Universität Konstanz

02 | Introduction to Machine Learning

Giordano De Marzo

https://giordano-demarzo.github.io/

Deep Learning for Social Sciences

Universität Konstanz

Outline

1. The Machine Learning Paradigm
2. Evaluating Models
3. The Perceptron

Universität Konstanz

The Machine Learning Paradigm

Learning from Data

Machine learning represents a paradigm shift in how computers solve problems.

Traditional programming: Humans write explicit rules for the computer to follow

Machine learning: Computers discover patterns and rules from examples in data

This data-driven approach can be categorized into **two main types**:

- **Unsupervised learning:** Finding patterns in unlabeled data
- **Supervised learning:** Learning from labeled examples to make predictions

Supervised learning further divides into:

- **Classification:** Predicting categories (spam/not spam, dog/cat)
- **Regression:** Predicting numerical values (price, temperature)

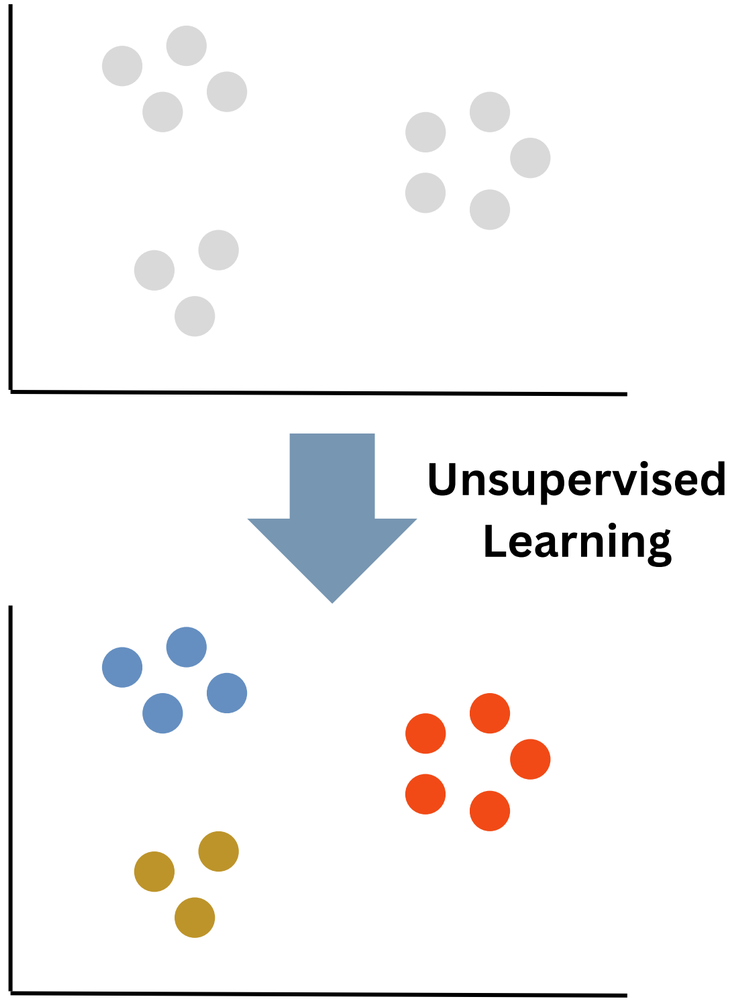
Unsupervised Learning

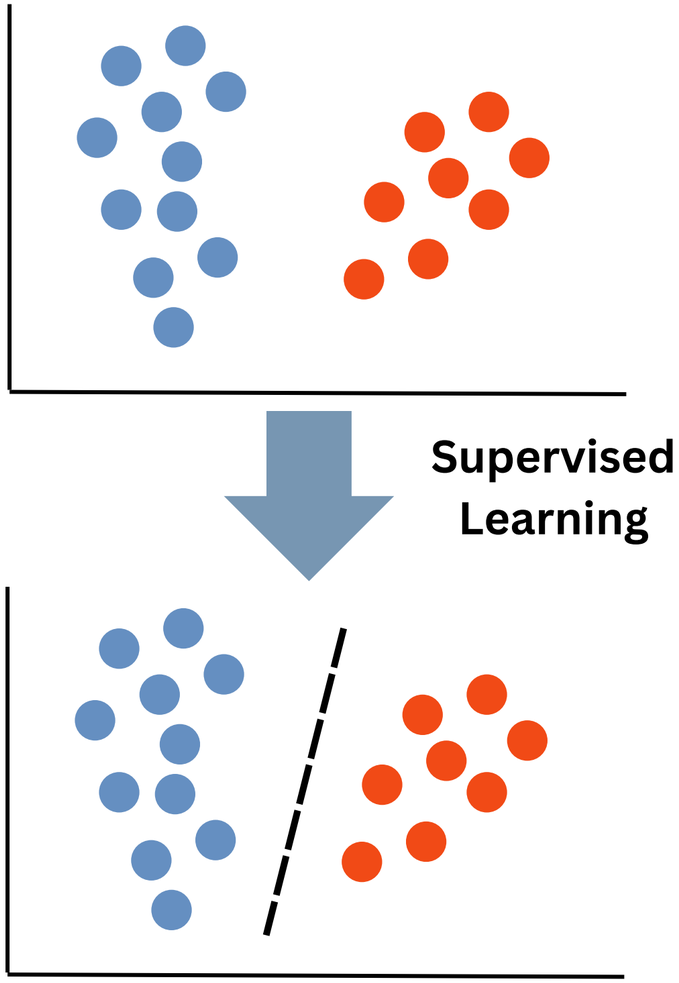
Unsupervised learning algorithm find structure in data without explicit guidance

- No "correct answers" or labels are provided in training
- The algorithm must discover meaningful patterns
- Unlabeled data is typically more abundant

Several distinct tasks fall under the unsupervised learning umbrella.

- Clustering algorithms
- Dimensionality reduction techniques
- Anomaly detection methods





Supervised Learning

Supervised learning relies on examples where the correct answer is already known.

- The fundamental idea is to learn from input-output pairs
- These paired examples serve as a teacher, showing the model what output should result from each input
- Creating these teaching examples typically requires human effort

The essential goal is to generalize beyond the training examples.

The machine learning pipeline

The machine learning workflow follows a structured sequence of steps from raw data to a working model.

- **Data Collection:** Gather examples relevant to your problem ○ Example: house listings with prices, tweets with sentiment labels
- **Feature Engineering:** Select and transform inputs into a usable representation
- **Preprocessing:** Scale numerical features, encode categorical ones, handle missing values
- **Model Selection:** Choose a model family suited to the task (regression, classification)
- **Training:** Fit the model parameters on training data
- **Evaluation:** Measure performance on held-out data to verify the model generalizes

Features and Targets

In supervised learning, every example in the dataset consists of features (inputs) and a target (output).

- Features (**X**): The properties used to make predictions
  - Predicting house price → size (m²), number of rooms, location, year built
  - Detecting spam → word frequencies, email length, number of links
- Target (**y**): The value the model is asked to predict
  - house price in € (a number)
  - spam or not spam (a category)
- Choosing informative features is often more impactful than choosing a better model

X

y

Feature Engineering

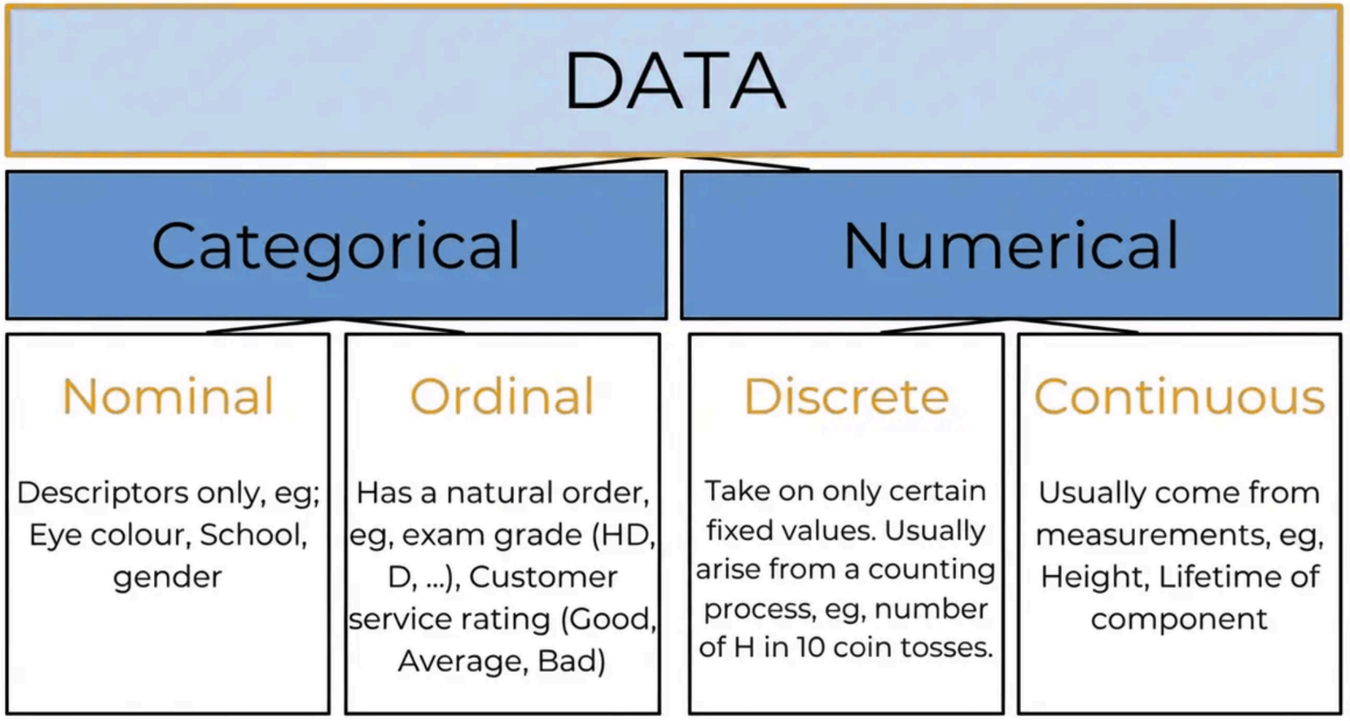
Raw data rarely arrives in a form that is directly useful and feature engineering is needed to transform observations into informative inputs.

- Creating new features: Combine or transform existing variables
  - "Age of house" = current year − year built
  - "Debt-to-income ratio" = monthly debt / monthly income
- Selecting relevant features: Remove irrelevant or redundant variables
  - Including noise features hurts model performance
- In classical ML, feature engineering is a major effort and often determines success
- In deep learning, raw inputs are often fed directly and the network learns its own representations

Numerical and Categorical Features

Features come in different types and machine learning requires them to be handled differently.

- Numerical features: Take continuous or discrete numeric values
- Categorical features: Represent discrete labels with no natural numeric meaning



Features Scaling

Numerical features often live on very different scales and this can cause serious problems for many ML algorithms.

- **The problem:** Features with large ranges dominate distance computations and training
  - Example: "income" (0–100,000) vs. "age" (0–100)
  - income drowns out age entirely
- **Standardization (Z-score normalization):** Rescale to mean = 0, std = 1
  - $x' = (x - \mu) / \sigma$
  - Best when the feature is roughly normally distributed
- **Min-Max normalization:** Rescale to range [0, 1]
  - $x' = (x - x\_min) / (x\_max - x\_min)$

Always scale features before using distance-based models (k-NN, SVM) or neural networks

One Hot Encoding

Categorical features cannot be fed directly into most ML models; they must be first converted to numbers

- **Naive approach**: Assign an integer to each category
  - "cat" → 0, "dog" → 1, "bird" → 2 ○
  - Problem: implies "bird" > "dog" > "cat"
- **One-Hot Encoding:** Create one binary column per category
  - Each row has exactly one 1
  - For high-cardinality features (many categories), other encodings may be needed

id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

The Dataset as a Matrix

We represent the full dataset as a feature matrix and a target vector.

- The dataset of N examples, each with D features, is stored as a feature matrix or input **X**
  - **X** is an N × D matrix: N rows (one per example), D columns (one per feature)
- The targets are stored as a target vector **y**
  - **y** is an N-dimensional vector: one scalar value per example

INPUT X	Feature 1	Feature 2
Sample 1	x_1^1	x_1^2
Sample 2	x_2^1	x_2^2
Sample 3	x_3^1	x_3^2

TARGET y
y_1
y_2
y_3

Regression and Classification

Supervised learning problems generally fall into two categories

Regression tasks

- The output is a continuous numerical value representing a quantity or magnitude.
- Common applications include predicting house prices based on features, forecasting temperatures, or estimating a person's age from a photograph.

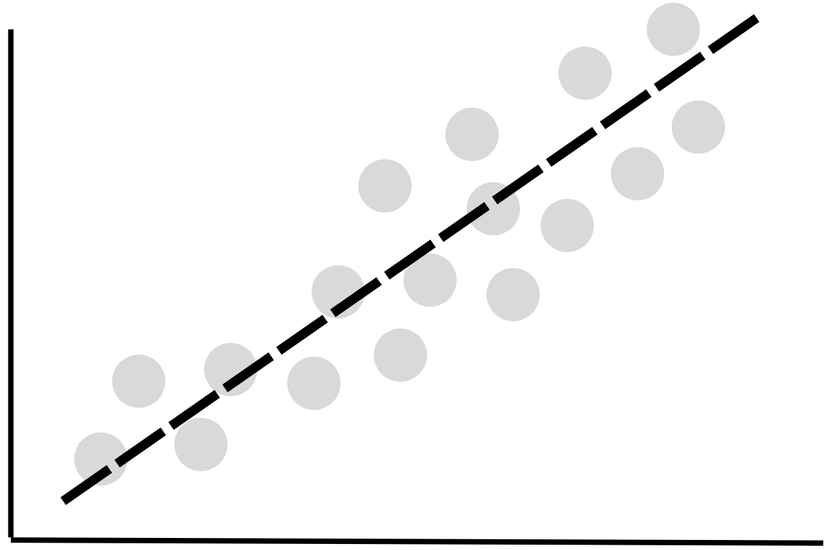
Classification tasks

- The goal is to assign inputs to distinct categories. The output is either a discrete class label or a probability distribution across possible classes.
- Examples include filtering spam emails, diagnosing diseases from symptoms, or recognizing handwritten digits.

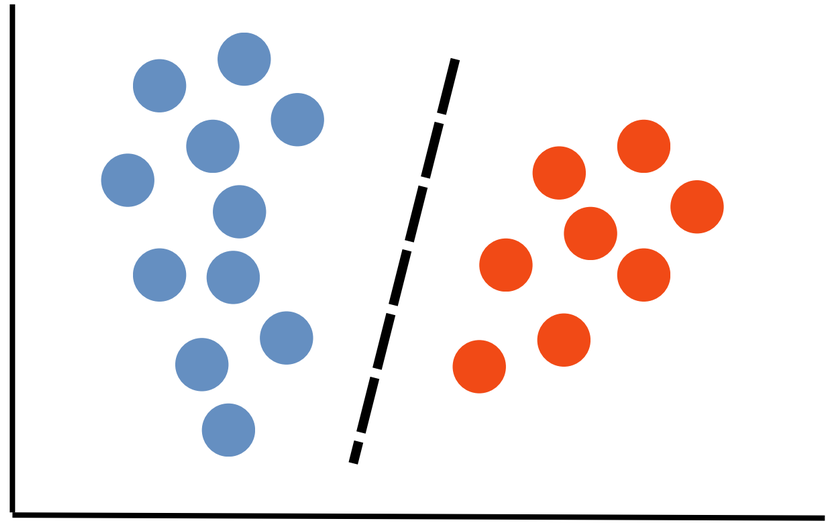
Regression and Classification

Supervised learning problems generally fall into two categories

Regression



Classification



Model and Parameters

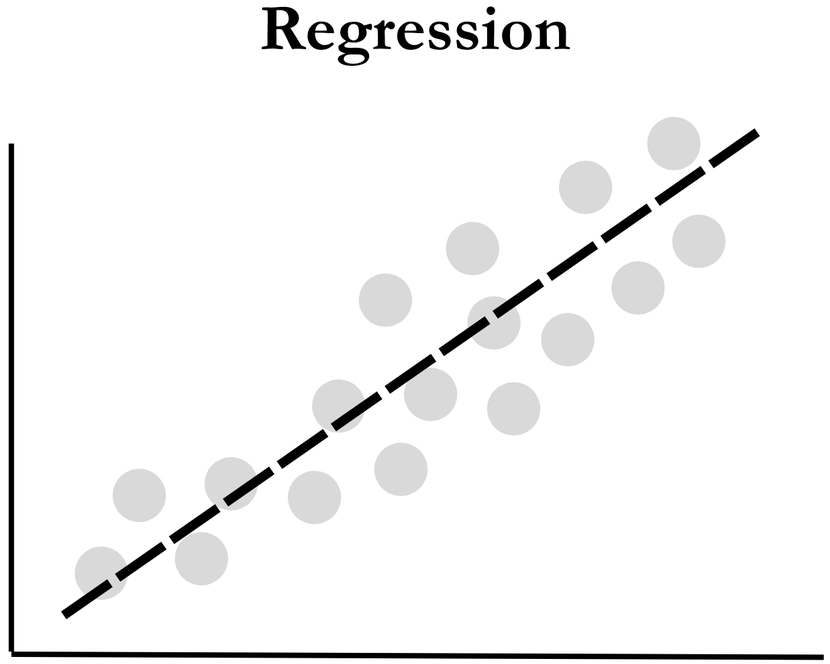
A model is a mathematical function with some parameters that maps inputs to predictions

- A **model** defines the functional form of the mapping: $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$
  - It specifies the "shape" of the input-output relationship
  - Example: a linear model assumes $\hat{\mathbf{y}} = \mathbf{w} \cdot \mathbf{x} + \mathbf{b}$
- **Parameters ($\boldsymbol{\theta}$):** The numbers inside the model that are learned from data
  - Parameters are what the model "learns"
  - they encode the knowledge extracted from data
- Same model architecture, different parameter values → completely different behavior

Example: Linear Regression

Linear regression is the most simple example of supervised learning

- The model is simply: $y = wx + b$
  - y is the output (prediction)
  - x is the input feature
  - w is the weight, and b is the bias.
- We provide training examples as pairs of (x, y) values, such as (house size, house price).
- The model learns to predict y from x by adjusting its parameters (w and b).
- After training, we can predict y values for new x inputs the model hasn't seen before.



Machine Learning Models

A rich zoo of model families exists, each making different assumptions about the data.

Regression:

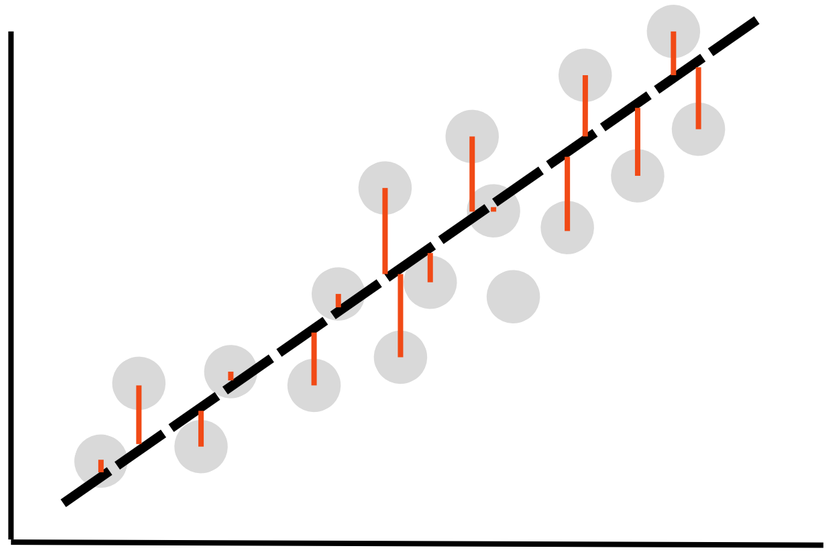
- Linear Regression: Fits a weighted linear combination of features
- Polynomial Regression: Adds polynomial feature interactions to capture nonlinearity
- Random Forest: Averages many decision trees and reduces overfitting
- Support Vector Regression (SVR): Finds a band minimizing prediction error

Classification:

- Logistic Regression: Models class probabilities with a sigmoid function
- Random Forest Classifier: Ensemble of trees; robust, widely used baseline
- Support Vector Machine (SVM): Finds the maximum-margin decision hyperplane
- k-Nearest Neighbors (kNN): Classifies by majority vote among the k closest training points

Loss Function

Mean Absolute Error



The Loss Function provides a quantitative measure of how well model predictions align with actual targets.

- They are a clear optimization target.
- We aim to find the combination of model parameters that minimizes the chosen loss function.
- Different problems call for different types of loss functions.
  - Mean Squared Error (MSE) or Mean Absolute Error (MAE) are typical choices for regression
  - Cross Entropy Loss is the typical choice for classification

Training

Training is the process by which a model adjusts its parameters to minimize the loss and it is the core of machine learning.

- Every model has a learning algorithm that specifies how parameters are updated
  - Most models use gradient descent
  - It iteratively move parameters in the direction that reduces the loss
- The training loop:
  a. Make a prediction: $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{w})$
  b. Compute the loss: $L(\mathbf{w}) = \text{loss}(\hat{\mathbf{y}}, \mathbf{y})$
  c. Update parameters to reduce the loss: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L$
  d. Repeat until the loss converges
- η is the learning rate

Parameters and Hyperparameters

Not all numbers in a machine learning system are learned from data, some must be set by the user before training begins.

- **Parameters:** Values learned automatically during training
  - Adjusted by the learning algorithm to minimize the loss
  - Examples: weights and biases of a neural network, coefficients of linear regression
- **Hyperparameters:** Set by the user before training; not touched by the learning algorithm
  - They control the structure of the model or how training proceeds
  - Examples: polynomial degree, number of trees, learning rate, number of layers

Notation

I will try to be consistent with the notation in the slides

- x will always be the input
- y the output
- bold letters denote vectors and matrices
  - for instance **x** is an input with multiple dimensions
- bold letters also denote matrices
  - for instance **W** can be the weight matrix of a Neural Network
  - **Wx** is the product between the matrix **W** and the vector **x**
- the parameters of the models are denoted by Greek letters or with a **W**
- the loss function is denoted with an L
  - it is a function of the model parameters L[**W**]

Universität
Konstanz

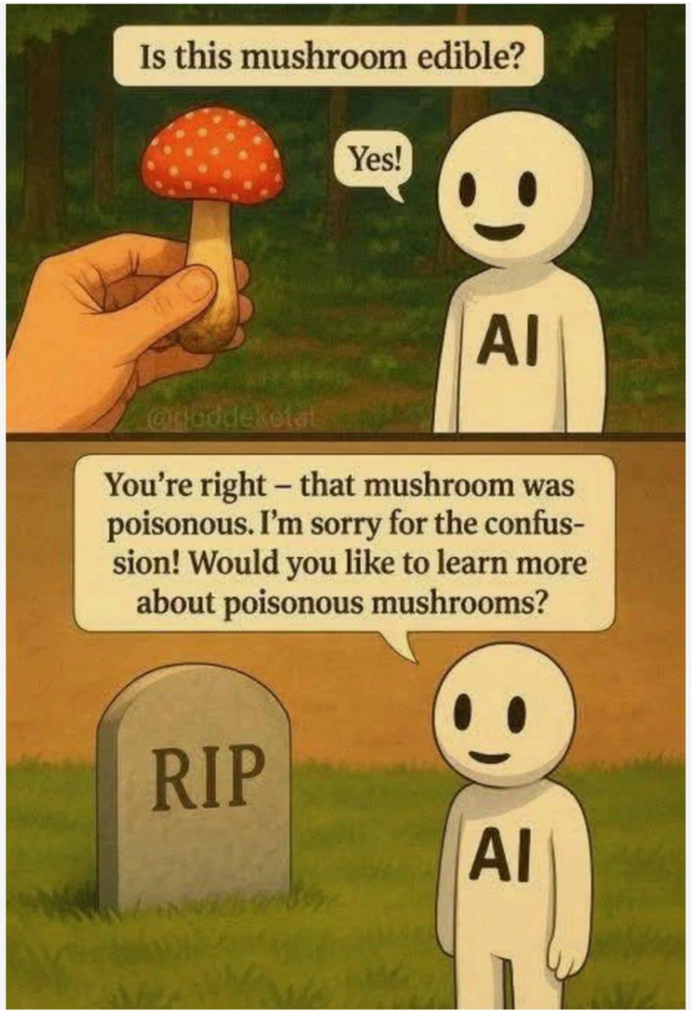
Evaluating Models

Model Evaluation

Training performance alone is never sufficient! We must always assess how a model performs on data it has never seen.

- A model that memorizes training data can be completely useless in the real world
- Example: a fraud detection model scoring 100% on training data but missing all real fraud cases in deployment
- Medical diagnostics, credit scoring, policy decisions, the stakes are real

Does the model learn the true pattern, or just the training examples?



Train, Validation and Test Set

To evaluate a model honestly, we partition the dataset into three subsets, each with a distinct role.

- **Training set (~70%):** Used to fit model parameters
  - The model sees this data during training
- **Validation set (~15%):** Used to tune hyperparameters and compare model variants
  - Never used for training
  - Simulates unseen data during development
- **Test set (~15%):** Used once at the very end to report final performance
  - The golden rule: touch it exactly once or twice maximum
  - If you look at test results and then retrain, **the test set is contaminated**

Generalization

The ultimate goal of machine learning is not to fit training data, but to perform well on new, unseen examples.

- **Generalization:** The ability of a model to make accurate predictions on data it was not trained on
- A model that **memorizes** training examples does **not generalize**
- Factors that improve generalization:
  - More and more diverse training data
  - Simpler models (fewer parameters)
  - Proper preprocessing and feature selection

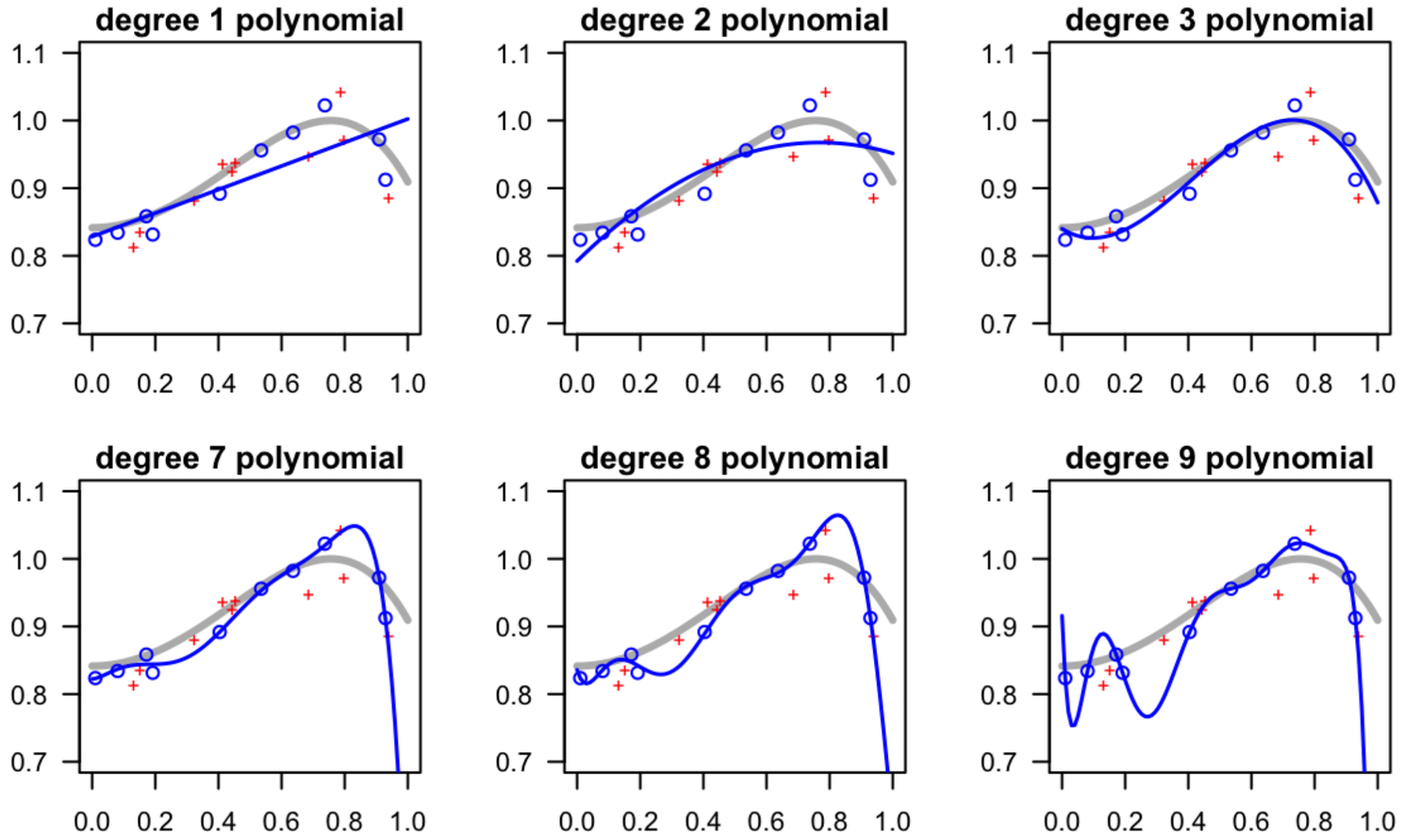
Overfitting and Underfitting

Two failure modes arise when model complexity is poorly matched to the data.

- **Underfitting:** The model is too simple to capture the true pattern
  - High training error and high test error
  - The model fails even on the data it trained on
  - Cause: model has too few parameters
- **Overfitting:** The model is too complex and memorizes noise in the training data
  - Low training error but high test error
  - The model has learned patterns that do not exist in new data
  - Cause: too many parameters relative to the amount of training data

A model must be just complex enough to capture the true signal, no more

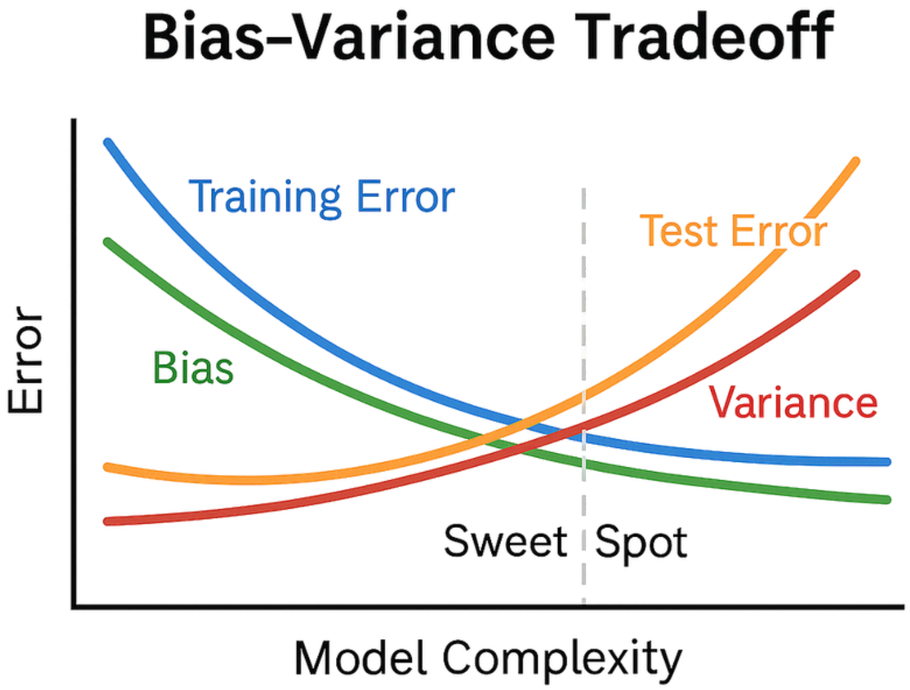
Example: Polynomial Regression



Bias-Variance Tradeoff

The tension between underfitting and overfitting has a precise mathematical formulation.

- The expected test error decomposes into three terms:
  - Bias: Error from wrong assumptions pattern
  - Variance: Error from sensitivity to training data
  - Irreducible noise: randomness in the data
- High bias, low variance → Underfitting
- Low bias, high variance → Overfitting



Metrics for Regression

After training a regression model, numerical metrics quantify the quality of predictions.

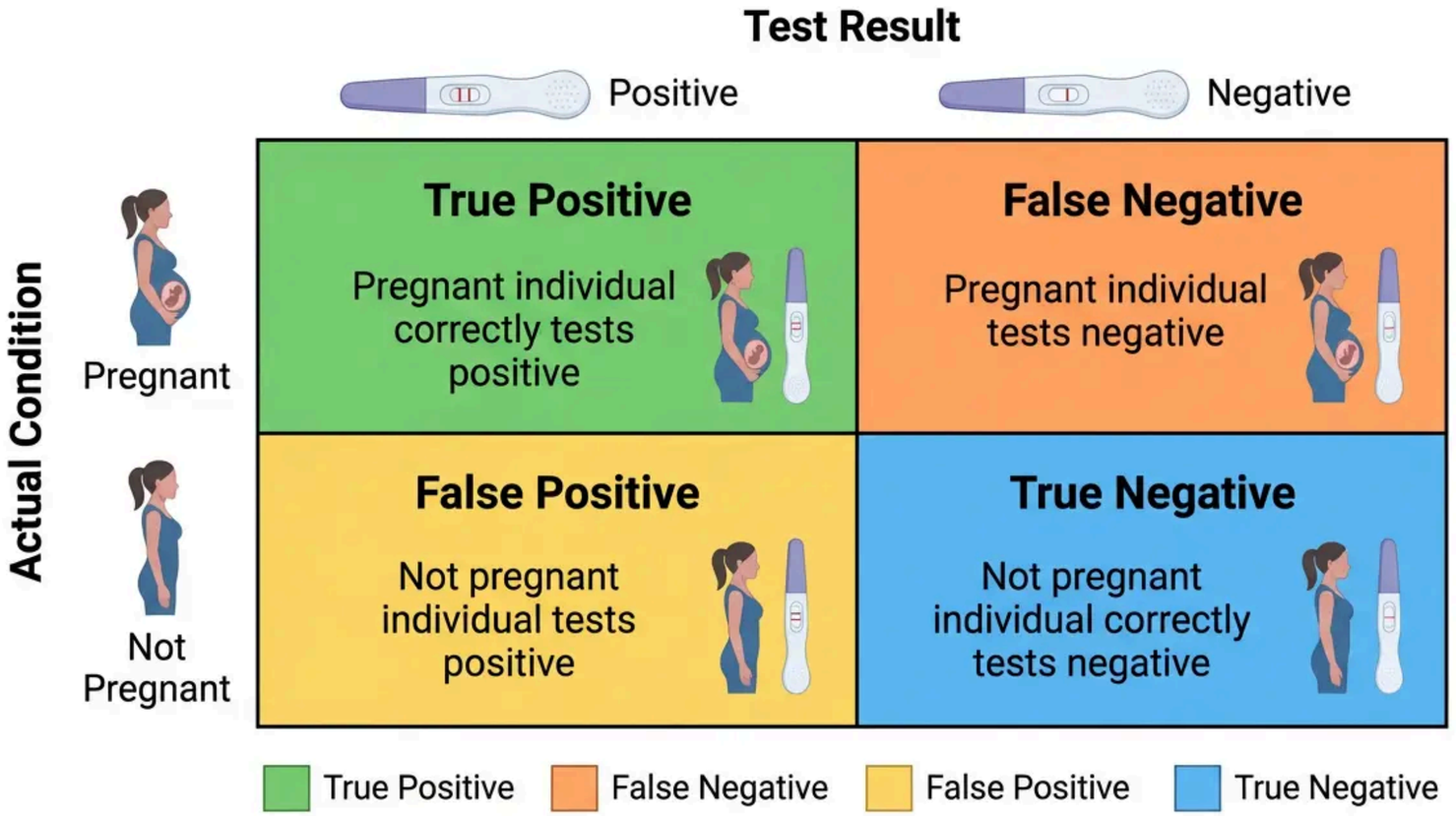
- Mean Absolute Error (MAE): Average absolute difference between prediction and target
  - $MAE = (1/N) \Sigma |\hat{y}_i - y_i|$
  - Intuitive: "on average, predictions are off by X units"
- Mean Squared Error (MSE): Average squared difference
  - $MSE = (1/N) \Sigma (\hat{y}_i - y_i)^2$
  - Penalizes large errors more heavily than MAE
- Root Mean Squared Error (RMSE):
  - $\sqrt{MSE}$, same units as the target, more interpretable
- R^2 (Coefficient of Determination)

Classification and Confusion Matrix

Before computing classification metrics, we need to understand the confusion matrix

- For binary classification (positive / negative), predictions fall into four cases:
  - True Positive (TP): Predicted positive, actually positive ✓
  - True Negative (TN): Predicted negative, actually negative ✓
  - False Positive (FP): Predicted positive, actually negative ✗
  - False Negative (FN): Predicted negative, actually positive ✗
- These four counts form a 2×2 confusion matrix
- Example: Spam filter:
  - FP: a legitimate email incorrectly flagged as spam (frustrating)
  - FN: a spam email that slips through (also bad)

Example: Confusion Matrix



Metrics for Classification

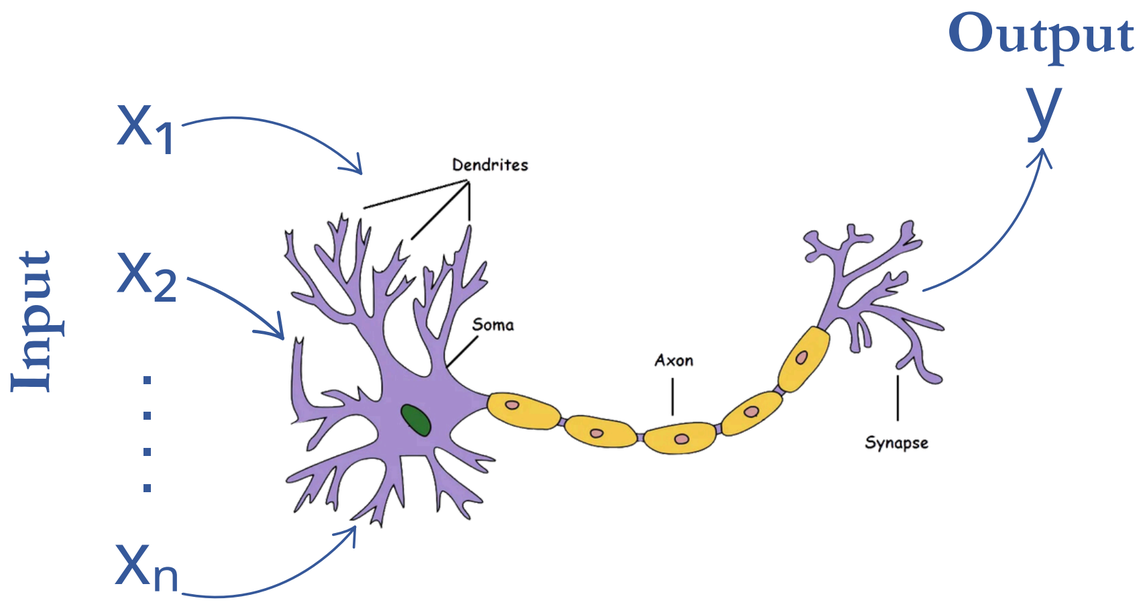
All standard classification metrics are computed directly from the confusion matrix.

- Accuracy: Fraction of all correct predictions
  - (TP + TN) / (TP + TN + FP + FN)
  - Misleading when classes are imbalanced (e.g., 99% negative)
- Precision: Of all predicted positives, how many are truly positive?
  - TP / (TP + FP) "when the model says positive, how often is it right?"
  - High precision → few false alarms
- Recall (Sensitivity): Of all actual positives, how many did the model catch?
  - TP / (TP + FN) "of all real positives, how many were found?"
  - High recall → few missed positives
- F1 Score: Harmonic mean of precision and recall
  - F1 = 2 × (Precision × Recall) / (Precision + Recall) → good with unbalanced classes

Universität
Konstanz

The Perceptron

Biological Neurons



Neural networks draw inspiration from the brain's biological structure.

- **Input:** Neurons receive signals from other neurons through dendrites.
- **Processing:** Incoming signals are combined and processed in the cell body.
- **Activation Function:** If the processed signal is strong enough, it triggers firing.
- **Output:** When activated, the neuron sends a signal through its axon.

The First Neural Network

The Perceptron was the first artificial neural network model.

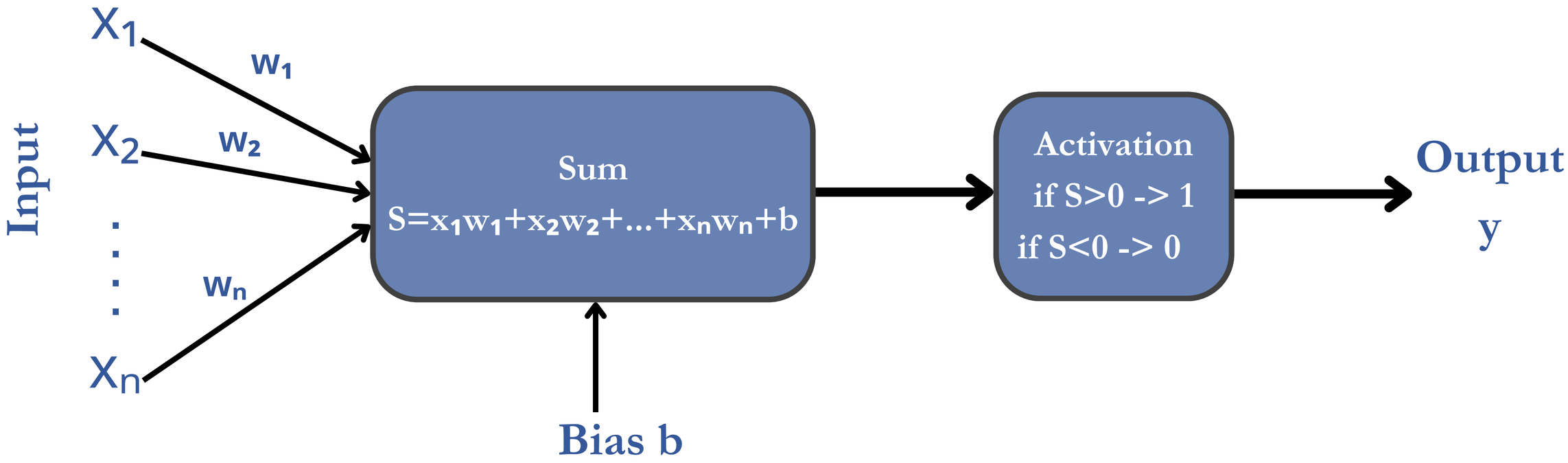
- Created in the 1950s by Frank Rosenblatt, a pioneer in artificial intelligence
- Simulates how neurons in the human brain process information

The Perceptron follows a simple operational principle:

- It receives multiple inputs, each with an associated weight
- These inputs are combined and produce an output if they exceed a threshold

Structure of the Perceptron

The Perceptron combines weighted inputs and activation:



Mathematical Notation

Mathematically we can represent the perceptron using a vector product

$y=a[\mathbf{wx}+b]$

Here

- $\mathbf{w}=(w_1, w_2, ..., w_n)$ is the vector containing the n weights
- $\mathbf{x}=(x_1, x_2, ..., x_n)$ is the vector containing the n-dimensional input
- a is the activation function. In our case
  - $a(x)=1$ if $x>0$
  - $a(x)=0$ if $x<0$
- b is the bias

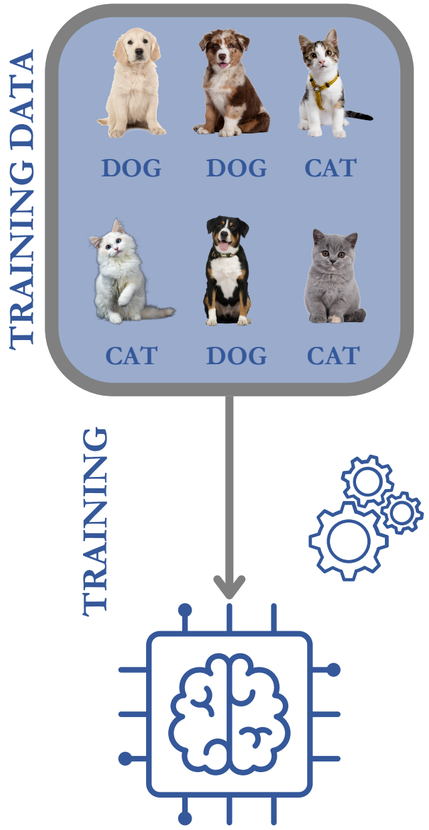
With **wx** we denote the scalar product of the two vectors

$\mathbf{wx}=x_1w_1+x_2w_2+...+x_nw_n$

Supervised Classification

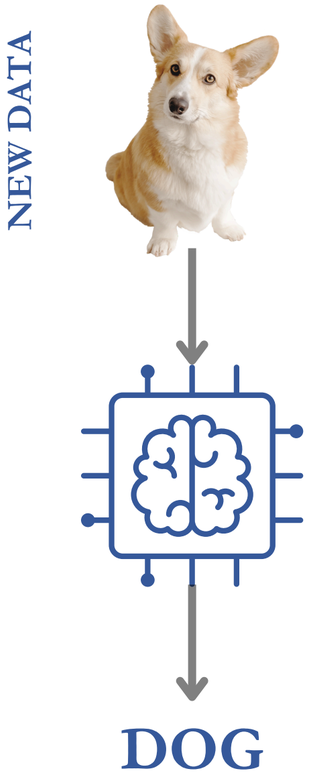
The computer learns from classified examples to predict categories for new data.

LEARNING



The model analyzes pre-classified examples and learns how to distinguish between categories

CLASSIFICATION



The model applies what it learned to categorizes new, previously unseen data

Classifying Dogs and Cats

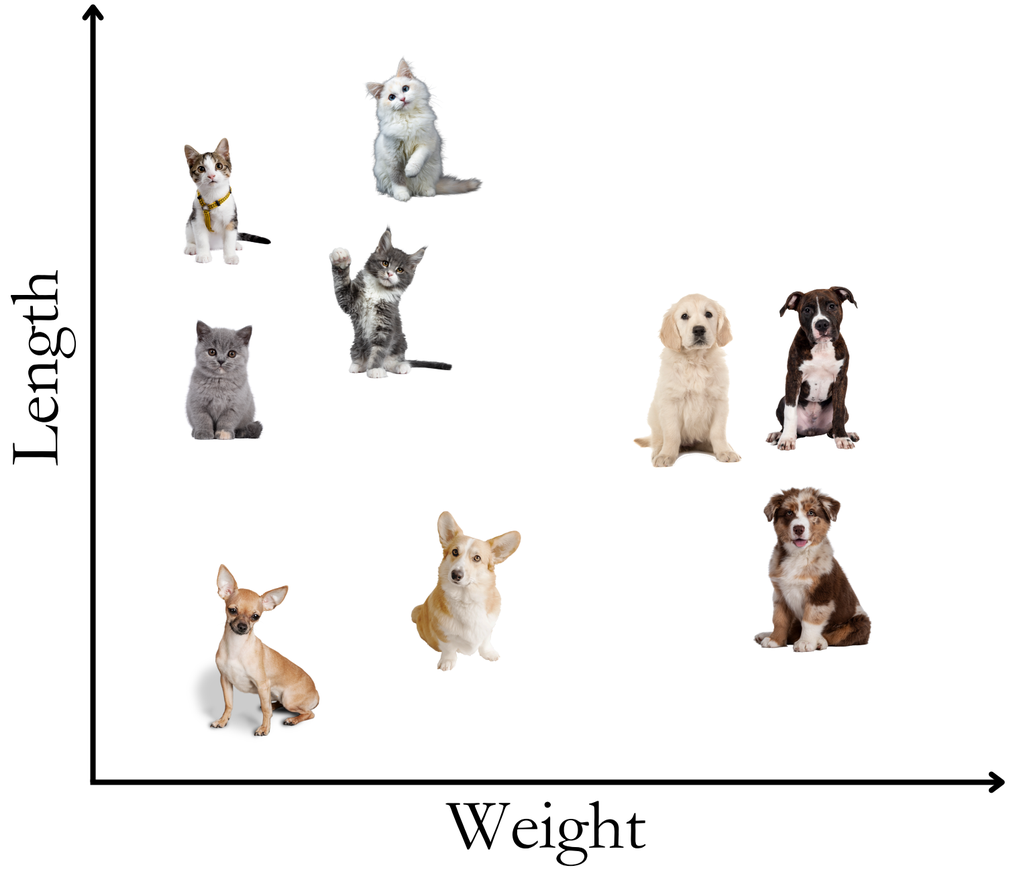
Let's consider a simplified version of the classification problem

- Dogs and cats can be characterized by various features
- We focus on just two: weight and length

When can plot different animals on a graph using weight and length

- Each animal is represented by a point

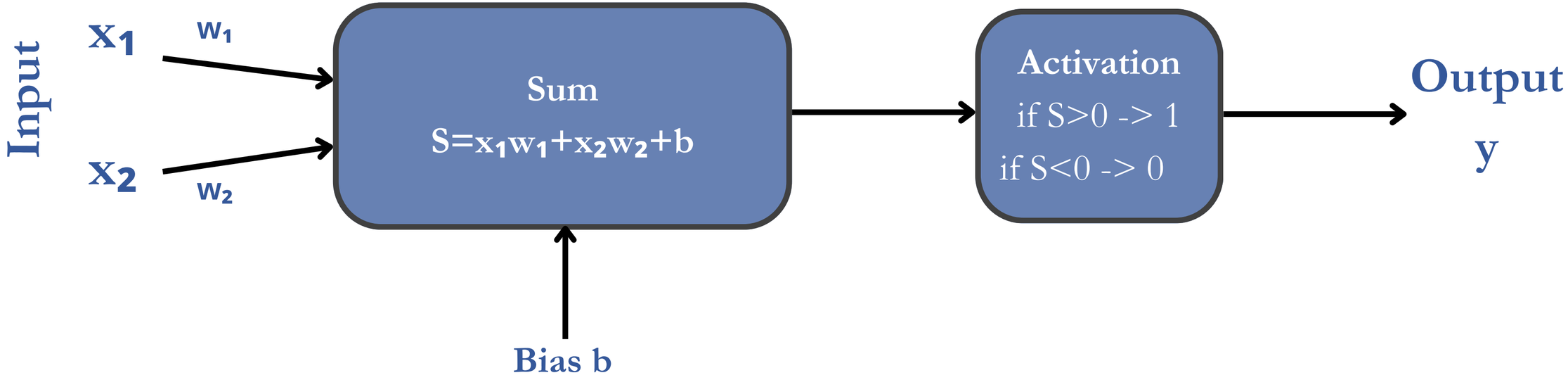
The Perceptron's job is to determine which group (dog or cat) a new animal belongs to



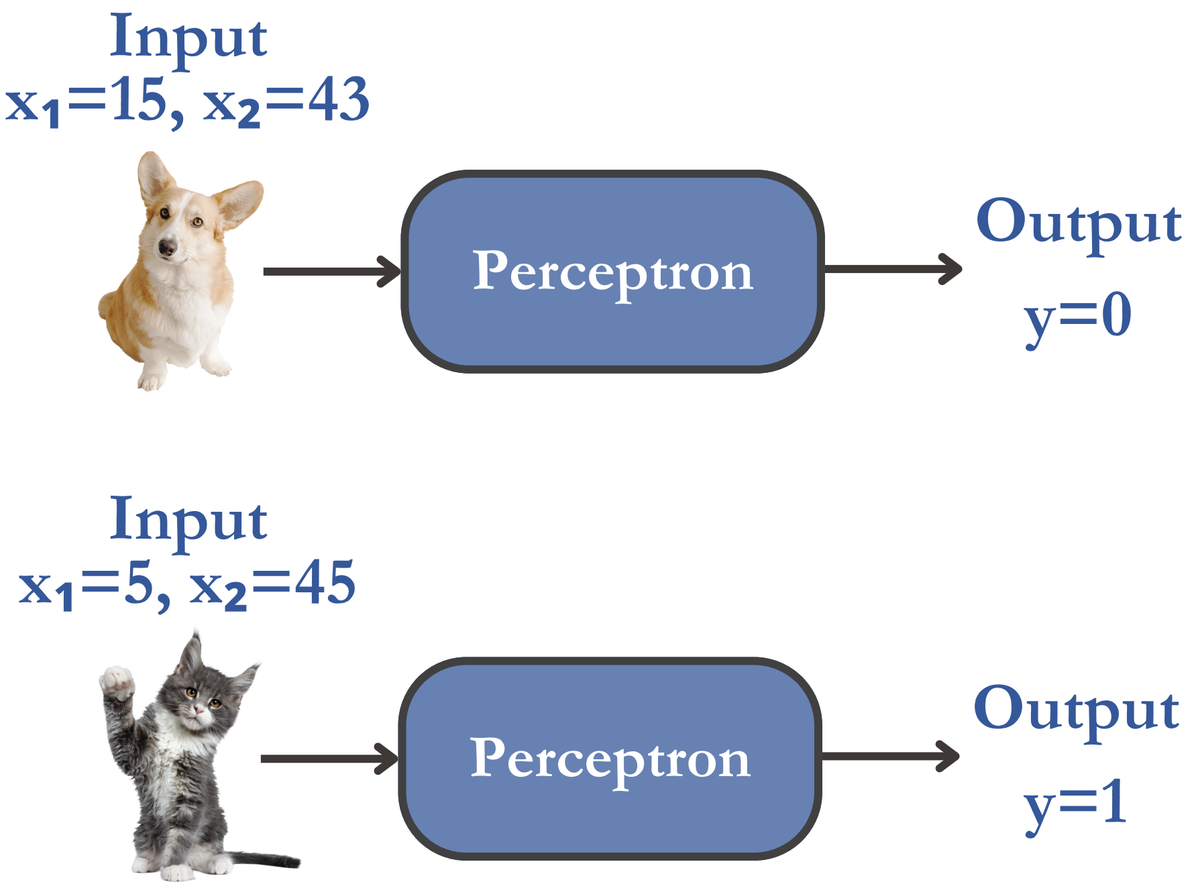
Perceptron with 2 Inputs

In this simple case we have just 2 inputs $\mathbf{x}=(x_1, x_2)$ and 3 parameters

- x_1 is the weight of the animal, x_2 is length
- w_1 and w_2 are the weights
- the bias is denoted by b



Automatic Classification



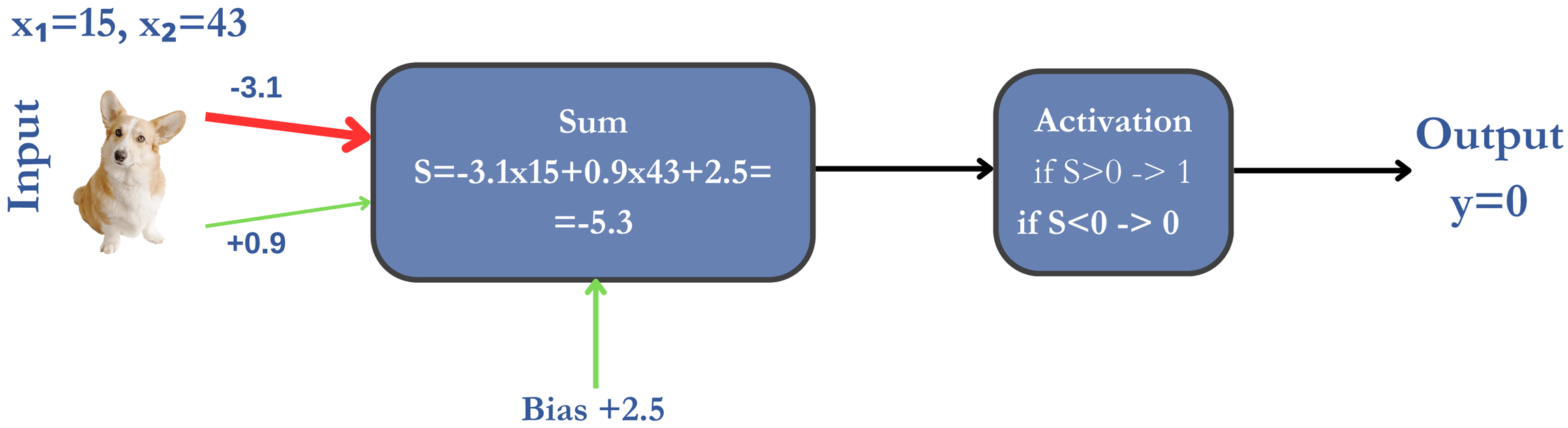
The Perceptron distinguishes between two categories:

- **Input:** Animal characteristics (e.g., weight and length)
- **Processing:** The Perceptron applies weights and calculates the output
- **Output:** Classification result
  - 0 = dog
  - 1 = cat

Perceptron with 2 Inputs

Let's see a practical example plugging some numbers in the perceptron

- we use as parameters w_1=-3.1, w_2=0.9 and b=2.5
- the input is **x**=(15, 43)



Summary

The Machine Learning Paradigm

Supervised learning maps features **X** to targets **y** through a model with learnable parameters The pipeline goes from raw data → preprocessing → training → and the loss function drives parameter updates thanks to a training algorithm

Evaluating Models

A model must always be evaluated on held-out data since training performance alone is meaningless. Overfitting and underfitting arise when model complexity is mismatched to the data. Metrics like RMSE, precision, recall and F1 quantify performance depending on the task

The Perceptron

The Perceptron is the simplest neural network: a weighted sum of inputs passed through an activation function. As we will see, it is the building block of all deep learning architectures.