
plt.plot(X, Y_pred, color="red", label="Polynomial Regression Line")
plt.scatter(new_data, new_predictions, color="green", label="New Predictions")
plt.title("Polynomial Regression (Degree = 2)")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.show()
```



Supervised Learning-Regression

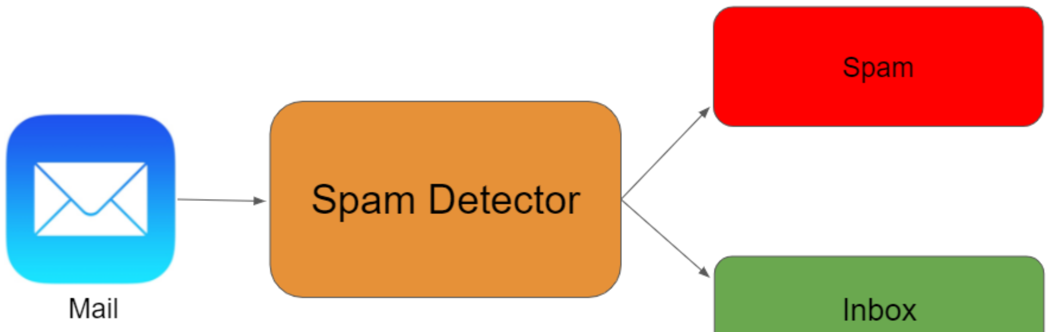
Generalization, Overfitting and Underfitting

Real-world data is inherently complex, encompassing variations, noise, and unpredictable factors. In the realm of machine learning and data science, the ultimate objective is to develop models capable of delivering accurate predictions and valuable insights when confronted with new and unseen data.

To achieve this objective, the concept of generalization plays a pivotal role. Generalization is a widely recognized technique in the world of machine learning and artificial intelligence

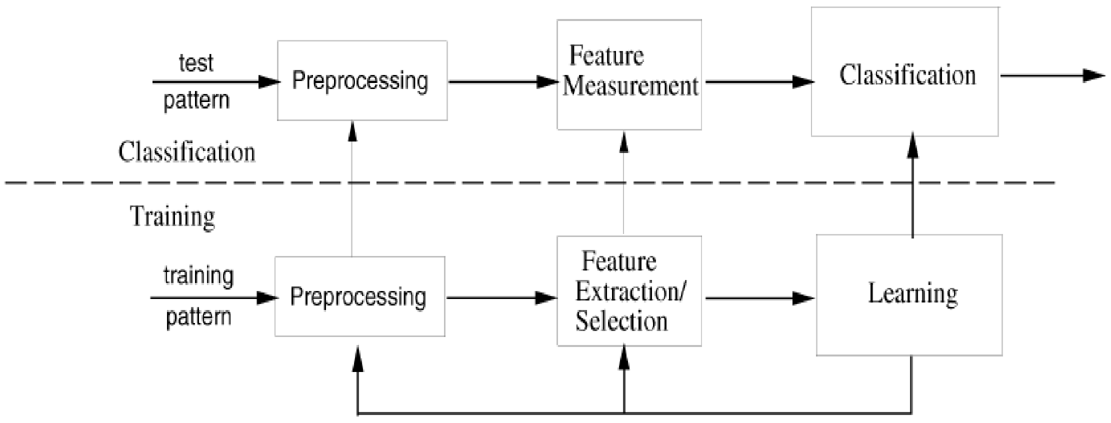
Generalization

- **Definition**: Generalization is the model's ability to perform well on new, unseen data after being trained on a given dataset. A well-generalized model learns patterns in the training data that are relevant to the larger data distribution, allowing it to make accurate predictions on data it hasn't seen.
- **Importance**: Achieving good generalization is essential in machine learning because it enables models to handle real-world scenarios accurately.
- **Influence Factors**: Generalization is influenced by the complexity of the model, the amount and quality of data, and the regularization techniques used.
- A spam email classifier is a great example of generalization in machine learning. Suppose you have a training dataset containing emails labeled as either *spam* or *not spam* and your goal is to build a model that can accurately classify incoming emails as spam or legitimate based on their content.

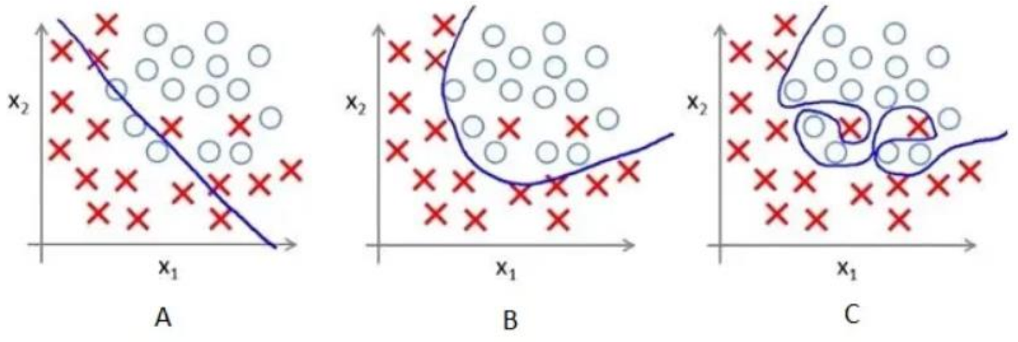


During the training phase, the machine learning algorithm learns from the set of labeled emails, extracting relevant features and patterns to make predictions. The model optimizes its parameters to minimize the training error and achieve high accuracy on the training data.

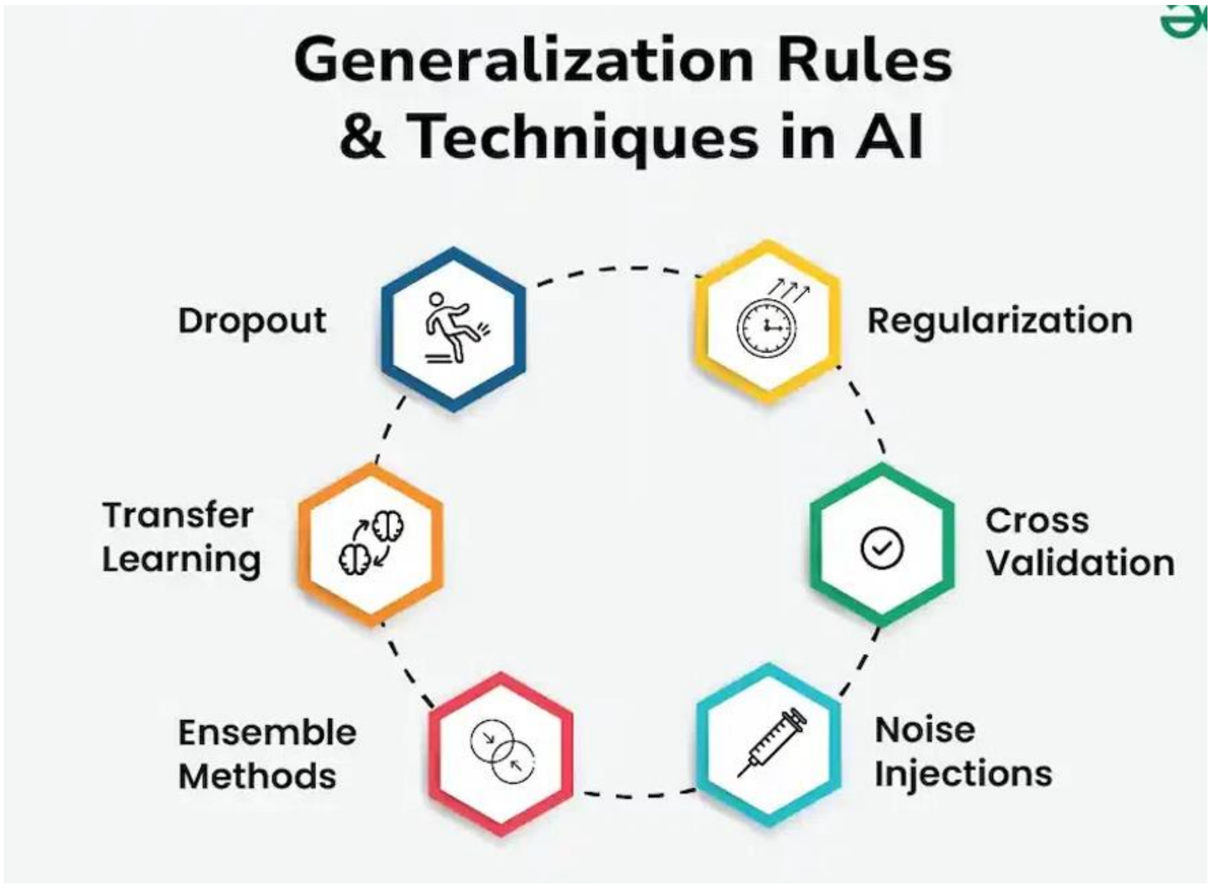
Now, the true test of the model's effectiveness lies in its ability to generalize to new, unseen emails. When new emails arrive, the model needs to accurately classify them as spam or legitimate without prior exposure to their content. This is where generalization comes in.



Generalization is a measure of how your model performs on predicting unseen data. So, it is important to come up with the best-generalized model to give better performance against future data. Let us first understand what is underfitting and overfitting, and then see what are the best practices to train a generalized model.



A: Underfitting, B: Generalized, C: Overfitting



Overfitting

Definition: Overfitting occurs when a model that performs well on the training data but poorly on unseen data because it has memorized specifics rather than generalizing the patterns.

Indicators:

- High accuracy on training data but low accuracy on validation/test data.

Causes:

- Complex models with too many parameters (e.g., deep neural networks).
- Small training datasets with insufficient diversity.

Solutions:

- **Regularization**: Techniques like L1, L2 regularization penalize large weights, encouraging simpler models.
- **Pruning**: Removing redundant parts of the model architecture, particularly for decision trees or neural networks.

    - **Cross-validation**: Using techniques like k-fold cross-validation to better evaluate model performance.
    - **Early Stopping**: Ending the training process when validation error starts increasing.
    - **Ensembling**: Combining multiple models

Underfitting

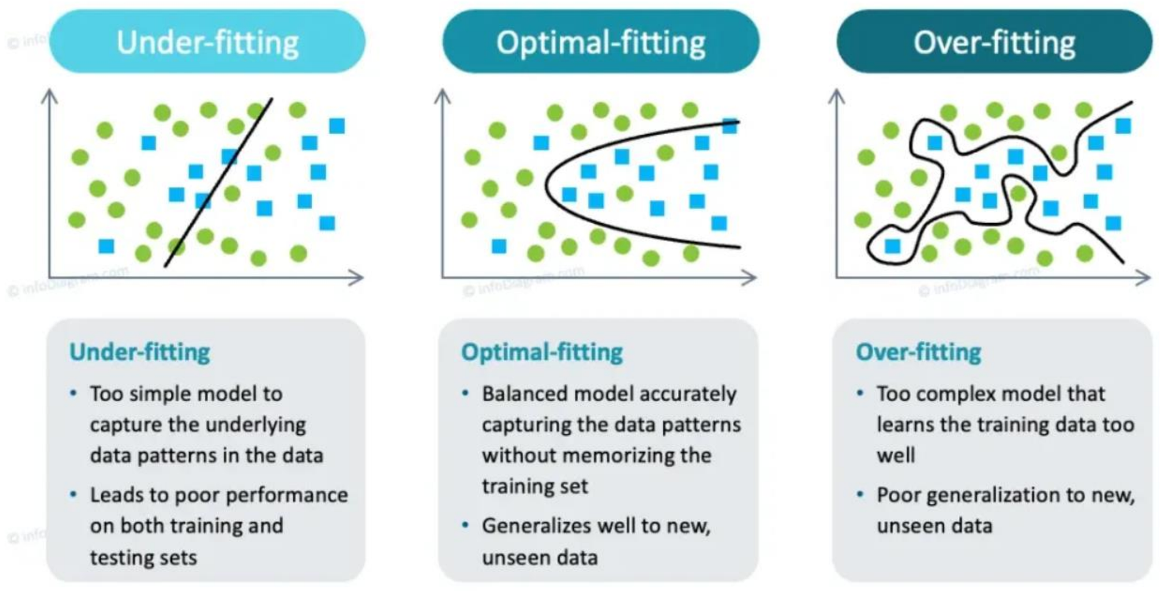
Definition: Underfitting occurs when a model is too simplistic to capture the underlying structure of the data. It fails to fit the training data and, as a result, also performs poorly on new data.

- **Indicators**:
    - Low accuracy on both training and validation/test datasets.
- **Causes**:
    - Models that are too simple (e.g., linear regression for non-linear data).
    - Insufficient training time.
    - Lack of relevant features in the data.

Solutions:

- **Model Complexity**: Use more complex models that can capture patterns (e.g., neural networks, ensemble methods).
- **Feature Engineering**: Add or transform features to make patterns more accessible.
- **Increasing Training Time**: Train the model longer to allow it to learn complex patterns.
- **Hyperparameter Tuning**: Adjust parameters to allow for a more flexible fit to the data.
- **Increasing Data Quality/Quantity**: More or better-quality data can help the model learn more about the true data distribution.

Aspect	Overfitting	Underfitting
Model Fit	Fits too closely to the training data	Fails to fit the training data well
Training Accuracy	Very high	Low or moderate
Test Accuracy	Low	Low
Example Model	Deep neural network on small dataset	Linear regression for complex data
Solution	Regularization, simpler model, early stopping	More complex model, add features, tuning



Under-fitting
- Too simple model to capture the underlying data patterns in the data
- Leads to poor performance on both training and testing sets

Optimal-fitting
- Balanced model accurately capturing the data patterns without memorizing the training set
- Generalizes well to new, unseen data

Over-fitting
- Too complex model that learns the training data too well
- Poor generalization to new, unseen data

Experiment 4. Build a linear regression model using python on given data set by

i. Prepare the data for ML model.

ii. Splitting Training data and Test data.

iii. Evaluate the model (intercept and slope).

iv. Visualize the training set and testing set using Matplotlib, Seaborn.

v. predicting the test set result.

vi. compare actual output values with predicted values.

Step 1: Import Libraries and Load the Dataset

We'll load the Boston Housing dataset, which is accessible online. We'll load it directly into a DataFrame using its URL.

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset from a URL
url = "https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"
data = pd.read_csv(url)

# Display the first few rows of the dataset
print(data.head())
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

Step 2: Prepare Data for Modeling

The dataset contains various columns, with MEDV (Median value of owner-occupied homes) as the target variable. We'll use RM (average number of rooms per dwelling) as the input feature to predict MEDV.

```
# Step 2: Prepare data for ML model
# We are predicting the target variable 'medv' (median value of owner-occupied homes in $1000s)
# and using 'rm' (average number of rooms per dwelling) as the input feature.
X = data[['rm']]
Y = data['medv']
```

Step 3: Train and Test Split

We'll split the data into training and testing sets for model training and evaluation

```
# Step 3: Splitting Training Data and Test Data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Step 4: Train and Evaluate the Model

Train a linear regression model on the training data and evaluate it by calculating the intercept and slope.

```
# Step 4: Train the model and evaluate the model (Intercept and Slope)
model = LinearRegression()
model.fit(X_train, Y_train)

# Model parameters
intercept = model.intercept_
slope = model.coef_[0]

print("Model Intercept:", intercept)
print("Model Slope:", slope)
```

```
Model Intercept: -36.24631889813795
Model Slope: 9.348301406497727
```

Step 5: Visualize and Compare Results

We'll visualize both training and test predictions and compare actual values with predicted values.

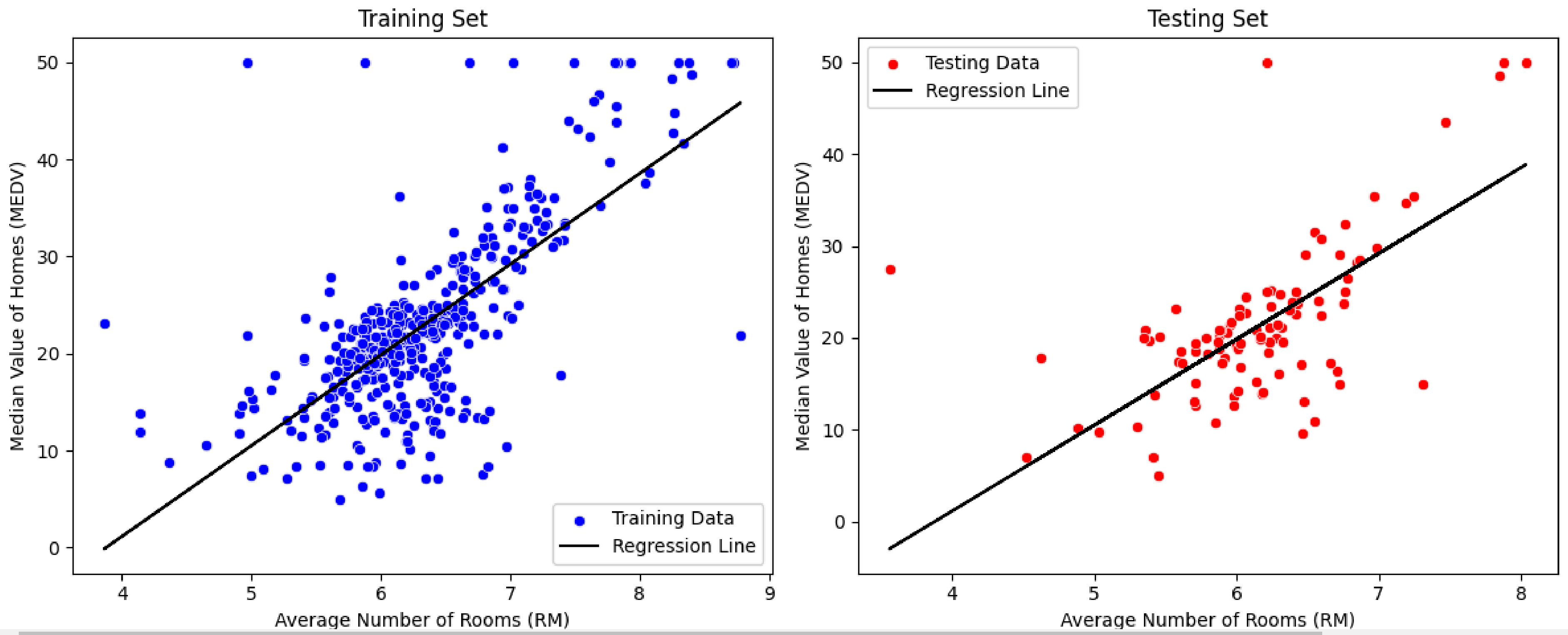
```
# Step 5: Visualize the Training Set and Testing Set
plt.figure(figsize=(12, 5))

# Training set visualization
```

```
plt.subplot(1, 2, 1)
plt.title("Training Set")
sns.scatterplot(x=X_train['rm'], y=Y_train, color='blue', label="Training Data")
plt.plot(X_train, model.predict(X_train), color='black', label="Regression Line")
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median Value of Homes (MEDV)")
plt.legend()

# Testing set visualization
plt.subplot(1, 2, 2)
plt.title("Testing Set")
sns.scatterplot(x=X_test['rm'], y=Y_test, color='red', label="Testing Data")
plt.plot(X_test, model.predict(X_test), color='black', label="Regression Line")
plt.xlabel("Average Number of Rooms (RM)")
plt.ylabel("Median Value of Homes (MEDV)")
plt.legend()

plt.tight_layout()
plt.show()
```



```
# Predict the Test Set Results
Y_pred = model.predict(X_test)

# Compare Actual Output Values with Predicted Values
comparison = pd.DataFrame({'Actual': Y_test.values, 'Predicted': Y_pred})
print(comparison.head())

# Calculate model evaluation metrics
mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

```
   Actual  Predicted
0    23.6  23.732383
1    32.4  26.929502
2    13.6  19.684568
3    22.8  20.451129
4    16.1  22.619935
Mean Squared Error: 46.144775347317264
Mean Absolute Error: 4.478335832064149
```