# What Is Logistic Regression?

Logistic Regression is a **supervised learning algorithm** used to **predict the probability** that an observation belongs to a particular class.

It predicts **discrete outcomes**, not continuous values.

# Where Is It Used?

Some **real-world examples**:

|  |  |
| --- | --- |
| Problem | Description |
| Email Spam Detection | 1 = Spam, 0 = Not Spam |
| Disease Diagnosis | 1 = Has disease, 0 = No disease |
| Customer Churn | 1 = Will leave, 0 = Will stay |
| Credit Approval | 1 = Approved, 0 = Denied |
| Pass/Fail Prediction | 1 = Pass, 0 = Fail |

# CODE

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Example data

data = pd.DataFrame({

    'Hours\_Studied': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

    'Passed': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]

})

# Split data

X = data[['Hours\_Studied']]

y = data['Passed']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Predict

y\_pred = model.predict(X\_test)

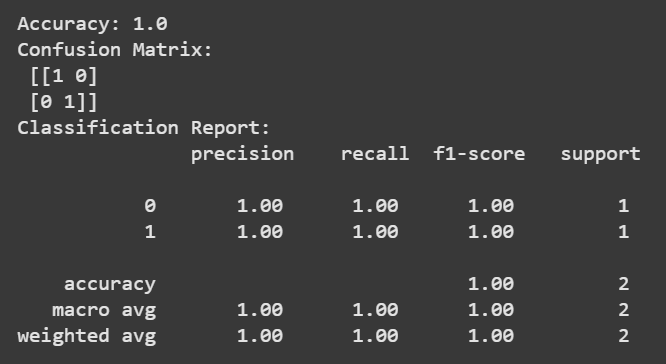
y\_prob = model.predict\_proba(X\_test)[:, 1]

# Evaluate

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))



# Regularization in Logistic Regression

Just like in Linear Regression, **Logistic Regression** also supports **L1 (Lasso)** and **L2 (Ridge)** regularization to prevent overfitting.

model = LogisticRegression(penalty='l2', C=1.0)  # L2 (default)

# Bias and Variance

## Bias

**Bias** measures how far the model’s predictions are from the real (true) values.

* High bias means the model makes **strong assumptions** and **simplifies too much**.
* It doesn’t capture the true relationship in the data → **Underfitting**.

**Why it happens:**

* Model is too simple
* Assumes a linear relationship when data is non-linear
* Not enough training or incorrect assumptions

**Common examples in ML:**

* Using **Linear Regression** on data that has a curved trend.
* Using too few features or ignoring interactions between variables

## Variance

**Variance** measures how much the model’s predictions change if you train it on different samples of data.

* High variance means the model is **too sensitive** to the training data.
* It learns not just the real pattern but also the noise → **Overfitting**.

**Why it happens:**

* Model is too complex
* Memorizes noise in training data
* Too many parameters or overfitting

**Common examples in ML:**

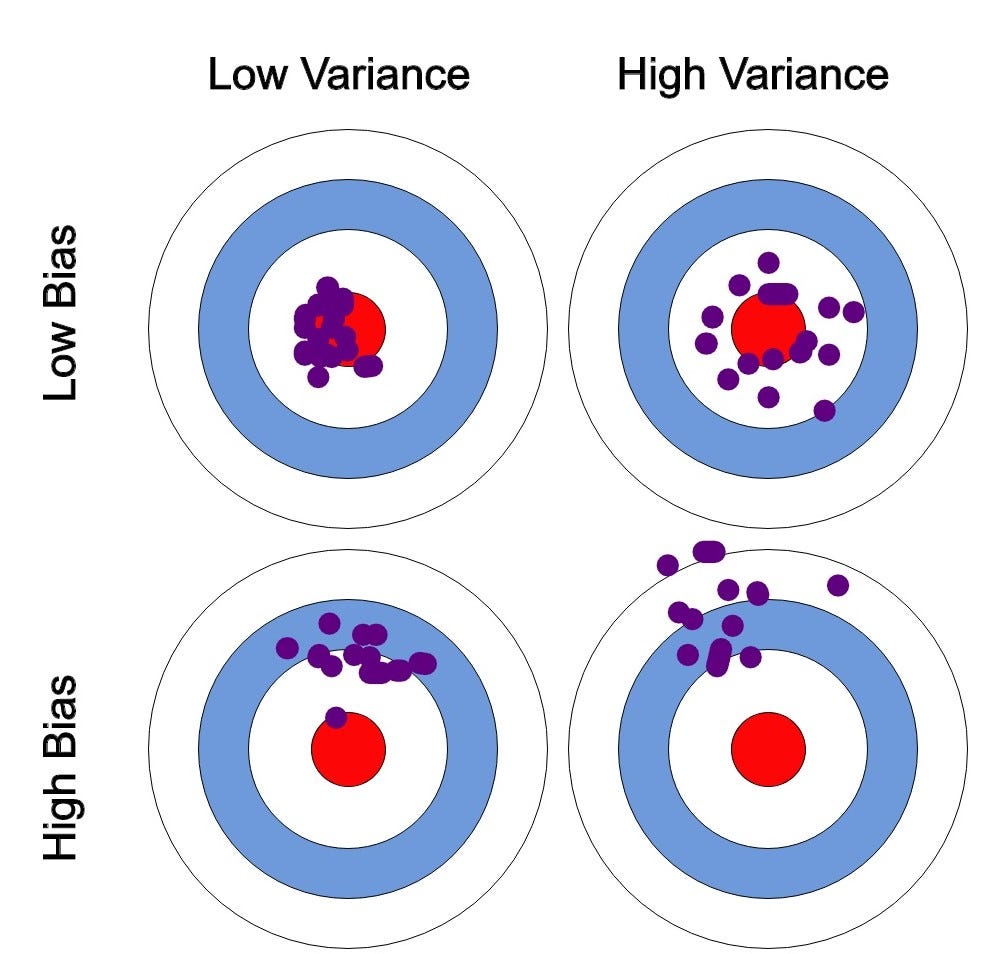
* Deep neural networks trained on small datasets.
* Decision trees grown without pruning.

## The Bias-Variance Tradeoff

There’s a balance between **bias** and **variance**:

* **High bias →** The model is *too simple* and can’t capture the complexity of data (underfitting).
* **High variance →** The model is *too complex* and learns noise (overfitting).

We want the model to sit in the **middle**, where both bias and variance are low — meaning it generalizes well.



# Support Vector Machine

**Support Vector Machine (SVM)** is a **supervised machine learning algorithm** used for **classification** and **regression** tasks.  
It tries to find the **best boundary (hyperplane)** that separates different classes in the data.

## Intuitive Explanation

Imagine you have two types of points (e.g., red and blue) on a graph.  
You want to draw a line that **divides** them correctly.

There can be many possible lines, but **SVM finds the best one** — the one with the **maximum margin** between the classes.

## The Key Idea: Maximum Margin Classifier

* A **hyperplane** is a decision boundary that separates data points of different classes.
* SVM tries to find the **hyperplane with the largest margin** — the greatest distance between itself and the nearest points from each class.
* The **closest points** to the hyperplane are called **Support Vectors** — they “support” the boundary.

## Types of SVM

| **Type** | **Description** |
| --- | --- |
| **Linear SVM** | When data is linearly separable (can be divided with a straight line). |
| **Non-linear SVM** | When data can’t be separated by a straight line — we use a kernel trick to transform data into a higher dimension where it becomes separable. |

## When to Use Which Kernel

SVM can’t always separate data with a straight line that’s where **kernels** come in.

|  |  |  |
| --- | --- | --- |
| Kernel | When to Use | Description / Example |
| Linear | When data is linearly separable | Works well with many features (e.g. text data like spam detection) |
| Polynomial | When data has curved boundaries | Example: Circle vs. outer points |
| RBF (Radial Basis Function) *(most common)* | When data is **non-linear** and you don’t know the pattern | Handles complex shapes — e.g., concentric circles |

## The “C” Parameter (Regularization)

C is one of the most **important hyperparameters** in SVM.

|  |  |  |
| --- | --- | --- |
| C Value | Meaning | Effect on Model |
| Small (e.g., 0.01 – 1) | Model allows more misclassifications | Higher bias, smoother boundary (less overfitting) |
| Large (e.g., 10 – 1000) | Model tries to classify all points correctly | Lower bias, higher variance (can overfit) |

## Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **Meaning** | **Default** | **Explanation** |
| kernel | Type of kernel function | 'rbf' | 'linear', 'poly', 'rbf', 'sigmoid' |
| C | Regularization parameter | 1.0 | Controls margin size and misclassification. Small C → wider margin (allow errors). |
| gamma | Kernel coefficient (for RBF/poly) | 'scale' | Controls influence of single training example. High gamma → more complex model. |
| degree | Degree for polynomial kernel | 3 | Used when kernel='poly'. |
| probability | Whether to enable probability estimates | False | Set True to get predict\_proba() output. |
| decision\_function\_shape | Return shape of output | 'ovr' | 'ovr' (one-vs-rest) for multi-class classification. |

## CODE

