Introduction to Data Science

- Introduction to Probability -

Mauricio Molina

Keio University, Faculty of Economics

May 21, 2025

Contents

Mathematical Definition of Probability

- Probability space $(\Omega, \mathcal{F}, P)$
  - Ω: Sample space
  - $\mathcal{F}$: σ-algebra of events
  - P: Probability measure

Definition (Axioms of Probability)

Given an event $A \in \mathcal{F}$, a non-negative function $P(\cdot)$ is a probability measure, if

1. $P(A) \geq 0, \quad \forall A \in \mathcal{F}$
2. $P(\Omega) = 1$
3. $P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i) \quad (A_i \text{ disjoint})$

Complement of an Event

- **Definition.** The *complement* of an event E, denoted E^c, is the event that E does *not* occur.

$$E^c = \{\omega \in S : \omega \notin \Omega\}.$$

- Since E and E^c together exhaust the sample space Ω,

$$P(E) + P(E^c) = 1.$$

Example: Dice Toss

$$\Omega = \{1, 2, 3, 4, 5, 6\}, \quad E = \{\text{even outcome}\} = \{2, 4, 6\}.$$

Then the complement is

$$E^c = \{\text{not even}\} = \{1, 3, 5\},$$

Union and Intersection of Events

- **Union:**
$$A \cup B = \{\omega \in \Omega : \omega \in A \text{ or } \omega \in B\}.$$
- **Intersection:**
$$A \cap B = \{\omega \in \Omega : \omega \in A \text{ and } \omega \in B\}.$$

Example: Dice Toss

$$\Omega = \{1, 2, 3, 4, 5, 6\},$$

$$A = \text{Getting a result lesser than } 4 = \{1, 2, 3\},$$

$$B = \{\omega \text{ odd}\} = \{1, 3, 5\}.$$

$$A \cup B = \{1, 2, 3, 5\}, \quad A \cap B = \{1, 3\}.$$

Probability Values for Events

Consider the sample space $\Omega = \{1, 2, 3, 4, 5, 6\}$ and events

$$A = \{1, 2, 3\}, \quad B = \{1, 3, 5\}.$$

- $P(\{i\}) = \frac{1}{6}$ for each $i = 1, 2, \ldots, 6$.
- $P(\emptyset) = 0$.
- $P(A \cap B) = \frac{1}{3}$. Probability of A **and** B ocurring.
- $P(A \cup B) = \frac{5}{6}$. Probability of A **or** B ocurring.

Statistical Definition of Probability (I)

The probability of an event A is a measure of our *belief* that A will occur. If an experiment is performed n times, the *relative frequency* of A is

$$\text{Relative frequency of } A = \frac{\text{frequency of } A}{n},$$

where "frequency of A" is the number of trials in which A occurred.

Statistical Definition of Probability (II)

If you repeat the experiment more and more times ($n \to \infty$), you effectively sample the entire population. In that limit, the relative frequency defines the probability of A:

$$P(A) = \lim_{n \to \infty} \frac{\text{frequency of } A}{n}.$$

Empirical Probability via Simulation

Example: Estimate $P(i)$ by simulation

```
import numpy as np

# Possible die faces
dice_data = np.array([i for i in range(1,7)])

# Number of trials
calc_steps = 1000

# Simulate 1000 rolls
dice_rolls = np.random.choice(dice_data, calc_steps)

# Compute relative frequencies
for i in range(1, 7):
 p = len(dice_rolls[dice_rolls == i]) / calc_steps
 print(f"Probability of getting {i}: {p:.3f}")
```

Python `function` — General Form

Template

```
def function_name(param1, param2=default):
    """Short docstring explaining what the function does."""
    # 1. (optional) set-up or calculations
    # 2. main computation
    result = ...
    return result
```

- `def` keyword starts the definition.
- **Parameters** receive the data the function needs.
- A short **docstring** explains purpose & arguments.
- `return` sends a value back to the caller.

Example Function: coin_tosses(n)

Code

```
import random

def coin_tosses(n):
 """Return a list with n coin-toss results ('H' or 'T')."""
 return [random.choice(['H', 'T']) for _ in range(n)]
```

Demo

```
>>> coin_tosses(5)
['T', 'H', 'H', 'T', 'H']
```

Python `class` — General Form

Class structure

```
class ClassName:
 """Short one-line description."""

 def __init__(self, arg1, arg2):
  # store initial state
  self.arg1 = arg1
  self.arg2 = arg2

 def method_name(self, x):
  # behaviour that uses the internal state
...
```

- `__init__` runs once when we create an object.
- Attributes (`self.arg1`, `self.arg2`) keep the object's data.
- Methods (`method_name`) define its behaviour.

Example Class: Dice

Defining a dice class

```
import random

class Dice:
 """A fair six-sided die."""

 def __init__(self):
  self.faces = (1, 2, 3, 4, 5, 6)

 def roll(self):
  """Return one random face value."""
  return random.choice(self.faces)
```

Using the class

```
>>> d = Dice()    # create an object
>>> d.roll()      # roll once
4
```

Example: roll a die n times

```
def roll_many(n):
 """Return a list with n dice rolls."""
 dice = Dice()
 return [dice.roll() for _ in range(n)]

roll_many(5)
[6, 2, 3, 5, 1] #Output
```

Conditional Probability

- For independent events E and F:

$$P(E \cap F) = P(E)\,P(F)$$

- If F is not impossible $(P(F) \neq 0)$, we define the **conditional probability** of E given F:

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)}$$

- Rearranging gives the **product rule** (always true):

$$P(E \cap F) = P(E \mid F)\,P(F)$$

- When E and F are independent, $P(E \mid F) = P(E)$; knowing F tells us nothing new about E.

Two-Child Example: Basic Probabilities

Consider a family with two children (unknown gender)

- Assume each child is equally likely to be a **boy (B)** or **girl (G)**, and the two genders are independent.
- All four gender pairs are equally likely:

$$\{\text{BB},\ \text{BG},\ \text{GB},\ \text{GG}\}$$

- Hence

$$P(\text{no girls}) = \tfrac{1}{4}, \quad P(\text{one girl, one boy}) = \tfrac{1}{2}, \quad P(\text{two girls}) = \tfrac{1}{4}.$$

Conditional on "Older Child Is a Girl"

$$\text{Event } B = \text{"both children are girls"}, \qquad \text{Event } G = \text{"older child is a girl"}.$$

$$P(B \mid G) = \frac{P(B \cap G)}{P(G)} = \frac{P(B)}{P(G)} = \frac{\frac{1}{4}}{\frac{1}{2}} = \frac{1}{2}.$$

- Here $B \cap G$ is the same as B, because if both children are girls, the older one is necessarily a girl.
- Result lines up with intuition: once we know the first child is G, the second is equally likely to be G or B.

Conditional on "At Least One Girl"

Event $L =$ "at least one child is a girl".

$$P(B \mid L) = \frac{P(B \cap L)}{P(L)} = \frac{P(B)}{P(L)} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}.$$

- Knowing only "at least one girl" leaves three equally likely pairs: BG, GB, GG.
- Only one of those three is GG, so the chance is $\frac{1}{3}$.
- Intuition: families with one girl + one boy are twice as common in this subset.

Python Setup: Enum + Helper

Code

```
import enum, random

# An Enum is a typed set of named values.  It makes
# the code more descriptive and less error-prone.

class Kid(enum.Enum):
 BOY  = 0
 GIRL = 1

def random_kid() -> Kid:
 """Return Kid.BOY or Kid.GIRL with equal prob."""
 return random.choice([Kid.BOY, Kid.GIRL])
```

- `Kid` gives us readable labels instead of 0 / 1.
- `random_kid()` encapsulates one fair coin-flip.

Monte-Carlo Check of Conditional Probabilities

Code

```
random.seed(0)
both_girls   = 0
older_girl   = 0
either_girl  = 0

for _ in range(1000):
 younger = random_kid()
 older   = random_kid()

 if older == Kid.GIRL:
  older_girl += 1
 if older == Kid.GIRL and younger == Kid.GIRL:
  both_girls += 1
 if older == Kid.GIRL or younger == Kid.GIRL:
  either_girl += 1

print("P(both | older): ", both_girls / older_girl)   #  # P(both | older) = 1/2
print("P(both | either):", both_girls / either_girl)  #  # P(both | either) = 1/3
```

Bayes's Theorem

- We want $P(E \mid F)$ but only know $P(F \mid E)$.
- Start from the definition of conditional probability:

$$P(E \mid F) = \frac{P(E \cap F)}{P(F)}$$

- Write $P(E \cap F)$ as $P(F \mid E)\,P(E)$:

$$P(E \mid F) = \frac{P(F \mid E)\,P(E)}{P(F)}$$

- Split F into the mutually exclusive events $(F \cap E)$ and $(F \cap \neg E)$:

$$P(F) = P(F \mid E)\,P(E) + P(F \mid \neg E)\,P(\neg E)$$

- Result—**Bayes's Theorem**:

$$P(E \mid F) = \frac{P(F \mid E)\,P(E)}{P(F \mid E)\,P(E) + P(F \mid \neg E)\,P(\neg E)}$$

Medical-Test Example

Imagine a certain disease that affects 1 in every 10,000 people. And imagine that there is a test for this disease that gives the correct result ("diseased" if you have the disease, "non-diseased" if you don't) 99% of the time. What does a positive test mean? Let's use T for the event "your test is positive" and D for the event "you have the disease." Then Bayes's Theorem says that the probability that you have the disease, conditional on testing positive, is:

$$\text{Let } D = \text{“have the disease”}, \quad T = \text{“test is positive”}.$$

$$\begin{aligned} P(D) &= 1/10\,000 = 0.0001 \\ P(\neg D) &= 0.9999 \\ P(T \mid D) &= 0.99 \quad \text{(sensitivity)} \\ P(T \mid \neg D) &= 0.01 \quad \text{(false-positive rate)} \end{aligned}$$

Apply Bayes's Theorem:

$$P(D \mid T) = \frac{0.99 \times 0.0001}{0.99 \times 0.0001 + 0.01 \times 0.9999} = \boxed{0.0098 \;\approx (\; 0.98\%)}.$$

- Fewer than 1 % of positive testers actually have the disease.
- Doctors often over-estimate this probability by orders of magnitude.

Intuitive "Population of One Million" View

- Imagine $1\,000\,000$ randomly tested people.
- **Disease present:**
  - Expected cases: $1\,000\,000 \times 0.0001 = 100$.
  - True positives: $100 \times 0.99 = 99$.
- **Disease absent:**
  - Healthy individuals: $999\,900$.
  - False positives: $999\,900 \times 0.01 \approx 9\,999$.
- Total positive tests: $99 + 9\,999 = 10\,098$.
- Therefore

$$P(D \mid T) = \frac{99}{10\,098} \approx 0.0098.$$

Only about 1 in 100 positive results indicates a real disease case—exactly the Bayes calculation from the previous slide.

What If the Same Person Tests Positive Twice?

Assume the **second test** is independent of the first and has the same **sensitivity** 0.99 and **false–positive rate** 0.01.

$$D = \text{“has disease”}, \qquad T_1^+ = \text{“first test positive”},\ T_2^+ = \text{“second test positive”}.$$

We want the posterior

$$P(D \mid T_1^+, T_2^+).$$

Apply Bayes again (or combine both tests at once):

$$P(D \mid T_1^+, T_2^+) = \frac{P(T_1^+, T_2^+ \mid D)\, P(D)}{P(T_1^+, T_2^+ \mid D)\, P(D) \;+\; P(T_1^+, T_2^+ \mid \neg D)\, P(\neg D)}.$$

Because the two tests are independent conditional on D (or $\neg D$):

$$P(T_1^+, T_2^+ \mid D) = 0.99^2, \qquad P(T_1^+, T_2^+ \mid \neg D) = 0.01^2.$$

Numerical Posterior After Two Positives

$$
\begin{aligned}
P(D) &= 0.0001, \\
P(\neg D) &= 0.9999, \\
0.99^2 &= 0.9801, \\
0.01^2 &= 0.0001.
\end{aligned}
$$

$$
P(D \mid T_1^+, T_2^+) = \frac{0.9801 \times 0.0001}{0.9801 \times 0.0001 \ + \ 0.0001 \times 0.9999} \ = \boxed{0.495 \approx \ (\ 49.5\%)}.
$$

- One positive test $\Rightarrow$ probability $\approx$ **0.98 %**.
- Two independent positives $\Rightarrow$ probability jumps to nearly **50 %**.

Quick Simulation to Verify

Python snippet

```
import random, math

def experiment(trials=1000000):
 have_disease = 0
 two_pos_and_disease = 0
 two_pos = 0

 for _ in range(trials):
  # true disease status
  d = random.random() < 0.0001

  # two independent tests
  t1 = (random.random() < (0.99 if d else 0.01))
  t2 = (random.random() < (0.99 if d else 0.01))

  if t1 and t2:
   two_pos += 1
   if d:
    two_pos_and_disease += 1

return two_pos_and_disease / two_pos

print(experiment())   # around 0.495
```

Monte-Carlo matches the analytic result: about 49–50 % of patients with *two* positive tests truly have the disease.

Assignment 6 :

Solve the following three problems in your Jupyter Notebook. Show the code (if any) and a short explanation for each answer. Upload the completed `.ipynb` file to K-LMS by next week's Tuesday at midnight.

Q1: A lottery drum contains **1 000 tickets**, of which **100 are winners**. A-san draws one ticket first, then B-san draws one ticket from the remaining 999. No ticket is returned to the drum.

1. Compute the probability that A-san's ticket is a winner.
2. Compute the probability that B-san's ticket is a winner.
3. Compute the probability that *both* A-san and B-san draw winning tickets.
4. Verify your answers with a short Monte-Carlo simulation ($\geq$ 100,000 trials).

Show all reasoning or code in your notebook. Comment briefly on how the probabilities would change if B-san drew *before* A-san.

Q2: Create a Python class `Dice` that models a fair six-sided die.

1. Include a method `roll()` that returns a single random face value.
2. Add a method `frequency(n)` that rolls the die n times (user input) and returns a dictionary or list with the count of each face.
3. Add a method `histogram(n)` that performs `n` rolls and draws a bar chart of the observed frequencies (you may use `matplotlib.pyplot`).

Demonstrate your class by calling `histogram(10 000)` and explaining in one sentence what you see.

Q3: Consider the rare-disease example from the slides, but *now* suppose the test is even more accurate at detecting the disease—returning a true positive 99.99 % of the time—while its false-positive rate (reporting "positive" when the patient is actually healthy) rises to 5 %. The disease remains very rare, with prevalence $P(D) = 0.0001$.

Answer the following:

1. Compute $P(D \mid T^+)$: the proportion of positive test results that truly indicate disease under the new parameters.
2. Compute $P(D \mid T_1^+, T_2^+)$: the posterior probability after *two* independent positive tests.
3. In two–three sentences, explain why a higher false-positive rate can overwhelm the improved sensitivity when the disease is rare.
4. Verify your analytic results with a Monte-Carlo simulation of at least one million patients.