Chapter 4: Probability Theory and Distributions for machine learning

Firas IBRAHIM

2025-2026

Contents

Many modern **machine learning techniques** are grounded in the principles of **probabilistic reasoning**. This chapter refreshes the key ideas of probability theory to establish a solid foundation for the statistical and machine learning methods explored later in the course.

We begin by defining the **sample space** and **events**, the basic building blocks of any probabilistic model. From there, we examine the concepts of **conditional probability** $P(A \mid B)$ and **independence**— tools that allow us to reason about uncertainty in a structured way.

A central focus of this chapter is **Bayes' Theorem**, a fundamental rule for **updating probabilities** in light of new evidence:

$$P(B \mid A) = \frac{P(A \mid B)\, P(B)}{P(A)}.$$

Next, we introduce **random variables** (discrete and continuous), and compute key quantities such as the **expected value** $E[X]$ and **variance** $\mathrm{Var}(X)$. These concepts quantify the behavior and variability of random outcomes.

Finally, the chapter concludes with **hands-on Python examples**, illustrating how to implement and experiment with these fundamental probability concepts programmatically.

0.1 Sample Spaces and Events

To understand the probabilistic foundations of machine learning and statistical reasoning, we first need a precise language for describing **uncertainty** and **possible outcomes**. This begins with the key concepts of **sample spaces** and **events**, which form the building blocks of probability theory.

0.1.1 The Sample Space: Defining All Possible Outcomes

Before discussing the probability of something happening, we must first define what can possibly happen. The complete set of all potential outcomes of a random experiment is called the **sample space**, denoted by the Greek letter Ω (omega). What counts as an “outcome” depends on the experiment itself:

- **Flipping a coin once:** $\Omega = \{H, T\}$
- **Rolling a six-sided die:** $\Omega = \{1, 2, 3, 4, 5, 6\}$
- **Flipping a coin twice:** $\Omega = \{HH, HT, TH, TT\}$
- **Drawing a card from a deck:** $\Omega = \{\text{all 52 cards}\}$
- **Measuring temperature:** $\Omega = \{t \in \mathbb{R} \mid -50 \leq t \leq 60\}$
- **Email classification:** $\Omega = \{\text{spam, not spam}\}$

Some of these examples, such as the coin or die, are **discrete** because the number of possible outcomes is finite. Others, like measuring temperature, are **continuous**, where outcomes can take infinitely many values within an interval.

Defining the sample space correctly is the **first step in solving any probability problem** — it ensures that all possible outcomes are accounted for before assigning probabilities.

0.1.2 Events: Subsets of Interest

An **event** is any **subset** of the sample space Ω. If the outcome of the experiment lies within that subset, we say that the event has occurred. Events allow us to focus on specific situations of interest within a broader context.

Rolling a die $(\Omega = \{1, 2, 3, 4, 5, 6\})$:

$$A = \{6\} \text{ (rolling a six)}, \quad B = \{2, 4, 6\} \text{ (rolling an even number)}, \quad C = \{4, 5, 6\} \text{ (rolling a number g}$$

Flipping two coins $(\Omega = \{HH, HT, TH, TT\})$:

$$D = \{HT, TH\} \text{ (exactly one head)}, \quad E = \{HH, HT, TH\} \text{ (at least one head)}.$$

Drawing a card:

$$F = \{\text{all hearts}\} \text{ (drawing a heart)}, \quad G = \{\text{Jack, Queen, King of any suit}\} \text{ (drawing a face card)}.$$

Temperature measurement:

$$H = \{t \in \Omega \mid t \leq 0\} \text{ (temperature below freezing)}, \quad I = \{t \in \Omega \mid 20 \leq t \leq 25\} \text{ (temperature betwee}$$

Email classification:

$$J = \{\text{email predicted as spam}\}, \quad K = \{\text{email actually is spam}\}.$$

Two special events exist for every sample space:

- The **certain event** (Ω): always occurs since every outcome is in Ω.
- The **impossible event** ($\emptyset$): never occurs since it contains no outcomes.

0.1.3 Combining Events Using Set Operations

Since events are **sets**, we can use standard **set operations** to create new events from existing ones. This is essential for understanding relationships between different occurrences.

- **Union ($A \cup B$):** represents the event that *at least one* of A or B occurs.

  $$\text{Example: } A = \{2,4,6\},\ B = \{4,5,6\} \Rightarrow A \cup B = \{2,4,5,6\}.$$

  This means "getting an even number or a number greater than 3."

- **Intersection ($A \cap B$):** represents the event that *both* A and B occur.

  $A \cap B = \{4,6\}$ means "getting a number that is both even and greater than 3."

- **Complement (A^c):** represents the event that A *does not occur.*

  $$A = \{2,4,6\} \Rightarrow A^c = \{1,3,5\} \text{ (rolling an odd number).}$$

- **Difference ($A - B$):** represents the event that A occurs but B does not.

  $$A - B = \{2\} \text{ means "rolling an even number that is not greater than 3."}$$

In practical applications, these operations describe relationships like “false positives” or “false negatives” in classification problems, or overlapping situations such as “rainy and windy days.”

Finally, two important identities known as **De Morgan’s Laws** describe how complements interact with unions and intersections:

$$(A \cup B)^c = A^c \cap B^c \quad \text{and} \quad (A \cap B)^c = A^c \cup B^c.$$

These laws are fundamental for logical reasoning and are widely used in both probability and computer science.

0.2 Conditional Probability and Independence

In many real-world situations, the likelihood of an event changes when we gain new information. For instance, the probability of rain increases if we know the sky is cloudy. This idea — updating probabilities as new evidence arises — is central to the concept of **conditional probability**.

0.2.1 Understanding Conditional Probability

Often, we are interested in the probability that an event A occurs *given* that another event B has already happened. This is known as the **conditional probability** of A given B, written as $P(A \mid B)$. Conceptually, it represents how our degree of belief in A changes once we learn that B has occurred.

The key idea is that the occurrence of B effectively *reduces the sample space*: instead of considering all possible outcomes, we now focus only on those where B has happened. Within this restricted sample space, we measure how often A also occurs.

The formal definition of conditional probability is:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}, \quad \text{for } P(B) > 0$$

Here, $P(A \cap B)$ is the probability that both events occur, and $P(B)$ is the probability of B. This formula can be interpreted as “the fraction of B’s outcomes that also belong to A."

0.2.2 Example: Email Filtering — The Word "Offer"

Suppose we are analyzing emails to determine whether they are spam. Let:

- S = the event that an email is spam
- O = the event that an email contains the word "offer"

From a large dataset, we have:

$$P(S) = 0.3, \quad P(O) = 0.2, \quad P(S \cap O) = 0.15$$

We want to know how likely an email is spam given that it includes the word "offer." Applying the formula:

$$P(S \mid O) = \frac{P(S \cap O)}{P(O)} = \frac{0.15}{0.20} = 0.75$$

This means that knowing an email contains the word "offer" increases our belief that it is spam from $P(S) = 0.3$ to $P(S \mid O) = 0.75$. Modern spam filters rely on many such probabilities — not just single words but also phrases, links, or sender behavior. Each contributes a piece of evidence that, when combined (often using **Naïve Bayes** or similar models), leads to an overall classification.

0.2.3 Example: Medical Diagnosis

Let:

- D = a patient has a certain disease
- T = the test result is positive

Suppose:

$$P(D) = 0.01, \quad P(T \mid D) = 0.95, \quad P(T \mid D^c) = 0.05$$

Using these, we can later compute $P(D \mid T)$ — the probability that the patient has the disease given a positive test. Conditional probability allows medical practitioners to interpret test results properly, balancing false positives and false negatives.

0.2.4 Independence of Events

Sometimes, knowing that one event occurred gives us *no information* about another. When this happens, the events are said to be **independent**.

Formally, events A and B are independent if:

$$P(A \mid B) = P(A) \quad \text{or equivalently,} \quad P(A \cap B) = P(A)P(B)$$

In other words, the occurrence of one event does not affect the likelihood of the other.

0.2.5 Example: Independent Events — Coin Flips

Consider flipping a fair coin twice:

- A = getting heads on the first flip
- B = getting heads on the second flip

We have $P(A) = 0.5$, $P(B) = 0.5$, and $P(A \cap B) = 0.25$. Since $P(A \cap B) = P(A)P(B) = 0.25$, the events are independent. The result of the first flip tells us nothing about the second — a classic example of independence.

0.2.6 Example: Dependent Events — Drawing Cards Without Replacement

Now consider drawing two cards from a deck **without replacement**:

- A = the first card is an Ace
- B = the second card is an Ace

Initially, $P(A) = \frac{4}{52}$. If the first card is an Ace, there are only 3 Aces left among 51 cards, so $P(B \mid A) = \frac{3}{51}$. If the first card is not an Ace, then $P(B \mid A^c) = \frac{4}{51}$.

Since $P(B \mid A) \neq P(B)$, the events are **dependent** — the first draw changes the probability of the second.

0.2.7 Example: Weather and Traffic

Let:

- R = "it rains in the morning"
- T = "traffic is heavy during the commute"

Typically, $P(T \mid R) > P(T)$; when it rains, heavy traffic becomes more likely. This dependency shows how environmental or contextual factors can influence probabilities in real-world systems.

0.2.8 Why These Concepts Matter in Machine Learning

Understanding **conditional probability** and **independence** is fundamental to many machine learning and statistical methods:

- **Probabilistic Models:** Algorithms such as the Naïve Bayes Classifier use these principles directly. They rely on the assumption that features are conditionally independent given the class label, which simplifies computation.
- **Feature Relationships:** Conditional probabilities reveal dependencies between variables in a dataset, such as $P(\text{disease} \mid \text{symptom})$ or $P(\text{purchase} \mid \text{ad click})$.
- **Bayesian Inference:** The entire Bayesian approach to machine learning revolves around updating beliefs as new evidence arrives — exactly what conditional probability formalizes.
- **Model Evaluation:** Recognizing independence and dependence helps us identify when model assumptions are realistic and when they might fail.

Mastering how to compute and interpret $P(A \mid B)$, and how to recognize independent or dependent events, is a vital step toward understanding **Bayes' Theorem**, which provides the mathematical foundation for reasoning under uncertainty.

0.3 Bayes' Theorem and How It Works

In probability theory, **Bayes' Theorem** provides a method to reverse the direction of conditional probability. While $P(A \mid B)$ describes the probability of event A occurring given that B has already happened, Bayes' Theorem allows us to find the reverse — $P(B \mid A)$ — the probability that B is true once A is observed.

This principle lies at the heart of reasoning under uncertainty and helps us update our understanding of the world as new information arrives. Named after Reverend **Thomas Bayes**, this theorem forms the foundation of Bayesian inference and plays a central role in modern statistics and machine learning.

0.3.1 The Core Formula

Bayes' Theorem is formally expressed as:

$$P(B \mid A) = \frac{P(A \mid B)\, P(B)}{P(A)}, \quad \text{where } P(A) > 0$$

Here:

- $P(B)$ is the **prior probability**: our initial belief in event B before seeing any data.
- $P(A \mid B)$ is the **likelihood**: how consistent the observed data A is with B being true.
- $P(A)$ is the **evidence**: the overall probability of observing A under all possible causes.
- $P(B \mid A)$ is the **posterior probability**: our updated belief about B after observing A.

In essence, Bayes' Theorem provides a systematic rule for revising our beliefs as new information becomes available.

0.3.2 Example: Weather Prediction

Imagine designing a system to predict rain based on morning weather conditions. Let:

- R = event that it will rain today
- C = event that the sky is cloudy in the morning

From historical data, we know:

$$P(R) = 0.3, \quad P(C) = 0.4, \quad P(C \mid R) = 0.8$$

We want to find the probability that it will rain *given that the sky is cloudy*, $P(R \mid C)$.

Using Bayes' Theorem:

$$P(R \mid C) = \frac{P(C \mid R)P(R)}{P(C)} = \frac{0.8 \times 0.3}{0.4} = 0.6$$

Thus, the probability of rain increases from 0.3 (our prior belief) to 0.6 after observing a cloudy morning. This example illustrates how Bayes' Theorem allows us to refine our predictions as new evidence emerges — a concept widely used in forecasting, diagnostics, and AI systems.

0.3.3 Why Bayes' Theorem Matters in Machine Learning

Bayes' Theorem underpins many probabilistic models in machine learning and data science.

- **Learning from New Data:** It offers a formal framework for updating knowledge as more data becomes available. In *Bayesian learning*, model parameters are treated as random variables whose probabilities evolve with new evidence.

- **Classification Models:** The *Naïve Bayes Classifier* directly applies Bayes' rule to compute the probability of a class given a set of features, $P(\text{Class} \mid \text{Features})$, assuming the features are conditionally independent.
- **Decision-Making Under Uncertainty:** Bayesian reasoning allows algorithms to make rational decisions even when data is incomplete or noisy — for instance, in natural language processing, robotics, and recommendation systems.

0.4 Introduction to Random Variables

In probability, we often begin with the idea of a **sample space**, which lists all possible outcomes of a random process, and **events**, which are subsets of that space. However, in many cases, we are not concerned with the exact outcome itself but rather with a numerical quantity that summarizes some aspect of it.

For example, consider flipping a coin three times. The sample space is:

$$S = \{HHH, HHT, HTH, THH, HTT, THT, TTH, TTT\}.$$

Each element represents one possible sequence of heads (H) and tails (T). While the full sequence gives complete information, we might instead be interested in a simpler measure — such as the total number of heads obtained. This leads to the concept of a **random variable**.

0.4.1 Definition

A **random variable** is a function that assigns a real number to each outcome in the sample space:

$$X : \Omega \to \mathbb{R}.$$

Random variables are typically denoted by capital letters such as X, Y, or Z. They serve as numerical summaries of random phenomena, allowing us to describe outcomes quantitatively rather than symbolically.

0.4.2 Example: Counting Heads in Three Coin Flips

Let X represent the *number of heads* obtained when flipping a fair coin three times. Then:

$$X \in \{0, 1, 2, 3\}.$$

Each outcome in the sample space corresponds to a value of X:

Outcome	X **(Number of Heads)**
TTT	0
HTT, THT, TTH	1
HHT, HTH, THH	2
HHH	3

Since each outcome has probability 1/8, we can compute:

$$P(X = 0) = \tfrac{1}{8},$$
$$P(X = 1) = \tfrac{3}{8},$$
$$P(X = 2) = \tfrac{3}{8},$$
$$P(X = 3) = \tfrac{1}{8}.$$

Thus, X provides a concise numerical description of the experiment — summarizing the number of heads without tracking every specific sequence.

0.4.3 Example: Rolling Two Dice

Let Y denote the *sum* of the numbers obtained when rolling two fair six-sided dice. The sample space contains 36 equally likely outcomes, but we can describe the results entirely in terms of Y:

$$Y \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

For example:

- $Y = 2$ occurs in one way: $(1, 1)$

- $Y = 3$ occurs in two ways: $(1, 2)$, $(2, 1)$
- $Y = 7$ occurs in six ways: $(1, 6)$, $(2, 5)$, $(3, 4)$, $(4, 3)$, $(5, 2)$, $(6, 1)$

By assigning probabilities to each possible value of Y, we obtain a complete probability distribution that captures the behavior of the sum of two dice. This example illustrates how random variables simplify the analysis of probabilistic systems.

0.4.4 Types of Random Variables

Random variables are classified into two main types:

1. **Discrete Random Variables:** These take on a countable number of distinct values. Examples include the number of heads in coin flips, the number of customers arriving at a store, or the number of emails received in an hour.
2. **Continuous Random Variables:** These can assume any value within a continuous interval. Examples include a person's height, the arrival time of a bus, or the temperature at noon.

0.4.5 Example: Continuous Random Variable — Measuring Temperature

Suppose we record the midday temperature in a city during summer. Let Z denote the temperature (in Celsius) on a given day. The variable Z can take on any real value within a range, for example:

$$Z \in [20, 45].$$

Here, Z is a **continuous random variable** because it can take infinitely many possible values within this interval.

We might describe the probability that the temperature lies between two values using a probability density function (PDF), $f_Z(z)$. For instance:

$$P(25 \leq Z \leq 30) = \int_{25}^{30} f_Z(z)\, dz.$$

Unlike discrete probabilities, which assign values to individual outcomes, the PDF spreads probability continuously over an interval, making continuous variables essential for modeling physical quantities like temperature, speed, or time.

0.5 Mean and Variance

Random variables represent numerical outcomes that arise from uncertain processes. To analyze and describe their behavior, it is often useful to summarize how the values are distributed. Two fundamental concepts provide this summary: the **mean**, which indicates the central or average value, and the **variance**, which measures how much the values deviate from that center.

0.5.1 Mean (or Average)

The **mean** of a random variable X, often denoted by μ or $E[X]$, represents the long-run average value we would expect if the random process were repeated many times. In other words, it tells us where the distribution of X tends to balance.

Discrete Random Variables

If X can take values $x_1, x_2, \ldots, x_n$ with respective probabilities $P(X = x_1), P(X = x_2), \ldots, P(X = x_n)$, then the mean is given by:

$$\mu = E[X] = \sum_i x_i P(X = x_i)$$

Continuous Random Variables

If X is continuous with probability density function $f(x)$, then the mean is computed as:

$$\mu = E[X] = \int_{-\infty}^{\infty} x f(x)\, dx$$

Example (Average Daily Temperature).
Suppose the temperature X in a certain city on a spring day is modeled as a continuous random variable with density function $f(x)$ over the range $[10, 30]$ degrees

Celsius. The mean temperature is obtained by:

$$\mu = \int_{10}^{30} x f(x)\, dx$$

If $f(x)$ is approximately uniform, then $\mu \approx 20$, meaning the typical temperature centers around 20°C.

0.5.2 Variance and Standard Deviation

While the mean provides a measure of the central value, it does not indicate how spread out the data are. The **variance** measures this dispersion. It tells us how much, on average, the values of X differ from the mean μ.

$$\text{Var}(X) = E[(X - \mu)^2]$$

For discrete variables:

$$\text{Var}(X) = \sum_i (x_i - \mu)^2 P(X = x_i)$$

For continuous variables:

$$\text{Var}(X) = \int_{-\infty}^{\infty} (x - \mu)^2 f(x)\, dx$$

An equivalent computational formula, often easier to use, is:

$$\text{Var}(X) = E[X^2] - (E[X])^2$$

Example (Customer Waiting Time).
Let X be the waiting time (in minutes) for customers at a coffee shop, which follows the probability distribution shown below:

$$X = \{1, 2, 3, 4, 5\}, \quad P(X = x) = \{0.1, 0.2, 0.4, 0.2, 0.1\}.$$

Then the mean waiting time is:

$$E[X] = (1)(0.1) + (2)(0.2) + (3)(0.4) + (4)(0.2) + (5)(0.1) = 3.$$

The variance is:

$$\text{Var}(X) = \sum (x-3)^2 P(X=x) = 1.2.$$

Hence, the standard deviation is:

$$\sigma = \sqrt{1.2} \approx 1.095.$$

This indicates that most customers wait within about one minute of the average waiting time.

0.5.3 Standard Deviation

The **standard deviation** is the square root of the variance:

$$\sigma = \sqrt{\text{Var}(X)}.$$

It is expressed in the same units as the original variable, making it easier to interpret. A smaller standard deviation means values are closely clustered around the mean, while a larger one indicates greater variability.

0.6 Applying Probability Concepts in Python

This section demonstrates how probability theory can be explored and verified using Python. Through simulation, we can visualize how theoretical ideas — such as event likelihood, conditional probability, and Bayes' theorem — appear in practice. This hands-on approach helps bridge the gap between mathematical definitions and real-world data behavior.

0.6.1 Simulating Sample Spaces and Events

Let us begin with a simple example: flipping a fair coin. The sample space is

$$\Omega = \{H, T\}.$$

Instead of performing the experiment manually, we can simulate a large number of coin tosses using Python to estimate probabilities. Below, we simulate 10,000 tosses and compute the probability of getting a head.

```
import numpy as np

# Simulate 10,000 fair coin tosses
num_tosses = 10000
tosses = np.random.choice(['H', 'T'], size=num_tosses)

# Define Event A: getting a Head
is_head = (tosses == 'H')
num_heads = np.sum(is_head)

# Empirical probability
prob_head_empirical = num_heads / num_tosses

# Theoretical probability
prob_head_theoretical = 0.5

print(f"First 10 tosses: {tosses[:10]}")
print(f"Number of heads: {num_heads}")
print(f"Empirical probability of head: {prob_head_empirical:.4f}")
print(f"Theoretical probability: {prob_head_theoretical:.4f}")
```

As the number of tosses increases, the simulated probability of getting a head approaches the theoretical value of 0.5. This demonstrates the **Law of Large Numbers**, which states that empirical averages converge to their expected theoretical values as the number of trials increases.

0.6.2 Conditional Probability

Conditional probability tells us how the likelihood of one event changes when we know another has occurred.

Suppose we toss two fair coins. Define:

$$A = \text{“first coin shows heads”}, \quad B = \text{“both coins show the same result”}.$$

We will simulate these events and estimate $P(A|B)$.

```
# Simulate 10,000 two-coin tosses
num_trials = 10000
first_coin = np.random.choice(['H', 'T'], size=num_trials)
second_coin = np.random.choice(['H', 'T'], size=num_trials)

# Define events
is_first_head = (first_coin == 'H')  # Event A
is_same = (first_coin == second_coin)  # Event B

# Conditional probability P(A|B)
A_given_B = np.sum(is_first_head & is_same) / np.sum(is_same)

print(f"Empirical P(A|B): {A_given_B:.4f}")
print("Theoretical P(A|B): 0.5")
```

The results should be close to 0.5, showing that the probability of the first coin being heads is independent of whether both coins show the same result. This simulation confirms the **independence of events**.

0.6.3 Implementing Bayes' Theorem

Bayes' theorem provides a systematic way to update probabilities based on new evidence. It is expressed as:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}.$$

To make this more concrete, consider a **spam email classifier**. Let S be the event that an email is spam, and W be the event that the email contains the word "offer." Suppose we know:

$$P(S) = 0.2, \quad P(W|S) = 0.7, \quad P(W|\neg S) = 0.1.$$

We want to find $P(S|W)$, the probability that an email is spam given that it contains the word "offer."

```
# Given probabilities
P_S = 0.2              # Prior probability of spam
P_W_given_S = 0.7      # "Offer" in spam
P_W_given_notS = 0.1   # "Offer" in non-spam

# Derived probabilities
P_notS = 1 - P_S
P_W = P_W_given_S * P_S + P_W_given_notS * P_notS  # Total probability of "offer"
P_S_given_W = (P_W_given_S * P_S) / P_W             # Bayes' theorem

print(f"P(W): {P_W:.4f}")
print(f"P(S|W): {P_S_given_W:.4f}")
```

The result shows that even though the word "offer" is common in spam, the probability of an email being spam given that it contains the word is still below 0.65. This example highlights the importance of considering both prior probabilities and false positive rates — the core intuition behind **Bayesian reasoning** in machine learning.

0.6.4 Simulating Random Variables

We can also use Python to simulate random variables directly by sampling from probability distributions. While later sections will introduce well-known distributions in detail, we can begin with a simple custom discrete random variable.

Consider a biased coin where the probability of getting Heads (H) is $P(H) = 0.7$

and the probability of getting Tails (T) is $P(T) = 0.3$. Let us define:

$$X = \begin{cases} 1, & \text{if Head occurs} \\ 0, & \text{if Tail occurs.} \end{cases}$$

We can simulate this random variable using NumPy:

```
# Define probabilities for the outcomes (0 for Tails, 1 for Heads)
outcomes = [0, 1]
probabilities = [0.3, 0.7]  # P(X=0) = 0.3, P(X=1) = 0.7

# Simulate 1,000 flips of the biased coin
num_flips = 1000
flips = np.random.choice(outcomes, size=num_flips, p=probabilities)

# Calculate empirical probabilities
num_tails = np.sum(flips == 0)
num_heads = np.sum(flips == 1)

prob_tails_empirical = num_tails / num_flips
prob_heads_empirical = num_heads / num_flips

print(f"Simulated {num_flips} biased coin flips.")
print(f"Number of Tails (X=0): {num_tails}, Empirical Probability: {prob_tails_empi
print(f"Number of Heads (X=1): {num_heads}, Empirical Probability: {prob_heads_empi
```

The empirical probabilities from the simulation should closely match their theoretical values as the number of flips increases. This example illustrates how random variables can be represented and explored through **simulation**, forming the foundation for statistical and machine learning methods.

0.7 Introduction to Probability Distributions

Building upon the foundational concepts of probability covered earlier, this section explores the mathematical functions that describe how probabilities are distributed across the possible values of a random variable. These functions, known as **probability distributions**, are essential for representing uncertainty — a core element in both statistics and machine learning.

In this section, we introduce several fundamental probability distributions that are widely used in statistical modeling and serve as building blocks for many machine learning algorithms. You will learn to:

- Recognize and explain key **discrete distributions** such as the Bernoulli, Binomial (n, p), and Poisson (λ) distributions.
- Understand major **continuous distributions** including the Normal distributions (μ, σ^2).
- Interpret the **properties and real-world applications** that make each distribution suitable for modeling different types of random phenomena.
- Use **Python libraries**, particularly `SciPy`, to compute probabilities (such as probability mass and density functions), evaluate cumulative probabilities, generate random samples, and visualize these distributions.

0.8 Bernoulli and Binomial Distributions

The **Bernoulli** and **Binomial** distributions are two of the most fundamental discrete probability models. They describe random processes that result in only two possible outcomes, such as *success/failure*, *yes/no*, or *true/false*. These models appear frequently in applications like spam detection, click-through prediction, quality control, and A/B testing.

0.8.1 The Bernoulli Distribution: A Single Binary Event

The **Bernoulli distribution** represents the simplest random variable—one that can take only two outcomes. Typical examples include:

- Flipping a coin (Head or Tail)
- A user clicking or not clicking an advertisement

- An email being spam or not spam

Let X be a random variable such that:

$$X = \begin{cases} 1 & \text{if success occurs} \\ 0 & \text{if failure occurs} \end{cases}$$

Then X follows a Bernoulli distribution with parameter p, where $p = P(X = 1)$ represents the probability of success. The probability of failure is $P(X = 0) = 1 - p$.

Probability Mass Function (PMF):

$$P(X = k \mid p) = p^k (1-p)^{1-k}, \quad k \in \{0, 1\}$$

Key Properties:

- **Mean:** $E[X] = p$
- **Variance:** $\text{Var}(X) = p(1-p)$

The Bernoulli distribution provides the foundation for the **Binomial distribution**, which models multiple independent Bernoulli trials.

0.8.2 The Binomial Distribution: Repeated Bernoulli Trials

The **Binomial distribution** extends the Bernoulli case to n independent and identical trials, each with probability of success p. It answers questions such as:

> "If we flip a biased coin 10 times, what is the probability of getting exactly 6 heads?"

A random variable X follows a Binomial distribution if it represents the number of successes in n independent Bernoulli trials. We write:

$$X \sim \text{Binomial}(n, p)$$

Probability Mass Function (PMF):

$$P(X = k \mid n, p) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, 2, \ldots, n$$

where:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

represents the number of ways to achieve k successes out of n trials.

Key Properties:

- **Mean:** $E[X] = np$
- **Variance:** $\mathrm{Var}(X) = np(1-p)$

Example: Suppose a biased coin has a probability $p = 0.6$ of landing heads, and it is flipped $n = 5$ times. The probability of getting exactly $k = 3$ heads is:

$$P(X = 3) = \binom{5}{3} (0.6)^3 (0.4)^2 = 10 \times 0.216 \times 0.16 = 0.3456$$

Thus, there is approximately a 34.6% chance of observing exactly three heads.

0.8.3 Working with Bernoulli and Binomial Distributions in Python

We can use the `scipy.stats` module in Python to calculate probabilities, generate random samples, and visualize these distributions.

```
import numpy as np
from scipy.stats import bernoulli, binom
import matplotlib.pyplot as plt

# --- Bernoulli Example ---
p_success = 0.7  # Probability of success (e.g., ad click)
rv_bern = bernoulli(p_success)
```

```
print(f"P(X=1): {rv_bern.pmf(1):.4f}")
print(f"P(X=0): {rv_bern.pmf(0):.4f}")
print(f"Mean: {rv_bern.mean():.4f}, Variance: {rv_bern.var():.4f}")
print(f"Samples: {rv_bern.rvs(size=10)}")

# --- Binomial Example ---
n_trials = 10
p_success_bin = 0.2
rv_binom = binom(n_trials, p_success_bin)

k = 3
print(f"P(X={k}): {rv_binom.pmf(k):.4f}")
print(f"P(X<={k}): {rv_binom.cdf(k):.4f}")
print(f"Mean: {rv_binom.mean():.4f}, Variance: {rv_binom.var():.4f}")
print(f"Samples: {rv_binom.rvs(size=15)}")

# --- Plotting Binomial PMF ---
k_values = np.arange(0, n_trials + 1)
pmf_values = rv_binom.pmf(k_values)

plt.bar(k_values, pmf_values)
plt.title("Binomial Distribution (n=10, p=0.2)")
plt.xlabel("Number of successes (k)")
plt.ylabel("Probability P(X=k)")
plt.show()
```

This Python example shows how to compute and visualize Bernoulli and Binomial probabilities. By experimenting with different values of n and p, you can observe how the distribution changes — for instance, increasing p shifts the probability mass toward higher numbers of successes.

0.9 Poisson Distribution

The **Poisson distribution** is a key discrete probability model that describes the number of times an event occurs within a fixed interval of time or space. While the **Binomial distribution** focuses on the number of successes in a fixed number of independent trials, the Poisson distribution instead models the *count* of events that happen independently but at a known constant average rate.

This makes it particularly useful for modeling real-world count data such as:

- The number of emails received per hour.
- The number of cars passing through a toll booth per minute.
- The number of defects in a batch of manufactured products.
- The number of customer arrivals at a store within an hour.

0.9.1 Definition

A discrete random variable X follows a Poisson distribution with rate parameter λ (lambda) if it represents the number of events occurring in a given interval, where $\lambda > 0$ denotes the expected number of events during that interval.

We write:

$$X \sim \text{Poisson}(\lambda)$$

The **probability mass function (PMF)** is given by:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad k = 0, 1, 2, \ldots$$

where:

- k is the observed number of events,
- $e \approx 2.71828$ is the base of the natural logarithm,

This formula expresses the probability of observing exactly k events in a given interval, given that the expected rate of occurrence is λ.

0.9.2 Properties

The Poisson distribution has several notable properties:

- **Mean:** $E[X] = \lambda$
- **Variance:** $\mathrm{Var}(X) = \lambda$

An important feature of this distribution is that its mean and variance are equal, a characteristic that often helps identify Poisson behavior in empirical data.

0.9.3 Applications in Machine Learning

The Poisson distribution appears frequently in statistical modeling and machine learning:

1. **Modeling Count Data:** Used to model the number of occurrences, such as website visits, email arrivals, or event counts within fixed time windows.
2. **Queuing Systems:** Describes arrivals of customers, packets, or tasks over time in service systems.
3. **Risk and Reliability Analysis:** Models rare events such as equipment failures, insurance claims, or accidents.
4. **Poisson Regression:** A type of generalized linear model (GLM) that models a count-based response variable as a function of explanatory features.

.

0.10 Normal (Gaussian) Distribution

The **Normal distribution**, also known as the **Gaussian distribution** or the **bell curve**, is one of the most widely used continuous probability distributions in statistics and machine learning. It models many natural phenomena such as human height, exam scores, or measurement errors. Because of its frequent appearance in real-world data, the Normal distribution forms the foundation of many statistical and machine learning techniques.

Definition

A continuous random variable X follows a Normal distribution if its probability density function (PDF) is given by:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

This distribution is completely determined by two parameters:

- **Mean (μ)** – Determines the center or location of the curve. It is the point around which data are symmetrically distributed.
- **Variance (σ^2)** – Measures the spread or width of the curve. A larger variance produces a wider, flatter curve, while a smaller variance results in a narrower, taller curve. The standard deviation $\sigma = \sqrt{\sigma^2}$ has the same units as X.

We denote this as:

$$X \sim \text{Normal}(\mu, \sigma^2)$$

The Standard Normal Distribution

A special case of the Normal distribution is the **Standard Normal Distribution**, denoted as Z, which has a mean of 0 and a standard deviation of 1:

$$Z \sim \text{Normal}(0, 1)$$

Its PDF is:

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

Any Normal random variable $X \sim \text{Normal}(\mu, \sigma^2)$ can be transformed into a standard form using the **Z-score transformation**:

$$Z = \frac{X - \mu}{\sigma}$$

The Z-score measures how many standard deviations a value X lies from the mean

μ. This transformation is particularly useful for:

- **Comparison:** Allows comparing values from different Normal distributions by converting them to the same scale.
- **Probability Calculation:** Enables computation of probabilities using the cumulative distribution function (CDF) of the Standard Normal distribution, denoted by $\Phi(z)$.

Thus, for any $X \sim \text{Normal}(\mu, \sigma^2)$:

$$P(X \leq x) = P\left(Z \leq \frac{x-\mu}{\sigma}\right) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

Relevance in Machine Learning

The Normal distribution plays a crucial role in data analysis and machine learning due to its mathematical simplicity and empirical significance:

- **Modeling Errors:** In linear regression, residuals (errors) are often assumed to follow a Normal distribution.
- **Feature Transformation:** Many algorithms perform better when input features follow a Normal distribution; transformations such as log or Box–Cox are used to achieve this.
- **Initialization of Neural Networks:** Weights are often initialized with values drawn from a Normal distribution to ensure stable learning.
- **Probabilistic Models:** Algorithms like Gaussian Naive Bayes and Linear Discriminant Analysis (LDA) rely on Normal distribution assumptions.
- **Central Limit Theorem (CLT):** This theorem states that the sampling distribution of the mean of any independent random variables tends to be Normal as the sample size increases, justifying many statistical methods.

0.11 Working with Distributions in SciPy

The **SciPy** library provides powerful and convenient tools for working with probability distributions in Python. Its `scipy.stats` module includes a rich collection of both discrete and continuous distributions, making it an essential component for simulation, statistical modeling, and many machine learning applications.

Each distribution in `scipy.stats` follows a consistent interface and supports a range of useful operations:

- **PDF (Probability Density Function):** For continuous distributions, the PDF $f(x)$ describes the likelihood of observing a value near x. Accessed using the `.pdf()` method.
- **PMF (Probability Mass Function):** For discrete distributions, the PMF $P(X = k)$ gives the probability that a random variable X takes a specific value k. Accessed using the `.pmf()` method.
- **CDF (Cumulative Distribution Function):** The CDF $F(x) = P(X \leq x)$ gives the probability that X is less than or equal to a given value. Accessed using `.cdf()`.
- **PPF (Percent Point Function):** Also called the inverse CDF or quantile function. Given a probability p, the PPF finds the value x such that $F(x) = p$. Accessed using `.ppf()`.
- **RVS (Random Variates Sampling):** Generates random numbers that follow a given distribution. Accessed using `.rvs()`.

These methods make it easy to compute probabilities, generate samples, and visualize how distributions behave — all from a unified interface.

0.11.1 Working with Continuous Distributions: The Normal Distribution

The **Normal (Gaussian)** distribution is one of the most commonly used continuous distributions. In `scipy.stats`, it is represented by `norm`. We define its mean (μ) using the `loc` parameter and its standard deviation (σ) using the `scale` parameter.

```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Define a Normal distribution with mean 0 and standard deviation 2
mu = 0
sigma = 2
my_normal = norm(loc=mu, scale=sigma)

# Compute probabilities
print("PDF at x=1:", my_normal.pdf(1))
print("CDF at x=1:", my_normal.cdf(1))

# Compute quantiles
print("95th percentile:", my_normal.ppf(0.95))

# Generate random samples
samples = my_normal.rvs(size=5)
print("Five random samples:", samples)

# Prepare data for plotting the PDF
x = np.linspace(mu - 4*sigma, mu + 4*sigma, 200)
pdf_values = my_normal.pdf(x)

# Plot the PDF
plt.plot(x, pdf_values, label="Normal PDF")
plt.title("Normal Distribution ( =0,  =2)")
plt.xlabel("x")
plt.ylabel("Density")
plt.legend()
plt.show()
```

Listing 1: Working with the Normal Distribution in SciPy

This example demonstrates how to compute probabilities, quantiles, and random samples. The `.pdf()` and `.cdf()` methods help visualize the distribution's shape, while `.rvs()` enables random sampling — useful for simulations or testing models.

0.11.2 Working with Discrete Distributions: The Binomial Distribution

The **Binomial** distribution models the number of successes in a fixed number of independent Bernoulli trials, each with probability p of success. In `scipy.stats`, it is represented by `binom`.

```python
from scipy.stats import binom
import matplotlib.pyplot as plt
import numpy as np

# Define a Binomial distribution: n=10 trials, p=0.5
n = 10
p = 0.5
my_binomial = binom(n=n, p=p)

# Compute PMF and CDF
print("PMF at k=5:", my_binomial.pmf(5))
print("CDF at k=5:", my_binomial.cdf(5))

# Compute quantile
print("90th percentile:", my_binomial.ppf(0.9))

# Generate random samples
samples = my_binomial.rvs(size=10)
print("Random samples (number of successes):", samples)

# Prepare data for plotting
k = np.arange(0, n + 1)
pmf_values = my_binomial.pmf(k)

# Plot the PMF
plt.stem(k, pmf_values, basefmt=" ")
plt.title("Binomial Distribution (n=10, p=0.5)")
plt.xlabel("Number of Successes")
plt.ylabel("Probability")
plt.show()
```

Listing 2: Working with the Binomial Distribution in SciPy

This example shows how the same interface applies to discrete distributions. You can compute probabilities (`.pmf()`), cumulative probabilities (`.cdf()`), quantiles (`.ppf()`), and random samples (`.rvs()`), just like with continuous distributions.

0.12 Simulation and Visualization of Probability Distributions

Simulation and visualization play a crucial role in developing an intuitive understanding of probability distributions. While mathematical expressions provide the theoretical foundation, observing these distributions through simulation helps in building practical intuition. This is especially important when testing whether a specific distribution fits empirical data — a frequent task in statistics and machine learning.

In this section, we use `Python` and popular libraries such as `NumPy` for numerical computation, `SciPy` for probability distributions, and `Matplotlib` or `Seaborn` for visualization. For interactive use, tools like `Plotly` can be integrated as well.

0.12.1 Library Setup

Before proceeding, ensure that all required libraries are installed and imported:

```python
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns

# Optional: set style for Seaborn plots
sns.set_style("whitegrid")
```

Listing 3: Importing required libraries

0.12.2 Simulating and Visualizing a Discrete Distribution: The Binomial Case

The Binomial distribution models the number of successes in a fixed number of independent Bernoulli trials, each with a constant probability of success p. For example, flipping a biased coin multiple times can be modeled as a sequence of Bernoulli trials.

Assume a biased coin with $p = 0.6$ (probability of heads) is flipped $n = 20$ times per experiment, and this experiment is repeated 1000 times. We can simulate the number of heads obtained in each experiment using `scipy.stats.binom.rvs()`:

```python
# Parameters for the Binomial distribution
n_binom = 20         # number of trials
p_binom = 0.6        # probability of success
size_binom = 1000    # number of experiments

# Simulate Binomial outcomes
binomial_samples = stats.binom.rvs(n=n_binom, p=p_binom, size
    =size_binom)

print("First 10 simulated outcomes (number of successes):")
print(binomial_samples[:10])
```

Listing 4: Simulating Binomial outcomes

To visualize, we compare the empirical histogram from the simulation with the theoretical probability mass function (PMF):

```python
# Calculate theoretical PMF
k_values = np.arange(0, n_binom + 1)
pmf_values = stats.binom.pmf(k_values, n=n_binom, p=p_binom)

plt.figure(figsize=(10, 6))
plt.hist(binomial_samples, bins=k_values, density=True, alpha
    =0.6,
         color='#4dabf7', label='Simulated Data')
plt.plot(k_values, pmf_values, 'o-', color='#f03e3e', label='
    Theoretical PMF')

```

```
plt.xlabel(f"Number of Successes (k) in {n_binom} Trials")
plt.ylabel("Probability / Density")
plt.title(f"Binomial Distribution (n={n_binom}, p={p_binom})
       Simulation vs. Theory")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 5: Visualizing Binomial distribution

As the number of simulations increases, the histogram approaches the shape of the theoretical PMF, illustrating the **Law of Large Numbers** in action.

0.12.3 Simulating and Visualizing a Continuous Distribution: The Normal Case

The Normal (Gaussian) distribution is one of the most fundamental continuous probability distributions. It is defined by its mean μ and standard deviation σ, and is widely used in modeling natural phenomena, measurement errors, and residuals in regression models.

We simulate data from a standard Normal distribution ($\mu = 0, \sigma = 1$) and compare the histogram of simulated values with the theoretical probability density function (PDF):

```
# Parameters for the Normal distribution
mu_norm = 0
sigma_norm = 1
size_norm = 1000

# Simulate Normal samples
normal_samples = stats.norm.rvs(loc=mu_norm, scale=sigma_norm
   , size=size_norm)
print("First 10 simulated values:")
print(normal_samples[:10])

# Theoretical PDF
x_values = np.linspace(mu_norm - 4*sigma_norm, mu_norm + 4*
    sigma_norm, 200)
pdf_values = stats.norm.pdf(x_values, loc=mu_norm, scale=sigma_norm)

plt.figure(figsize=(10, 6))
plt.hist(normal_samples, bins=30, density=True, alpha=0.6, color='#74c0fc',
         label='Simulated Data')
plt.plot(x_values, pdf_values, color='#f76707', linewidth=2, label='Theoretical PDF')

plt.xlabel("Value")
plt.ylabel("Density")
plt.title(f"Normal Distribution (  ={mu_norm},   ={sigma_norm})     Simulation vs. Theory")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 6: Simulating and plotting Normal distribution

This comparison demonstrates how the empirical data aligns closely with the theoretical curve, confirming that simulated samples accurately represent the Normal distribution. Such simulations are essential for assessing how well theoretical models describe real-world data.