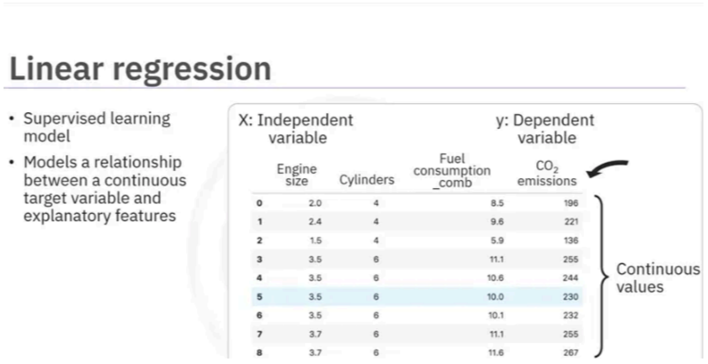
Machine Learning with Python (M1)

Created	@December 4, 2025 11:40 PM
Module Code	IBMM01: Introduction to Machine Learning

Module 2: Linear and Logistic Regression

Overview of Regression

Regression is a supervised machine learning technique used to model the relationship between a **continuous target variable** (like CO_2 emissions) Continuous value refers to a type of data that can take on an infinite number of values within a given range. and one or more **predictor features** (like engine size, fuel consumption, etc.).



Linear regression

- Supervised learning model
- Models a relationship between a continuous target variable and explanatory features

X: Independent variable — y: Dependent variable — Continuous values

	Engine size	Cylinders	Fuel consumption _comb	CO_2 emissions
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

Types of Regression

1. **Simple Regression**
    - Uses **one independent variable** to predict the target.
    - Can be **linear** (straight-line relationship) or **nonlinear** (curved relationship).
    - Example: Predicting CO_2 emissions using only *engine size*.

2. **Multiple Regression**
    - Uses **two or more independent variables**.
    - Can also be **linear or nonlinear**.
    - Example: Predicting CO_2 emissions using *engine size + cylinders + fuel consumption*.

Applications of Regression

Regression is used whenever the goal is to predict a **continuous numeric value**, for example:

- Predicting CO_2 emissions of a new car.
- Forecasting sales using customers, leads, etc.
- Predicting house prices based on size, bedrooms, etc.
- Predicting machine maintenance needs.
- Estimating employee income from education, experience, etc.
- Predicting rainfall or wildfire severity.
- Predicting spread of infectious disease.
- Estimating likelihood of diseases like diabetes or heart disease.

Regression Algorithms

- **Classical:** Linear regression, polynomial regression.
- **Modern ML:** Random Forest, XGBoost, KNN, SVM, Neural Networks.

Key Takeaways

- Regression predicts **continuous values**.
- **Simple regression** → one feature
- **Multiple regression** → multiple features
- Both can be **linear or nonlinear** depending on the relationship.

- Regression has **many real-world applications** across finance, healthcare, environment, retail, etc.

Intro to Simple Linear Regression

What is Simple Linear Regression?

Simple Linear Regression is a machine learning technique used to model a **linear relationship** between:

- **One independent variable (predictor)** → e.g., engine size
- **One dependent variable (target)** → e.g., CO_2 emissions

It predicts a **continuous numeric value**.

1. Plotting Data

If you plot:

- **x-axis:** engine size
- **y-axis:** CO_2 emissions

You will usually see a **pattern** (often a line-like trend). If the relationship looks roughly straight-line, simple linear regression can model it.

2. Best-Fit Line

Regression finds the **best possible straight line** that fits the data.

The equation is:

$$\hat{y} = \theta_0 + \theta_1 x_1$$

Where:

- **$\hat{y}$ (y-hat)** = predicted value
- **x_1** = input (engine size)
- **θ_0** = intercept (bias)
- **θ_1** = slope (coefficient)

The slope tells how much CO_2 emissions change when engine size increases by 1 unit.

For each point:

- **Residual error** = difference between actual value and predicted value

  (vertical distance from the point to the line)

MSE=mean squared error

OLS (Ordinary Least Squares) tries to find **θ_0 and θ_1** that make MSE as **small as possible**.

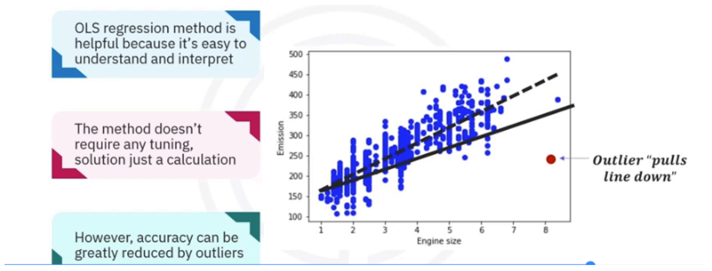
$$\theta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

This measures how x and y **move together** (covariance).

This measures how spread out x is (variance).

👉 If x goes up and y goes up → slope is positive
👉 If x goes up and y goes down → slope is negative


OLS regression method is helpful because it's easy to understand and interpret

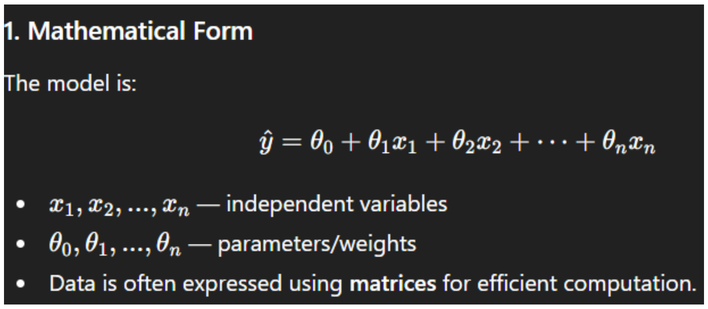
The method doesn't require any tuning, solution just a calculation

However, accuracy can be greatly reduced by outliers

Multiple Linear Regression

Multiple Linear Regression (MLR) is an extension of simple linear regression.

Instead of one independent variable, MLR uses **two or more predictors** to estimate a dependent variable.



1. Mathematical Form

The model is:

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

- $x_1, x_2, ..., x_n$ — independent variables
- $\theta_0, \theta_1, ..., \theta_n$ — parameters/weights
- Data is often expressed using **matrices** for efficient computation.

---

2. Why Use Multiple Linear Regression?

Compared to simple linear regression:

- It captures more complex relationships.
- It measures the **individual effect** of each variable.
- It usually makes **better predictions**, e.g., predicting CO_2 emissions using:
  - engine size
  - number of cylinders
  - fuel consumption

---

3. Categorical Variables

To include categorical features:

- **Binary variables** → use 0/1 encoding
- **Multi-class variables** → convert into multiple boolean (one-hot) columns

---

4. Pitfalls of MLR

1. **Overfitting**

   Too many variables cause the model to memorize training data.

2. **Impossible or unrealistic what-if scenarios**

Changing a single variable while others are correlated may be unrealistic.

3. **Multicollinearity**

   When independent variables are correlated, the model becomes unstable.

   → Solution: **Remove redundant correlated features**

---

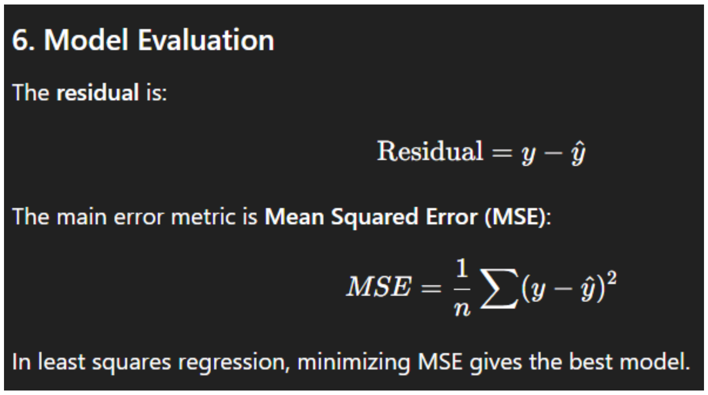
5. What-If Scenarios

Used to predict how outcomes change when input features change.

However, they fail when:

- The scenario is unrealistic
- Prediction involves values far from training data
- Correlated variables prevent isolating one feature's effect

---



6. Model Evaluation

The **residual** is:

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

The main error metric is **Mean Squared Error (MSE)**:

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

In least squares regression, minimizing MSE gives the best model.

---

7. How Parameters Are Estimated

Two common methods:

A. Ordinary Least Squares (OLS)

- Uses linear algebra to calculate optimal θ directly.
- Fast for small/medium datasets.

B. Optimization Methods

- Adjust θ iteratively to reduce error.

- Best for large datasets.
- Example: **Gradient Descent**

---

8. Key Takeaways

- MLR uses **multiple features** to predict a target more accurately than simple linear regression.
- Must carefully select variables to avoid overfitting and multicollinearity.
- Used widely for predictions and analyzing relationships between variables.
- Parameters can be found using **OLS** or **gradient descent**.

Polynomial & Nonlinear Regression

1. What is Nonlinear Regression?

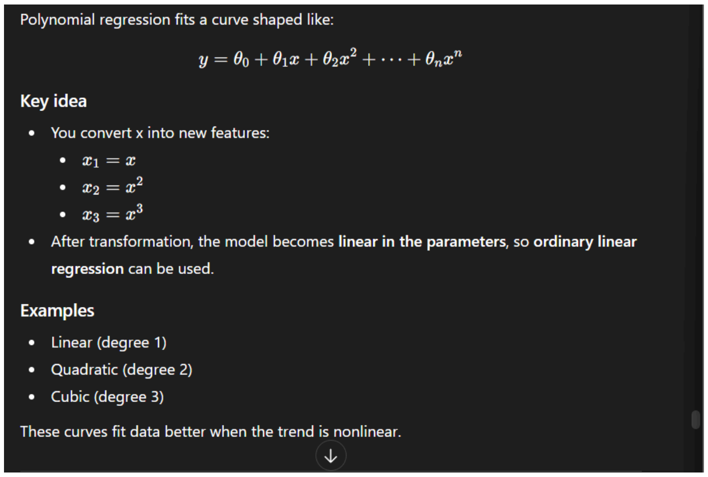
Nonlinear regression models the relationship between input variables (x) and output (y) using **nonlinear equations** such as:

- Polynomial functions
- Exponential functions
- Logarithmic functions
- Sinusoidal (periodic) functions

A straight line **cannot** capture these complex relationships, so nonlinear models are used.

When is nonlinear regression needed?

- When data shows **curved patterns**, not a straight-line trend
- When growth is **exponential**
- When effects **diminish** after a certain point (logarithmic)
- When data follows **seasonal/periodic** patterns



Polynomial regression fits a curve shaped like:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$

Key idea

- You convert x into new features:
  - $x_1 = x$
  - $x_2 = x^2$
  - $x_3 = x^3$
- After transformation, the model becomes **linear in the parameters**, so **ordinary linear regression** can be used.

Examples

- Linear (degree 1)
- Quadratic (degree 2)
- Cubic (degree 3)

These curves fit data better when the trend is nonlinear.

3. Overfitting in Polynomial Regression

- A very high-degree polynomial can **fit every point perfectly**, including noise.
- This is **overfitting**: memorizing data instead of learning the true pattern.
- The goal is to capture the **trend**, not every small fluctuation.

4. Why Polynomial Regression Is Still "Linear"

Polynomial regression is nonlinear in **x**,

but **linear in the parameters** θ0,θ1,θ2...

Therefore, it can be solved using **linear regression techniques**.

5. Non-Polynomial Nonlinear Regression

Some relationships cannot be modeled even with polynomials.

Examples:

a) Exponential growth

Used for:

- GDP growth
- Population
- Compound interest

y^=θ0+θ1e^x

b) Logarithmic (diminishing returns)

After a point, extra input gives smaller gains

(Example: productivity drops after too many work hours)

c) Periodicity (sinusoidal)

Used for:

- Weather patterns
- Seasonal rainfall
- Temperature cycles

6. How to Choose a Nonlinear Model

A. Visual inspection

- Plot data
- See if it looks:
  - Linear
  - Curved
  - Exponential
  - Logarithmic
  - Seasonal

B. Try different models and compare performance

Plot predictions vs actual values.

7. How to Fit Nonlinear Models

Two main cases:

Case 1: You know the model equation

Example: exponential θ0+θ1ex\theta_0 + \theta_1 e^xθ0+θ1ex

📌 Use **gradient descent** or other optimization techniques to find parameters.

Case 2: You don't know the best equation

Use machine learning models that naturally capture nonlinear patterns:

- Decision trees
- Random forests
- Gradient boosting
- Neural networks
- k-Nearest Neighbors
- Support Vector Machines

These models automatically learn complex nonlinear shapes.

⭐ Final Takeaway

✔ Polynomial regression = nonlinear in x but linear in coefficients.

✔ Nonlinear regression = models that involve non-linear functions (exp, log, sin, etc).

✔ Polynomial regression can overfit if degree is too high.

✔ Choose model by visually inspecting data and testing patterns.

✔ For complex unknown relationships, use ML models (trees, SVMs, neural network

Logistic Regression

1. What Logistic Regression Is

- A **statistical model** used when the **target variable is binary** (0/1, yes/no, true/false).
- Predicts the **probability** that an observation belongs to class 1.
- In machine learning, logistic regression is a **binary classifier** and a **probability predictor**.

---

2. When Logistic Regression Is a Good Choice

Use logistic regression when:

1. **The dependent variable is binary**

   Example: churn (yes/no), disease (present/absent).

2. **You need probabilities**, not just class labels

   Example: probability a customer will leave.

3. **You want interpretability**

   Coefficients show the *importance* and *impact* of each feature.

4. **Data is linearly separable**

   Decision boundary is a line, plane, or hyperplane.

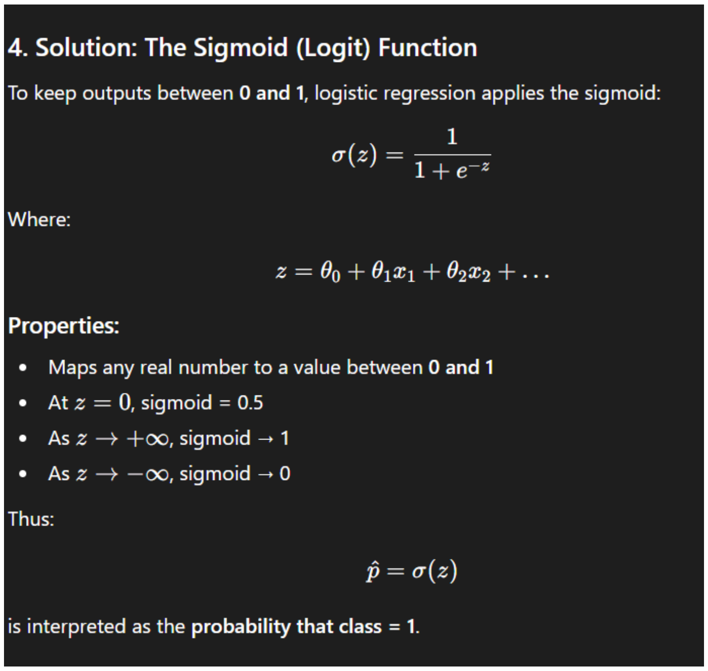
---

3. Why Linear Regression Fails for Classification

If you try to use linear regression for classification:

- Predictions can be **less than 0 or more than 1** → invalid probabilities.
- A simple threshold (e.g., 0.5) becomes a **hard step function** → too abrupt.
- Cannot distinguish differences once the prediction crosses 0.5.
- Does **not** output proper probabilities.

So linear regression is unsuitable.


4. Solution: The Sigmoid (Logit) Function

To keep outputs between **0 and 1**, logistic regression applies the sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Where:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

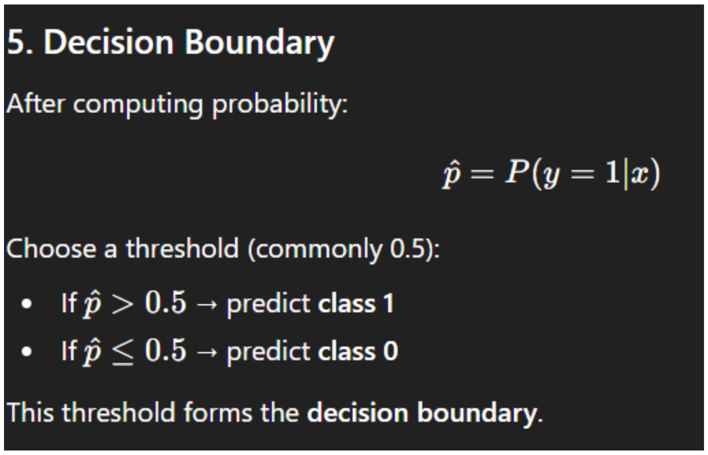
Properties:

- Maps any real number to a value between **0 and 1**
- At $z = 0$, sigmoid = 0.5
- As $z \to +\infty$, sigmoid → 1
- As $z \to -\infty$, sigmoid → 0

Thus:

$$\hat{p} = \sigma(z)$$

is interpreted as the **probability that class = 1**.


5. Decision Boundary

After computing probability:

$$\hat{p} = P(y = 1 | x)$$

Choose a threshold (commonly 0.5):

- If $\hat{p} > 0.5$ → predict **class 1**
- If $\hat{p} \leq 0.5$ → predict **class 0**

This threshold forms the **decision boundary**.

Logistic regression is widely used for:

- Predicting **heart attack risk** using age, BMI, gender
- Diagnosing **diabetes** using medical measurements
- Predicting **customer churn** (telecom example)
- **Loan default** prediction
- **Process/product** failure probability

7. Example: Customer Churn

Dataset includes:

- Customer demographics (age, gender)
- Account info (contract type, monthly charges)
- Services used
- Churn column (1 = left, 0 = stayed)

Goal: Predict p^=P(churn=1 ∣ x)

If

P(churn=1)=0.8

Then:

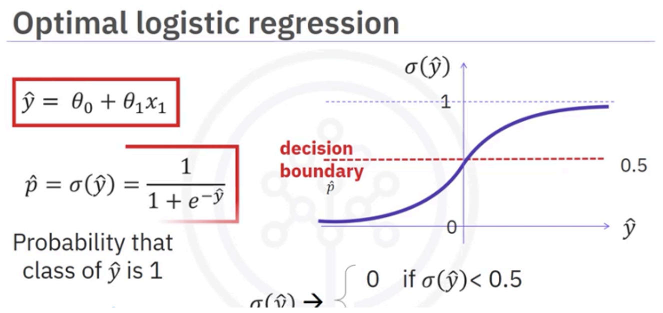
P(stay)=1−0.8=0.2

- Logistic regression predicts **probability** of binary outcomes.
- It uses the **sigmoid** function to convert any linear combination of features into a value between 0 and 1.
- You classify using a **threshold** like 0.5.
- It is ideal when you want **probabilistic outputs**, **interpretability**, and a **binary target**.

Training a Logistic Regression Model

1. Goal of Logistic Regression Training

- Find the **best parameter vector θ (theta)**.
- These parameters map input features → predicted class probabilities.
- Goal: **minimize prediction error** on training data.
-


Optimal logistic regression
$\hat{y} = \theta_0 + \theta_1 x_1$
$\hat{p} = \sigma(\hat{y}) = \frac{1}{1+e^{-\hat{y}}}$
Probability that class of $\hat{y}$ is 1
decision boundary
$\hat{p}$
$\sigma(\hat{y})$
1
0.5
0
$\hat{y}$
0 if $\sigma(\hat{y})$< 0.5
$\sigma(\hat{y}) \rightarrow$

⭐ 2. Steps in Training a Logistic Regression Model

1. **Choose initial parameters θ**

   – Usually random.

2. **Predict probabilities** p^=σ(θ^T x)

   using the sigmoid function.

3. **Calculate error using a cost function**

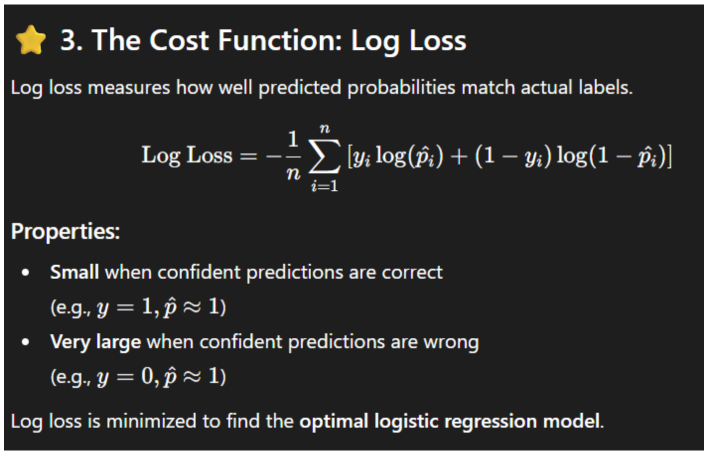
   – Logistic regression uses **log loss**.

4. **Update θ to reduce error**

   – Using gradient descent or SGD.

5. **Repeat** until:

   - log loss becomes small enough, OR
   - max iterations is reached.



⭐ 3. The Cost Function: Log Loss

Log loss measures how well predicted probabilities match actual labels.

$$\text{Log Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)\right]$$

Properties:

- **Small** when confident predictions are correct

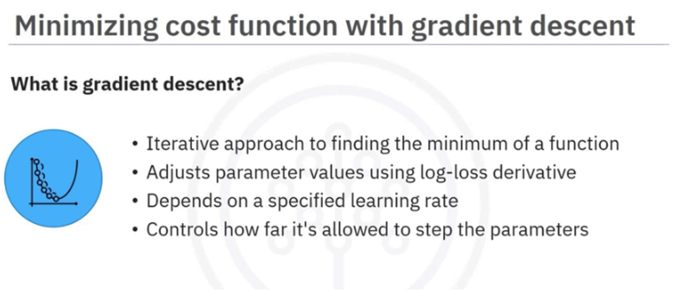
  (e.g., $y = 1, \hat{p} \approx 1$)

- **Very large** when confident predictions are wrong

  (e.g., $y = 0, \hat{p} \approx 1$)

Log loss is minimized to find the **optimal logistic regression model**.

⭐ 4. Gradient Descent (GD)



Minimizing cost function with gradient descent

What is gradient descent?

- Iterative approach to finding the minimum of a function
- Adjusts parameter values using log-loss derivative
- Depends on a specified learning rate
- Controls how far it's allowed to step the parameters

Gradient descent finds the best θ by repeatedly moving in the direction of **steepest descent** of the cost function.

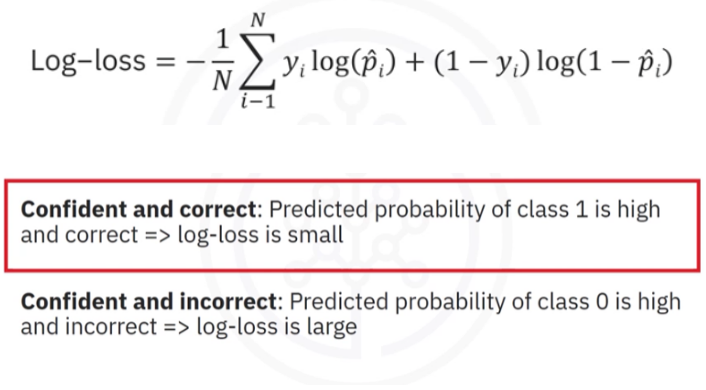
How it works:

- Compute gradient of log loss w.r.t parameters.
- Update θ:

$\theta = \theta - \alpha \nabla J(\theta)$

Where:

- α = learning rate
- $J(\theta)$ = log loss
-



$$\text{Log-loss} = -\frac{1}{N}\sum_{i-1}^{N} y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i)$$

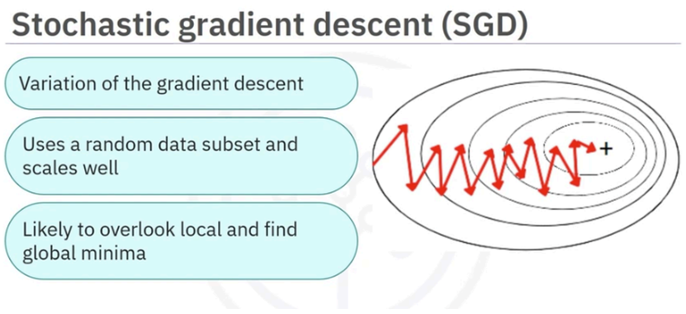
Confident and correct: Predicted probability of class 1 is high and correct => log-loss is small

Confident and incorrect: Predicted probability of class 0 is high and incorrect => log-loss is large

Key Points:

- Uses **entire dataset** on every update → accurate but **slow** for large datasets.
- If learning rate is too large → can overshoot minimum.
- If learning rate is too small → training becomes slow.

⭐ 5. Stochastic Gradient Descent (SGD)


Stochastic gradient descent (SGD)
Variation of the gradient descent
Uses a random data subset and scales well
Likely to overlook local and find global minima

SGD improves speed by:

- Using **random subsets** (mini-batches or single samples) instead of the whole dataset.

Advantages:

- Much **faster** on large datasets.
- More likely to find the **global** minimum (avoids local minima sometimes).
- Scales well with big data.

Drawbacks:

- Less stable; updates can "bounce around" the minimum.
- Convergence can be noisy.

Improvements:

- Decrease learning rate over time.
- Increase batch size as convergence nears.

Feature	Gradient Descent	Stochastic Gradient Descent
Uses entire dataset	Yes	No (random subset)
Speed	Slow on big data	Very fast
Accuracy	More stable	More noise
Convergence	Precise	Jumpy but faster
Best for	Small/medium data	Large-scale data

- Logistic regression training seeks **parameters that minimize log loss**.
- Log loss **rewards correct, confident predictions** and **punishes wrong, confident predictions**.
- **Gradient Descent**: accurate but slow for large datasets.
- **SGD**: faster, scalable, slightly noisier but often finds global minimum faster.

Comparing different regression types

Model Name	**Description**	**Code Syntax**
Simple linear regression	**Purpose:** To predict a dependent variable based on one independent variable.**Pros:** Easy to implement, interpret, and efficient for small datasets.**Cons:** Not suitable for complex relationships; prone to underfitting.**Modeling equation:** y = b0 + b1x	1. from sklearn.linear_model import LinearRegression 2. model = LinearRegression() 3. model.fit(X, y)Copied!Wrap Toggled!
Polynomial regression	**Purpose:** To capture nonlinear relationships between variables.**Pros:** Better at fitting nonlinear data compared to linear regression.**Cons:** Prone to overfitting with high-degree polynomials.**Modeling equation:** y = b0 + b1x + b2x2 + ...	1. from sklearn.preprocessing import PolynomialFeatures 2. from sklearn.linear_model import LinearRegression 3. poly = PolynomialFeatures(degree=2) 4. X_poly = poly.fit_transform(X) 5. model = LinearRegression().fit(X_poly, y)Copied!Wrap Toggled!

Model Name	Description	Code Syntax
Multiple linear regression	**Purpose:** To predict a dependent variable based on multiple independent variables.**Pros:** Accounts for multiple factors influencing the outcome.**Cons:** Assumes a linear relationship between predictors and target.**Modeling equation:** y = b0 + b1×1 + b2×2 + ...	1. from sklearn.linear_model import LinearRegression 2. model = LinearRegression() 3. model.fit(X, y)Copied!Wrap Toggled!
Logistic regression	**Purpose:** To predict probabilities of categorical outcomes.**Pros:** Efficient for binary classification problems.**Cons:** Assumes a linear relationship between independent variables and log-odds.**Modeling equation:** log(p/(1-p)) = b0 + b1×1 + ...	1. from sklearn.linear_model import LogisticRegression 2. model = LogisticRegression() 3. model.fit(X, y)Copied!Wrap Toggled!

Associated functions commonly used

Function/Method Name	Brief Description	Code Syntax
train_test_split	Splits the dataset into training and testing subsets to evaluate the model's performance.	1. from sklearn.model_selection import train_test_split 2. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)Copied!Wrap Toggled!
StandardScaler	Standardizes features by removing the mean and scaling to unit variance.	1. from sklearn.preprocessing import StandardScaler 2. scaler = StandardScaler() 3. X_scaled = scaler.fit_transform(X)Copied!Wrap Toggled!
log_loss	Calculates the logarithmic loss, a performance metric for classification models.	1. from sklearn.metrics import log_loss 2. loss = log_loss(y_true, y_pred_proba)Copied!Wrap Toggled!
mean_absolute_error	Calculates the mean absolute error between	

Function/Method Name	Brief Description	Code Syntax
	actual and predicted values.	1. from sklearn.metrics import mean_absolute_error 2. mae = mean_absolute_error(y_true, y_pred)Copied!Wrap Toggled!
mean_squared_error	Computes the mean squared error between actual and predicted values.	1. from sklearn.metrics import mean_squared_error 2. mse = mean_squared_error(y_true, y_pred)Copied!Wrap Toggled!
root_mean_squared_error	Calculates the root mean squared error (RMSE), a commonly used metric for regression tasks.	1. from sklearn.metrics import mean_squared_error 2. import numpy as np 3. rmse = np.sqrt(mean_squared_error(y_true, y_pred))Copied!Wrap Toggled!
r2_score	Computes the R-squared value, indicating how well the model explains the variability of the target variable.	1. 1 2. 2 1. from sklearn.metrics import r2_score 2. r2 =