Limits of Learning, Mathematical Notations & Python Foundations

Chapter 2: Essential Concepts for ML

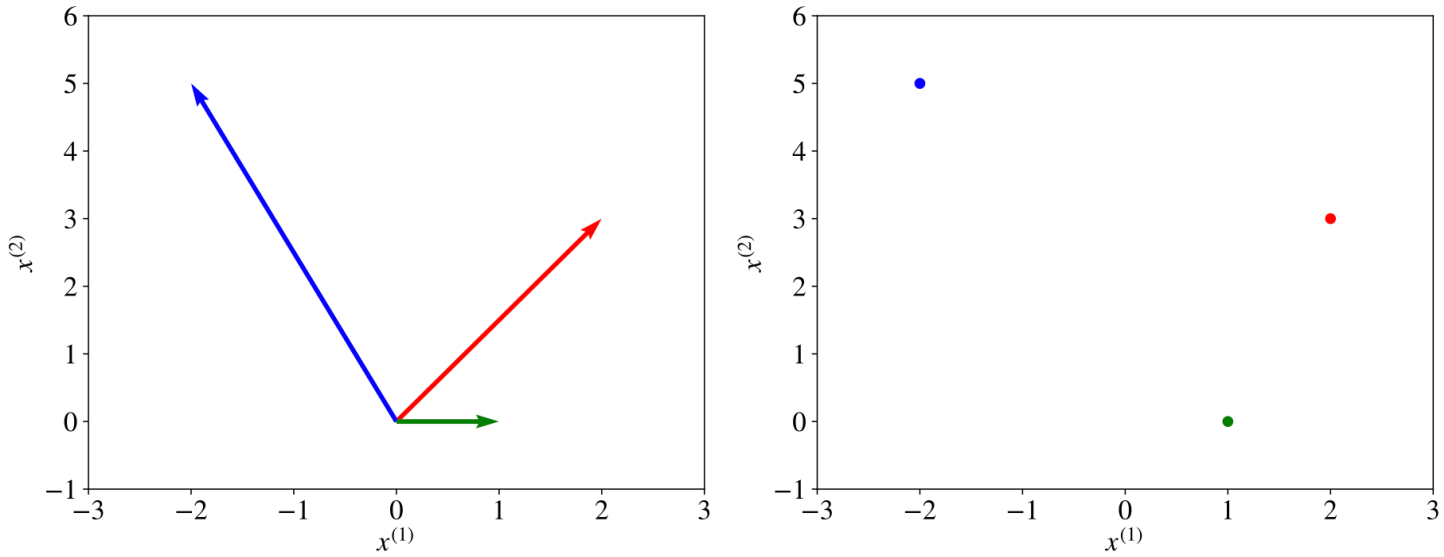
Scalars

- A scalar is a simple numerical value, like 15 or −3.25
- Variables or constants that take scalar values are denoted by an italic letter, like x or a

Vectors

- A vector is an ordered list of scalar values, called attributes
- Denoted as a **bold** character, for example, **x** or **w**
- Can be visualized as:
  - Arrows that point to some directions
  - Points in a multi-dimensional space

Vectors



Example vectors:

- **a** = [2, 3]
- **b** = [−2, 5]
- **c** = [1, 0]

Vector Attributes

- Attribute of a vector denoted as italic value with index: $w^{(j)}$ or $x^{(j)}$
- Index j denotes specific dimension of the vector
- Example: In vector **a** = [2, 3]:
  - $a^{(1)} = 2$
  - $a^{(2)} = 3$

Important: $x^{(j)} \neq x^2$ (power operator)

- For power of indexed attribute: $\left(x^{(j)}\right)^2$

Multiple Indices

Variables can have two or more indices:

- $x_i^{(j)}$
- $x_{i,j}^{(k)}$

Example: In neural networks, $x_{l,u}^{(j)}$ denotes:

- Input feature j of unit u in layer l

Sets

- An unordered collection of unique elements
- Denoted as calligraphic capital character, e.g., $\mathcal{S}$

Finite sets:

- $\{1, 3, 18, 23, 235\}$
- $\{x_1, x_2, x_3, x_4, \ldots, x_n\}$

Infinite sets:

- $[a, b]$ - includes endpoints a and b
- (a, b) - excludes endpoints a and b
- $\mathbb{R}$ - all real numbers from $-\infty$ to $+\infty$

Set Operations

Membership: $x \in S$ (x belongs to set S)

Intersection: $S_3 \equiv S_1 \cap S_2$

- Example: $\{1, 3, 5, 8\} \cap \{1, 8, 4\} = \{1, 8\}$

Union: $S_3 \equiv S_1 \cup S_2$

- Example: $\{1, 3, 5, 8\} \cup \{1, 8, 4\} = \{1, 3, 4, 5, 8\}$

Capital Sigma Notation

Summation over collection X = {x_1, x_2, ..., x_{n−1}, x_n}:

$$\sum_{i=1}^{n} x_i \overset{\text{def}}{=} x_1 + x_2 + \ldots + x_{n-1} + x_n$$

Summation over vector attributes x = [$x^{(1)}, x^{(2)}, \ldots, x^{(m)}$]:

$$\sum_{j=1}^{m} x^{(j)} \overset{\text{def}}{=} x^{(1)} + x^{(2)} + \ldots + x^{(m-1)} + x^{(m)}$$

Note: $\overset{\text{def}}{=}$ *means "is defined as"*

Capital Pi Notation

Product of elements in collection or vector attributes:

$$\prod_{i=1}^{n} x_i \overset{\text{def}}{=} x_1 \cdot x_2 \cdot \ldots \cdot x_{n-1} \cdot x_n$$

Where:

- a · b means a multiplied by b
- We often omit · for simplicity: ab = a multiplied by b

Operations on Sets

Derived set creation:

$S' \equiv \{x^2 \mid x \in S, x > 3\}$

Meaning: Create new set S′ containing x^2 for all x in S where $x > 3$

Cardinality operator:

$|S|$ returns the number of elements in set S

Operations on Vectors

Vector Addition:

$\mathbf{x} + \mathbf{z} = [x^{(1)} + z^{(1)}, x^{(2)} + z^{(2)}, \ldots, x^{(m)} + z^{(m)}]$

Vector Subtraction:

$\mathbf{x} - \mathbf{z} = [x^{(1)} - z^{(1)}, x^{(2)} - z^{(2)}, \ldots, x^{(m)} - z^{(m)}]$

Scalar Multiplication:

$\mathbf{x}c \overset{\text{def}}{=} [cx^{(1)}, cx^{(2)}, \ldots, cx^{(m)}]$

Dot Product

Dot product of two vectors (scalar result):

$$\mathbf{wx} \overset{\text{def}}{=} \sum_{i=1}^{m} w^{(i)} x^{(i)}$$

Alternative notation: $\boldsymbol{w} \cdot \boldsymbol{x}$

Requirements:

- Both vectors must have same dimensionality
- Otherwise dot-product is undefined

Matrix-Vector Multiplication

Matrix W and Vector x:

$$W = \begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \end{bmatrix} \mathbf{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{bmatrix}$$

Vector on right (column vector):

$$W\mathbf{x} = \begin{bmatrix} w^{(1,1)}x^{(1)} + w^{(1,2)}x^{(2)} + w^{(1,3)}x^{(3)} \\ w^{(2,1)}x^{(1)} + w^{(2,2)}x^{(2)} + w^{(2,3)}x^{(3)} \end{bmatrix} = \begin{bmatrix} \mathbf{w}^{(1)}\mathbf{x} \\ \mathbf{w}^{(2)}\mathbf{x} \end{bmatrix}$$

Requirement: Vector dimensions = matrix columns

Vector Transpose

Transpose operation:

$$\mathbf{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \rightarrow \mathbf{x}^T = \begin{bmatrix} x^{(1)}, x^{(2)} \end{bmatrix}$$

Vector on left (transposed):

$$\mathbf{x}^T W = \begin{bmatrix} w^{(1,1)}x^{(1)} + w^{(2,1)}x^{(2)}, w^{(1,2)}x^{(1)} + w^{(2,2)}x^{(2)}, w^{(1,3)}x^{(1)} + w^{(2,3)}x^{(2)} \end{bmatrix}$$

Requirement: Vector dimensions = matrix rows

Functions

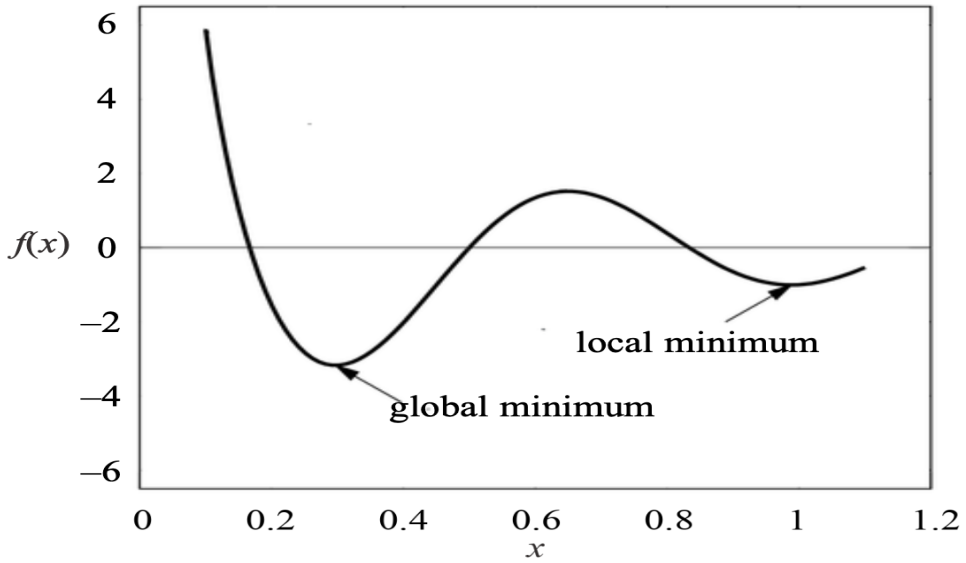
Definition:

- Relation associating each element x of set X (domain) to single element y of set Y (codomain)
- Notation: $y = f(x)$
- x = argument/input, y = value/output

Vector Function:

$\mathbf{y} = f(\mathbf{x})$ - function returning vector, can have vector or scalar argument

Functions



Local Minimum:

f(x) has local minimum at x = c if f(x) ≥ f(c) for every x in some open interval around x = c

Global Minimum:

Minimal value among all local minima

Max and ArgMax Operators

Max Operator

Given a set of values A = {a_1, a_2, ..., a_n}, the operator:

$$\max_{a \in A} f(a)$$

returns the highest value f(a) for all elements in the set A.

ArgMax Operator

$$\arg\max_{a \in A} f(a)$$

returns the element of the set A that maximizes f(a).

Simplified Notation

When the set is implicit or infinite, we can write:

- $\max_a f(a)$ or $\arg\max_a f(a)$

Min Operators

Operators **min** and **arg min** operate in a similar manner:

- $\min_{a \in A} f(a)$ returns the **lowest value**
- $\arg\min_{a \in A} f(a)$ returns the **element that minimizes** f(a)

Key Distinction

- **max/min**: Returns the **value**
- **argmax/argmin**: Returns the **argument** (element) that produces that value

Assignment Operator

Variable assignment:

$a \leftarrow f(x)$

Variable a gets new value: result of f(x)

Vector assignment:

$\mathbf{a} \leftarrow [a_1, a_2]$

Two-dimensional vector a gets value $[a_1, a_2]$

Derivatives and Gradients

What is a Derivative?

A derivative f' of function f describes **how fast f grows or decreases**

- **Constant derivative** (e.g., 5 or -3): function grows/decreases at constant rate
- **Function derivative** f'(x): growth rate varies across domain
- **f'(x) > 0**: function grows at point x
- **f'(x) < 0**: function decreases at point x
- **f'(x) = 0**: horizontal slope at point x

Basic Derivatives & Chain Rule

Known derivatives:

- $f(x) = x^2 \rightarrow f'(x) = 2x$
- $f(x) = 2x \rightarrow f'(x) = 2$
- $f(x) = 2 \rightarrow f'(x) = 0$ (any constant → 0)

Chain Rule: For $F(x) = f(g(x))$

$$F'(x) = f'(g(x)) \cdot g'(x)$$

Example: $F(x) = (5x + 1)^2$

- $g(x) = 5x + 1$, $f(g(x)) = (g(x))^2$
- $F'(x) = 2(5x + 1) \cdot 5 = 50x + 10$

Gradients: Multi-Variable Extension

Gradient = generalization of derivative for functions with **multiple inputs**

Partial Derivatives: Focus on one input, treat others as constants

Example: $f([x^{(1)}, x^{(2)}]) = ax^{(1)} + bx^{(2)} + c$

$$\frac{\partial f}{\partial x^{(1)}} = a + 0 + 0 = a$$

$$\frac{\partial f}{\partial x^{(2)}} = 0 + b + 0 = b$$

Gradient: $\nabla f = [\partial f/\partial x^{(1)}, \partial f/\partial x^{(2)}] = [a, b]$

Random Variables

Definition: A random variable, usually written as an italic capital letter like **X**, is a variable whose possible values are numerical outcomes of a random phenomenon.

Two Types:

1. **Discrete Random Variables**
2. **Continuous Random Variables**

Discrete Random Variables

Takes only a countable number of distinct values

- Examples: red, yellow, blue or 1, 2, 3, ...

Probability Mass Function (PMF):

- List of probabilities associated with each possible value
- Example: Pr(X = red) = 0.3, Pr(X = yellow) = 0.45, Pr(X = blue) = 0.25

Properties:

- Each probability ≥ 0
- Sum of all probabilities = 1
- Visualized as discrete bars (see figure 3a)

Continuous Random Variables

Takes an infinite number of possible values in some interval

- Examples: height, weight, time

Key Insight: Since values are infinite, $Pr(X = c) = 0$ for any specific value c

Probability Density Function (PDF):

- Describes probability distribution
- Codomain is non-negative
- Area under the curve = 1 (see figure 3b)

Note: Instead of exact probabilities, we use intervals: $Pr(a \leq X \leq b)$

Expectation for Discrete Variables

For discrete random variable X with k possible values {x_1, x_2, ..., x_k}:

$$E[X] \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i \Pr(X = x_i)$$

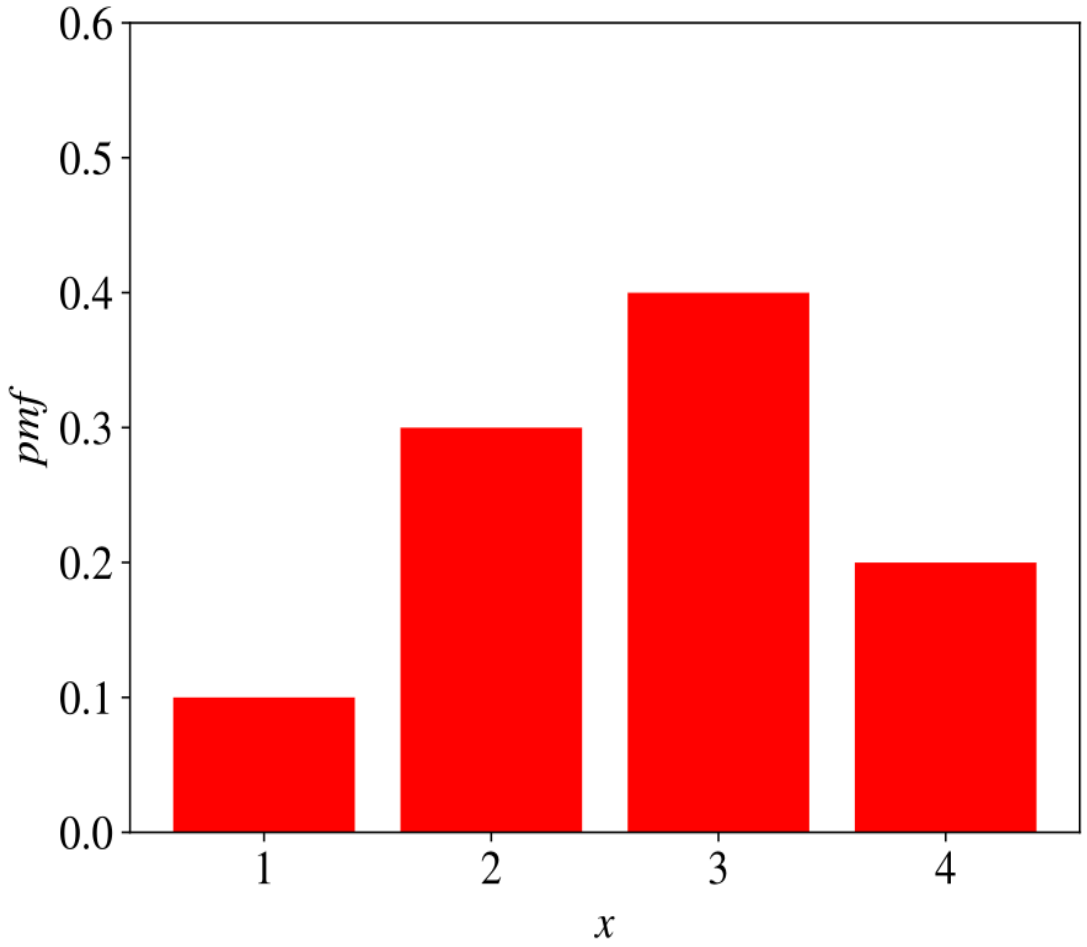
Expanded form:

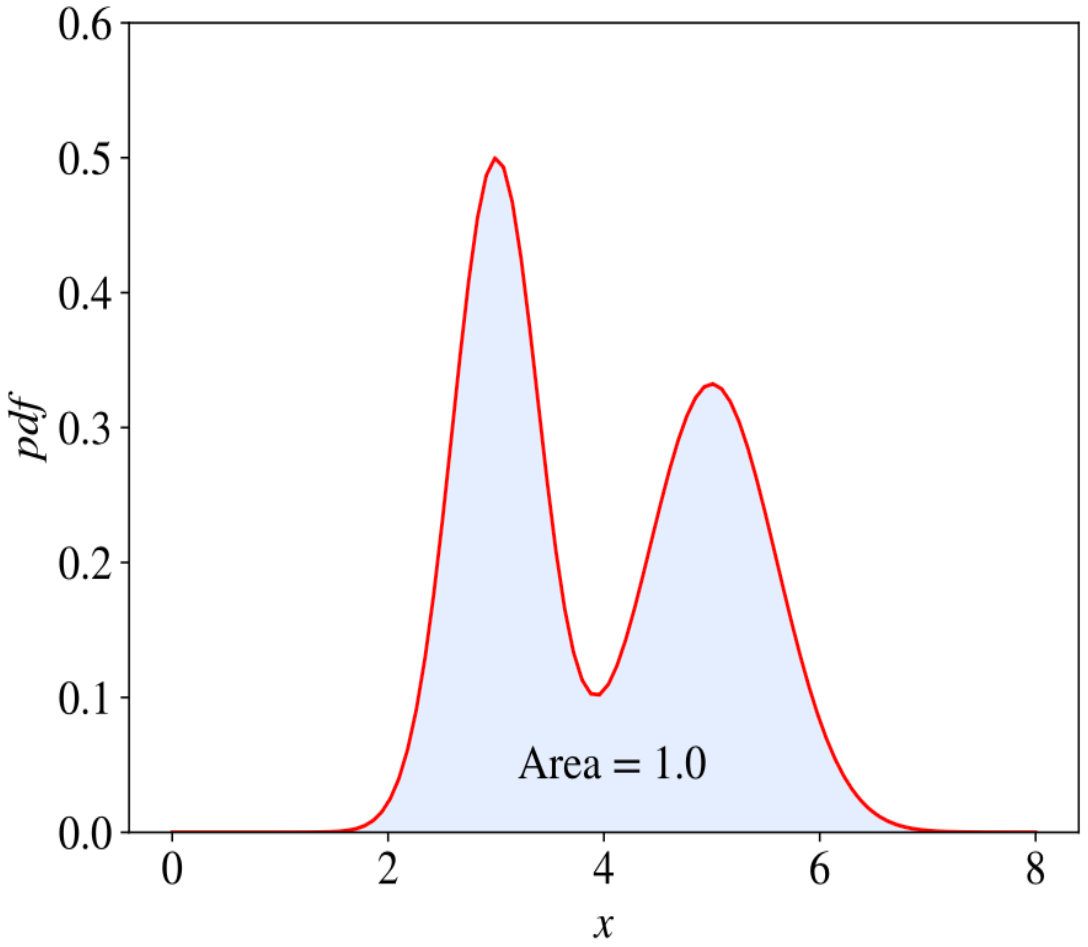
$$E[X] = x_1 \Pr(X = x_1) + x_2 \Pr(X = x_2) + \cdots + x_k \Pr(X = x_k)$$

Alternative names:

- Mean
- Average
- Expected value
- Often denoted as μ

Probability Mass Function (PMF) and **Probability Density Function (PDF)**





Standard Deviation for Discrete Variables

Standard deviation (usually denoted as σ):

$$\sigma \stackrel{\text{def}}{=} \sqrt{E[(X-\mu)^2]}$$

Expanded form:

$$\sigma = \sqrt{\Pr(X=x_1)(x_1-\mu)^2 + \Pr(X=x_2)(x_2-\mu)^2 + \cdots + \Pr(X=x_k)(x_k-\mu)^2}$$

where μ = E[X]

Expectation for Continuous Variables

For continuous random variable X:

$$E[X] \overset{\text{def}}{=} \int_{\mathbb{R}} x f_X(x) dx$$

where:

- f_X is the PDF of variable X
- $\int_{\mathbb{R}}$ is the integral over all real numbers

Key Concept: Integral is the continuous equivalent of summation

- Equals the area under the curve of function $x f_X(x)$
- PDF property: $\int_{\mathbb{R}} f_X(x) dx = 1$

From Theory to Practice

In Real Machine Learning:

- We don't know the true distribution f_X
- We can only observe some values of X
- These observed values are called **examples**
- Collection of examples = **sample** or **dataset**

The Challenge:

- Estimate properties of unknown distribution
- Make predictions based on limited samples
- Generalize from training data to new, unseen data

Bayes' Rule

Fundamental theorem for updating probabilities:

$$P(X = x|Y = y) = \frac{P(Y = y|X = x) \cdot P(X = x)}{P(Y = y)}$$

"The probability that X equals x given that Y equals y is equal to the probability that Y equals y given that X equals x, multiplied by the probability that X equals x, and divided by the probability that Y equals y."

Or

"The posterior probability of $X = x$ given $Y = y$ is the likelihood of $Y = y$ given $X = x$, times the prior probability of $X = x$, all normalized by the marginal probability of $Y = y$."

Parametric Models and Bayes' Rule

When we have a model with parameters θ:

Bayes' Rule becomes powerful for **parameter estimation** when we have:

- A model $f_\theta(x)$ with parameter vector θ
- Data samples to estimate θ

Example: Gaussian Distribution Model

$$f_\theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where $\theta \overset{\text{def}}{=} [\mu, \sigma]$

Parameter Update with Bayes' Rule

Updating parameters from data:

$$P(\theta = \hat{\theta}|X = x) \propto \frac{P(X = x|\theta = \hat{\theta}) \cdot P(\theta = \hat{\theta})}{P(X = x)}$$

Simplified (dropping normalization):

$$P(\theta = \hat{\theta}|X = x) \propto P(X = x|\theta = \hat{\theta}) \cdot P(\theta = \hat{\theta})$$

where $P(X = x|\theta = \hat{\theta}) \overset{\text{def}}{=} f_{\hat{\theta}}(x)$

Iterative Parameter Estimation

For sample S with finite possible θ values:

1. **Initialize prior:** Guess $P(\theta = \hat{\theta})$ such that $\sum_{\hat{\theta}} P(\theta = \hat{\theta}) = 1$
2. **For each example** $x_i \in S$**:**
    - Compute $P(\theta = \hat{\theta} | X = x_i)$ for all possible $\hat{\theta}$
    - Update prior: $P(\theta = \hat{\theta}) \leftarrow \frac{1}{N} \sum_{x \in S} P(\theta = \hat{\theta} | X = x)$
3. **Repeat** until convergence

Key Insight: Each new data point updates our belief about parameters

Maximum Likelihood Estimation

Best parameter values given data:

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{N} P(\theta = \hat{\theta} | X = x_i)$$

In practice, optimize log-likelihood:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \log P(\theta = \hat{\theta} | X = x_i)$$

Continuous Parameter Spaces

When θ has infinite possible values:

- Cannot enumerate all possibilities
- Need **numerical optimization** (e.g., gradient descent)
- Optimize log-likelihood directly:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log f_\theta(x_i)$$

Find: $\theta^* = \arg\max_\theta \mathcal{L}(\theta)$

Methods: Gradient descent, Newton's method, etc.

Machine Learning Fundamentals

The Core Assumption

Learning problems are characterized by some **unknown probability distribution** D over input/output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

Key Question: What if someone told you what D was?

Suppose you had access to a Python function `computeD(x, y)` that returns the probability of that (x, y) pair under D.

Result: Classification becomes simple!

The Bayes Optimal Classifier

If we knew D, we could define the **Bayes optimal classifier** as:

$$f^{(BO)}(\hat{x}) = \arg\max_{\hat{y} \in \mathcal{Y}} D(\hat{x}, \hat{y}) \quad (2.1)$$

In plain English:

- For any test input $\hat{x}$
- Simply return the $\hat{y}$ that maximizes `computeD` $(\hat{x}, \hat{y})$
- Choose the label with highest probability under the distribution

Why is it "Optimal"?

Theorem 1 (Bayes Optimal Classifier)

The Bayes Optimal Classifier $f^{(BO)}$ achieves minimal zero/one error of any deterministic classifier.

Important Notes:

- This theorem compares against deterministic classifiers
- A stronger result exists for randomized classifiers too (proof is messier)
- **Intuition:** For any given x, $f^{(BO)}$ chooses the label with highest probability, minimizing error probability

Proof Sketch

- Consider some other classifier g that claims to be better than $f^{(BO)}$
- Then there must be some x where $g(x) \neq f^{(BO)}(x)$
- Fix such an x for analysis
- Probability that $f^{(BO)}$ makes an error on this x: $1 - D(x, f^{(BO)}(x))$
- Probability that g makes an error on this x: $1 - D(x, g(x))$
- Since $D(x, f^{(BO)}(x)) > D(x, g(x))$, we have:

$$1 - D(x, f^{(BO)}(x)) < 1 - D(x, g(x))$$

Proof Sketch

- This means the probability that $f^{(BO)}$ errs on this particular x is **smaller** than the probability that g errs on it.

Generalization: This applies to **any** x for which $f^{(BO)}(x) \neq g(x)$

Therefore: $f^{(BO)}$ achieves smaller zero/one error than any g ∎

The Take-Home Message

If someone gave you access to the data distribution, forming an optimal classifier would be trivial.

The Reality:

- No one gives you this distribution
- We only have access to a **training set** sampled from D
- We need to learn the mapping from x to y without knowing D directly

The Challenge: How do we approximate $f^{(BO)}$ using only sample data?

Inductive Bias

The Necessity of Assumptions in Learning

The Learning Problem

Given: Limited examples from an infinite world

Challenge: How do we generalize to unseen cases?

Without Assumptions

- Any pattern could explain the data
- Infinite hypotheses are equally valid
- No basis for choosing one over another

The Learning Problem

With Inductive Bias

- Constrains what we consider "reasonable"
- Enables meaningful generalization
- Makes learning from finite data possible

Bottom line: Learning is impossible without some form of bias

Inductive Bias: What We Know Before the Data Arrives

The Spectrum of Bias

High Bias Example

Person strongly prefers blue color

- Chooses blue umbrella from poor company over black umbrella from excellent company
- Fast decision based on color preference
- Risk: Ignores quality due to color bias

The Spectrum of Bias

Low Bias Example

Person considers all factors equally

- Evaluates company reputation, price, durability, design, color
- Needs more information and time to decide
- Risk: Analysis paralysis, may miss good deals

The art: Matching your bias to what actually matters

Why This Matters

For Understanding:

- Every learning system embeds assumptions
- Performance depends on assumption quality
- There is no "assumption-free" learning

For Practice:

- Choose systems whose bias matches your domain
- Understand what your system assumes about the world
- Debug failures by examining bias mismatches

Remember: The question isn't whether to have bias, but what bias to have

Not Everything is Learnable

Why Machine Learning Sometimes Fails

Machine learning works well in many cases, but it's not magical.

There are fundamental reasons why learning algorithms can fail.

Four Sources of Learning Failure

1. Noise in Training Data

Feature-level noise: Sensor failures, measurement errors, typos

- Robot's laser range finder returns incorrect distance
- Typos in text reviews

Label-level noise: Incorrect classifications

- Student writes negative review but clicks positive rating
- Human annotation errors

Four Sources of Learning Failure

2. Insufficient Features

The available information is simply not enough to make accurate predictions

Medical diagnosis example:

- Have gene expressions, X-rays, family history
- Still impossible to definitively diagnose cancer
- Some problems are inherently uncertain

Contrived example:

- Trying to classify reviews as positive/negative
- Only have first 5 characters of each review
- Missing crucial information

Four Sources of Learning Failure

3. No Single Correct Answer

Different people legitimately disagree on the "right" answer

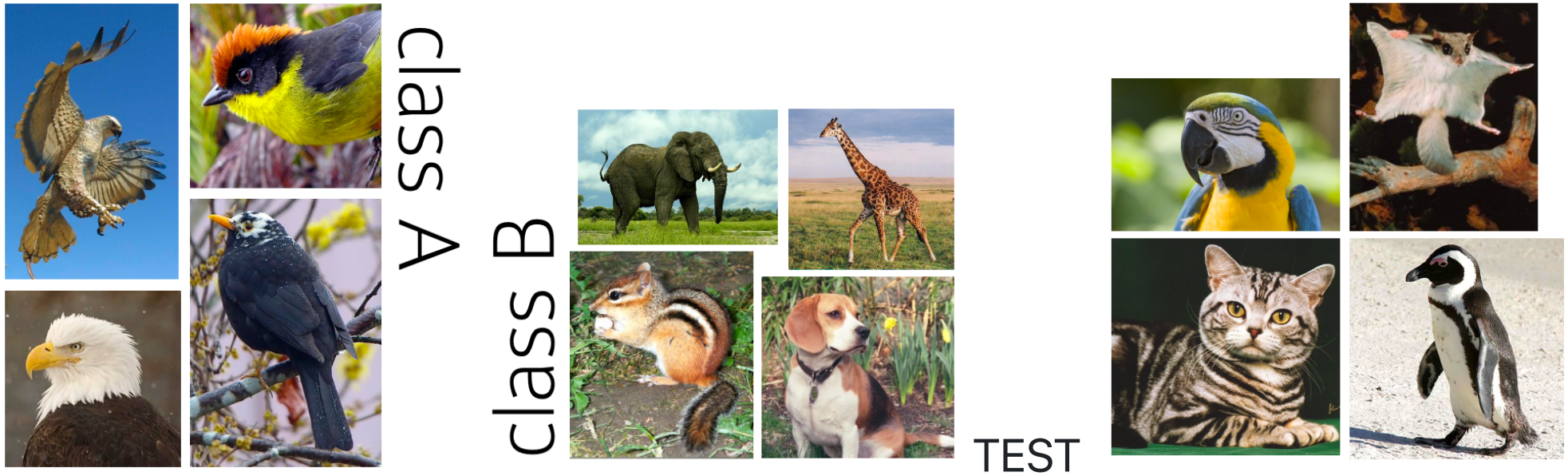
"Safe web search" example:

- Ask humans to label web pages as "offensive" or not
- What's offensive to one person is reasonable to another
- Subjective problems have inherent ambiguity

Often treated as label noise, but it's a distinct challenge

Four Sources of Learning Failure

4. Misaligned Inductive Bias



Four Sources of Learning Failure

4. Misaligned Inductive Bias

The learning algorithm's assumptions don't match the target concept

Bird/non-bird classification example:

- Humans might guess: bird vs. non-bird, or flying vs. non-flying
- Actual pattern: "background is in focus" vs. "background is blurry"
- This distinction is so unusual that most learners would miss it

Key insight: Neptunians who evolved to care about focus might learn this easily!

The Crucial Difference

Algorithm-Specific vs. Fundamental Problems

Inductive Bias Mismatch (Source #4):

- Problem with the *particular* learning algorithm
- Switching algorithms might solve the problem
- Algorithm's assumptions don't fit the data pattern

Other Sources (1-3):

- Fundamental properties of the learning problem
- No algorithm can overcome these limitations
- The problem itself is inherently difficult

Takeaway: Sometimes the algorithm is wrong for the data, sometimes the data is just hard

Thank you

Next

- Python, Numpy, Conda
- Fundamental Algorithms