📊 Module 02: Probability Basics for Machine Learning

Topics Covered

- Events, sample space, and basic probability rules
- Conditional probability and independence
- Bayes' theorem (intuitive and formula-level) and base-rate effects
- Sensitivity, specificity, false positive/negative rates, PPV, and NPV
- Class imbalance and its impact on interpreting errors

What Is Probability and Why It Matters for ML

Probability is the mathematics of uncertainty — it tells us how likely an event is to occur.
Machine Learning relies on probability because models don't make absolute decisions; they estimate the likelihood of outcomes based on data.

Examples:

- Probability that an email is spam = 0.9 → model is 90 % confident it's spam.
- Probability that a loan will default = 0.2 → 20 % risk of default.
- Probability that a patient has diabetes = 0.6 → moderate chance requiring further tests.

Probability lets ML systems **handle noise, uncertainty, and incomplete information** while still making reasoned predictions.

1️⃣ Events, Sample Space, and Basic Rules

- **Event:** A specific outcome (e.g., rolling an even number on a die).
- **Sample Space:** All possible outcomes (e.g., {1, 2, 3, 4, 5, 6}).
- **Rule:** P(Event) = Favorable outcomes / Total outcomes.

Example: P(Even) = 3 / 6 = 0.5.

In ML, an event could be "customer buys product" or "model predicts positive."

2️⃣ Conditional Probability and Independence

Conditional probability is the probability of A *given that* B has occurred.

Example: P(Person buys ice cream | Hot day).
It answers how one factor changes the likelihood of another.

In ML, conditional probability helps find relationships like
P(Class | Features) — the chance of a label given input data.

Independence: Events A and B are independent if P(A | B) = P(A).

Naïve Bayes classifier assumes this independence for simplicity, even though real data rarely follows it perfectly.

3️⃣ Bayes' Theorem and Base-Rate Effects

Bayes' Theorem lets us **update our beliefs when new evidence appears.**
Formula: $P(A \mid B) = [P(B \mid A) \times P(A)] / P(B)$

Intuition:

- Start with a belief (prior).
- Observe new data (likelihood).
- Update belief (posterior).

Example: A disease test is 99 % accurate, but if the disease affects only 1 in 10 000 people, most positive results are still false alarms. That's the **base-rate effect** — even accurate tests can mislead when the event is rare.

In ML, this teaches us to adjust our expectations based on how common each class is.

4️⃣ Sensitivity, Specificity, and Error Rates

These concepts come from the **confusion matrix**, a key tool for evaluating models.

- **Sensitivity (True Positive Rate):** Correctly detecting positives.
- **Specificity (True Negative Rate):** Correctly detecting negatives.
- **False Positive:** Model predicts positive when it's actually negative.
- **False Negative:** Model predicts negative when it's actually positive.
- **PPV (Positive Predictive Value):** How often a positive prediction is correct.
- **NPV (Negative Predictive Value):** How often a negative prediction is correct.

These metrics help decide what matters most in a task — catching every positive case (sensitivity) or avoiding false alarms (specificity).

5️⃣ Class Imbalance and Its Impact

Many real-world datasets are **imbalanced** — one class dominates.

Example: In 10 000 transactions, only 50 are fraudulent (0.5 %).
A model that always predicts "no fraud" is 99.5 % accurate — and completely useless.
That's why accuracy alone isn't enough.
We use probabilistic metrics like PPV, NPV, and Bayesian reasoning to judge how well a model really handles minority classes.

Understanding these concepts prepares you for later modules on **model evaluation, precision, recall, and F1-score.**

🎯 Module Goal

By the end of Module 02, you should be able to:

- ☑ Explain probability concepts in ML context.
- ☑ Apply Bayes' Theorem intuitively.
- ☑ Interpret confusion matrix metrics (sensitivity, specificity, PPV, NPV).
- ☑ Recognize how class imbalance distorts model accuracy.

Remember: Machine Learning is not about certainty — it's about **making smart decisions under uncertainty.**

Understanding Probability for Machine Learning – Events, Sample Space, and Basic Rules

In this lesson, we'll explore:
- What are events
- What is a sample space
- The basic rules of probability

By the end, you'll see how this simple math forms the foundation of almost every ML algorithm — from spam detection to self-driving cars.

🎲 1. What is Probability?

Probability is simply a **measure of uncertainty**.
It tells us *how likely* something is to happen.

Example:
- If you flip a fair coin, there's a **0.5 chance** (or 50%) it lands on heads.
- In ML, we might say there's a **0.9 probability** that an email is spam — meaning the model is 90% confident.

In short:

$$P(Event) = \frac{\text{Number of favorable outcomes}}{\text{Total number of outcomes}}$$

🎯 2. Sample Space

The **sample space**, often denoted as **S**, is the set of *all possible outcomes* of an experiment.
Let's start simple:

- When you toss a coin:
  S = {Heads, Tails}
- When you roll a die:
  S = {1,2,3,4,5,6}

In ML, we can think of the sample space as the **universe of all possible data points**.

Example:
- Suppose we're predicting if a student passes or fails based on study hours and attendance.
  The sample space includes **all combinations** of study hours, attendance, and outcomes (pass/fail).

👉 **Key takeaway:**
The sample space defines the world your probability model lives in. Just like your dataset defines the world your ML model learns from.

💡 3. Events

An **event** is a subset of the sample space — something we care about happening.
Example:

- Rolling an even number → E = {2,4,6}
- Getting heads → E = {H}

In ML terms:

- Event = "The model predicts correctly"
- Event = "The data point belongs to class A"
- Event = "The user clicks on the ad"

Events can be:
- **Simple event:** one outcome (like rolling a 3)
- **Compound event:** multiple outcomes (like rolling an even number)
- **Complementary event:** everything else (not rolling an even number)

We often use notation like:

$$P(E^c) = 1 - P(E)$$

That's just saying the probability of "not E" equals one minus the probability of "E".

4. Basic Probability Rules

Let's cover the three essential rules every ML student needs to know.

Rule 1: The Probability Range Rule
$0 \leq P(E) \leq 1$

A probability can never be negative or exceed 1.

If your ML model gives a probability greater than 1, something's definitely wrong with your math or scaling.

Rule 2: The Complement Rule
P(E) + P(E ^ c) = 1

Example:
If a model predicts that a user **will buy** a product with probability 0.7,
then P(Not Buy)=1−0.7=0.3.

This is exactly how classification models like **logistic regression** work. They compute one probability, and the rest follows automatically.

Rule 3: Addition Rule (for "OR" Events)
If A and B are **mutually exclusive** (can't happen together):
P(A or B)=P(A)+P(B)

Example:

- Rolling a 2 or 3 on a die

$$= \frac{1}{6} + \frac{1}{6} = \frac{2}{6}$$

If they **can** happen together, use the **general addition rule**:
P(A or B)=P(A)+P(B)−P(A and B)

In ML:

- Suppose A = "User clicks on an ad"
- B = "User buys the product"
  Some users might do both — that's P(A and B)P(A \text{ and } B)P(A and B).

We subtract that overlap to avoid double-counting.

Rule 4: Multiplication Rule (for "AND" Events)
If A and B are **independent**:
P(A and B) = P(A) × P(B)

Example:

- Probability of getting two heads in a row
  = 0.5 × 0.5 = 0.25

In ML, this independence assumption appears in models like **Naive Bayes**, where features are assumed to be independent given the class label.

That's why it's called *Naive* — but surprisingly, it works well in practice.

Bonus: Conditional Probability
This one's key for ML:

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

It reads: "Probability of A given B has occurred."

In ML:

- P(Spam | Has the word 'offer')
- The model learns these conditional probabilities from training data.

Conditional probability is the heart of **Bayes' Theorem**, which we'll explore later when we discuss **Naive Bayes classification**.

🤖 Section 5: Mini Real-World Use Cases

Let's connect this to real ML applications — so it doesn't just feel like math.

📧 Example 1: Spam Filter

When you get an email, your ML model might check:

- Event A: It contains the word "Free"
- Event B: It contains "Offer"

It calculates probabilities like:

- P(Spam | Free)=0.8
- P(Spam | Offer)=0.9
- P(Spam | Free and Offer)=0.95

That's the exact same math — just applied to real data!

💉 Example 2: Disease Detection

In healthcare, models use probability to estimate the risk of disease.
For example:

- Event A: Test result is positive
- Event B: Patient actually has the disease
  The model calculates how likely B is given A — that's where **Bayes' theorem** (coming next!) steps in.

🚀 6. Wrap Up

To recap:

- **Sample space** → all possible outcomes
- **Event** → the subset we care about
- **Probability** → measure of likelihood
- **Basic rules** → keep your probabilities logical and consistent

In ML, probability powers everything — from **classification**, **Bayesian inference**, **stochastic gradient descent**, to even **generative models**.

Next up, we'll use these concepts to understand **conditional probability and Bayes' theorem**, which directly connect to algorithms like **Naive Bayes** and **Probabilistic AI**.

Conditional Probability and Independence

🧠 Section 1: Recap of Probability

If you roll a fair die, the chance of getting a 4 is **1 out of 6** — simple.

But real-world data is not that clean. In ML, we often deal with situations like:
"What's the chance a person buys pizza if it's Friday night?"
That's no longer simple probability — it's *conditional.*

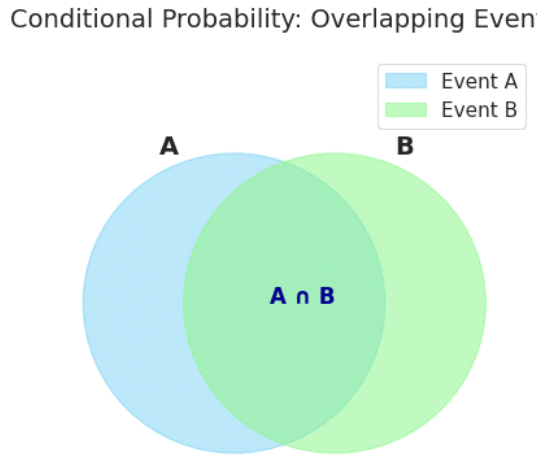
📓 Section 2: What is Conditional Probability?

Conditional probability measures how likely one event is *given* that another event has occurred.

Mathematically:

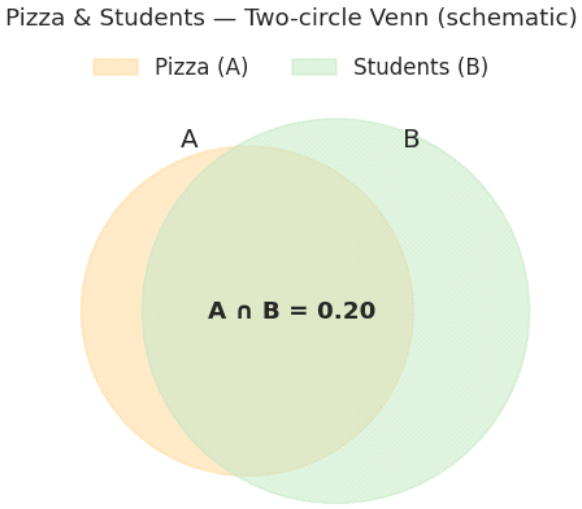
$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Read it as:
"The probability of A given B equals the probability of both A and B happening, divided by the probability of B."



💡 **Example:**
Suppose in a dataset of 1,000 people:

- 300 like pizza → P(A)=0.3
- 500 are students → P(B)=0.5
- 200 are students who like pizza → P(A∩B)=0.2



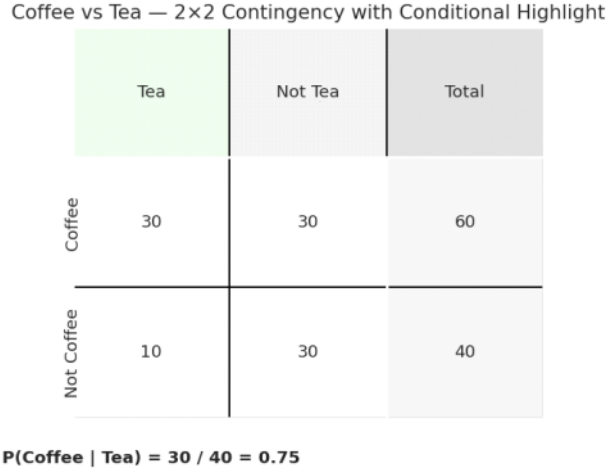
Then:

$$P(A|B) = \frac{0.2}{0.5} = 0.4$$

That means *if you know someone is a student, the probability they like pizza jumps from 0.3 to 0.4.*

Now, let's visualize this.
Imagine a class of 100 students.

- 60 of them like coffee.
- 30 of them like both coffee and tea.



Now, if I ask — "What's the probability that a student likes coffee **given that** they already like tea?"

We're focusing only on the group of tea lovers — maybe that's 40 students.
Out of those 40, 30 like coffee too.

So:
P(Coffee|Tea)=30/40=0.75

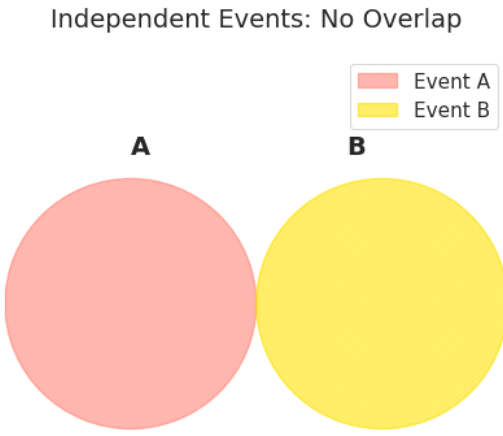
That's **75%**.

🤝 Section 3: Independence

If knowing one event doesn't change the probability of the other, they're **independent**.

Mathematically:
$P(A|B) = P(A)$

and equivalently,
$P(A \cap B) = P(A) \times P(B)$



💡 **Example:**
If gender and pizza preference are unrelated in your data, then whether someone's male or female doesn't change the probability of liking pizza.

So, if P(LikesPizza) = 0.6 and P(WearsGlasses) = 0.3,

then if these are independent,
$P(Both) = 0.6 \times 0.3 = 0.18$

— meaning 18% of people are expected to both like pizza and wear glasses, *just by chance*.

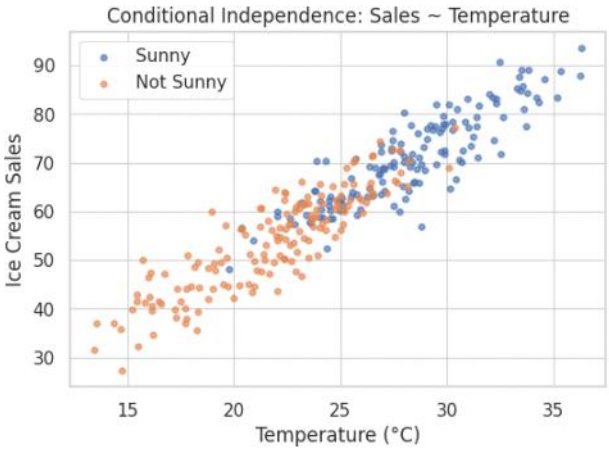
In ML, **independence** helps simplify complex problems.
For example, **Naive Bayes classifier** assumes that all features are independent given the class — even though they aren't truly independent — but it still works surprisingly well.

🧩 Section 4: Conditional Independence

Two events A and B are *conditionally independent* given C if:
$P(A|B,C) = P(A|C)$

That means once you know C, B gives no extra information about A.

💡 **Example:**
Suppose you're predicting whether someone will buy ice cream (A) based on temperature (C) and whether it's sunny (B). If you already know the temperature, the "sunny" info adds almost nothing — so A and B are conditionally independent given C.



In many ML models (like **Naive Bayes**, **Bayesian Networks**, or **Hidden Markov Models**), we assume certain variables are conditionally independent to simplify calculations.
If A and B were *always* independent, we could say
P(A,B)=P(A)P(B)

But real-world data often has dependencies. So instead, we assume independence **given** something else (a "context" variable like C).

Let's say you have:

- **A:** Whether a student passes the exam
- **B:** Whether they studied with friends
- **C:** Total hours studied

If you already know the total study hours (C), then whether they studied alone or with friends (B) adds very little extra info about passing (A).
So empirically you'd find:

$$P(\text{Pass} \mid \text{StudiedWithFriends}, \text{Hours}) \approx P(\text{Pass} \mid \text{Hours})$$

That's **conditional independence** in action.

🤖 Section 5: Why It Matters in ML

Conditional probability allows models to make predictions under uncertainty.
Examples:

- **Spam detection:** P(Spam | Contains "lottery")
- **Medical diagnosis:** P(Disease | Test Positive)
- **Recommender systems:** P(User likes item | User liked similar items)

When combined with independence assumptions, we get fast, interpretable models like **Naive Bayes**, still used in text classification today.

🎯 Section 6: Wrap-Up

So, to summarize:

- Conditional probability tells us *how one event affects another.*
- Independence means *no effect.*
- Conditional independence helps simplify large models.
  These ideas form the foundation of many ML algorithms that learn from *patterns and uncertainty.*

Next time your model makes a "smart guess," remember — it's all probability at play.

Bayes' Theorem & The Base-Rate Effect — How Machines Update Their Beliefs

1. Intro

In this lesson, we'll see:

What Bayes' Theorem really means — without getting lost in math.
How the **base-rate effect** can completely flip our intuition.
How this shows up in real-world ML problems like spam and fraud detection.

2. Concept Explanation

Okay, so what's the heart of Bayes' Theorem?
It's a rule that helps us **update our belief** when new information appears.
We start with what we *already* believe — the **prior probability**.
Then comes **new evidence** — something we observe.
Bayes' theorem combines both to produce an updated belief — the **posterior probability**.

Let's put it in formula form:

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

Now, Let's decode it in plain English.

- P(A)— the **prior** — what we thought *before* seeing anything.
- P(B|A)— the **likelihood** — how likely we'd see the evidence if our belief were true.
- P(B)— the **evidence** — how common that observation is in general.
- P(A|B)— the **posterior** — our *new* belief *after* seeing the evidence.

Scenario: "The Rain and the Traffic Jam"

You're getting ready for work and you notice **heavy traffic** outside your window. You wonder — is it because it's **raining**?
We'll translate this into Bayes terms:

- **A:** It's raining.
- **B:** There's a traffic jam.

🤔 Now decode it in plain English using Bayes' pieces:

- **P(A)** → *Prior*:
  Before looking outside, you check your weather app. It says there's a **30% chance of rain** today.
  That's your *prior belief* — how likely it is to rain before seeing any evidence.
- **P(B|A)** → *Likelihood*:
  If it **is** raining, you know there's usually a **90% chance of a traffic jam** because everyone drives slower and accidents increase.
- **P(B)** → *Evidence*:
  How often do you see traffic jams in general, regardless of weather? Let's say **40% of mornings** have traffic even without rain.
- **P(A|B)** → *Posterior*:
  Now that you **see traffic**, what's the updated chance it's actually raining?
  Using Bayes' Theorem, you'd find that it's **higher than 30%** — maybe around **60–70%**, depending on the exact numbers.

So Bayes' theorem is like:

"How much should I trust this new clue, given what I already know about the world?"

In ML, that's exactly what models do when they get new data — they update their predictions based on both prior knowledge and new evidence.

3. Mini Example 1 — Simple Intuition

Imagine a disease that affects only **1 in 1000 people** — that's 0.1%.
Now there's a test that's **99% accurate**.
You take the test and it's positive. What's the chance you actually have the disease?
Our gut says, "99%!"
But let's think carefully.

Only 1 person in 1000 truly has it. Out of those 1000,

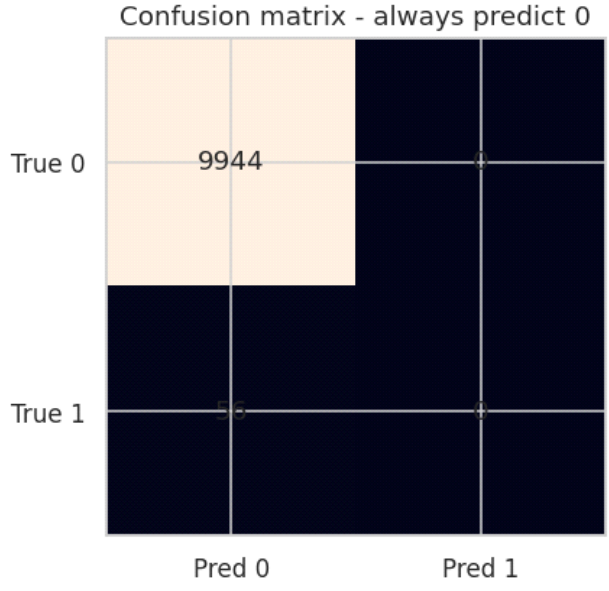
- 1 person will test positive *correctly*.
- But 10 others (1% of 999) will test positive *falsely*.

So total positive tests = 1 true + 10 false = 11.
Only 1 of those 11 actually has the disease.
So the true chance = **1/11 = about 9%**.

That's the **base-rate effect** — when we ignore how rare or common something is, we massively overestimate probabilities.

In short:

Even with accurate data, ignoring base rates can fool both humans and machines.



Accuracy: 99.44%

Real-Life Example: The Police Car and the Sports Car

Imagine you're driving home at night and you see a **red sports car speeding past you**. A friend says,
"Wow, that's probably a stolen car!"
Let's unpack why this thought is a **classic base-rate mistake**.

Step 1: What we know

- In your city, only **1 in 10,000 cars** is actually stolen. →
  That's the base rate (very rare).
- But, let's say **80% of stolen cars** are sports cars.

- And **10% of regular (non-stolen) cars** are also sports cars — people just like fast cars.

Now you see one red sports car.
What are the chances it's actually stolen?

Step 2: The gut reaction (the mistake)

Your friend focuses on the clue — "it's a sports car."
Because *most* stolen cars are sports cars, they assume it's *likely* stolen.
That feels logical… but it's wrong.

Step 3: The Bayes-style reality check

Let's crunch quick mental math.
Out of **10,000 cars**:

- **1** is stolen.
- **9,999** are not stolen.

Among those:

- Stolen sports cars = 80% of 1 = **0.8 car**
- Non-stolen sports cars = 10% of 9,999 = **about 1,000 cars**

So there are **~1,000 sports cars** on the road, but only **0.8** of them are stolen.

👉 **Probability that a random sports car is stolen:**
$0.8/(0.8+1000) \approx 0.08\%$
That's less than **1 in 1,000**, even though "most stolen cars are sports cars."

💡 The point

The friend ignored how rare stolen cars are in the first place — the **base rate**.
Even strong clues (like "sports car") don't matter much when the event itself is extremely uncommon.

In ML terms

This is exactly what happens in **fraud detection**, **rare disease diagnosis**, or **anomaly detection**.
Even when a model is great at spotting "suspicious" patterns, most flagged cases are false alarms — because the *real thing* is rare.
So:

> The base-rate effect reminds us that **even strong evidence can mislead you if the underlying event is rare**.

4. Mini Real-World Use Case

Let's connect this to ML.
A spam classifier also uses Bayes' logic under the hood.
It knows that most emails are normal — that's its **prior**.
Then it sees words like "prize" or "urgent." That's the **evidence**.

It calculates:

$$P(\text{Spam} \mid \text{Word}) \propto P(\text{Word} \mid \text{Spam}) \times P(\text{Spam})$$

If the word "lottery" appears mostly in spam, then P(Word | Spam) is high — that boosts the **posterior probability** of being spam.
But if the base rate of spam is low (say, 5%), even strong clues may not make it 100% certain.
Same logic applies to **fraud detection** — just because a transaction looks suspicious doesn't mean it's fraud, because frauds are rare.

That's Bayes' Theorem teaching ML models to *stay skeptical*.

5. Summary and Reflection

Let’s wrap this up.

- ☑ **Bayes’ Theorem** helps us update beliefs when new evidence arrives.
- ☑ **Base-rate effect** reminds us that priors — the overall frequency — *matter*.
- ☑ In ML, this thinking powers spam filters, medical diagnosis models, and fraud detectors.
- ☑ Always question your model’s “confidence.” Is it high because of evidence, or just because of data imbalance?

Model Evaluation – Understanding Sensitivity, Specificity, and Predictive Values

1. Intro

Imagine you've built a machine learning model that predicts whether a person has a disease — let's say diabetes. Your model gives results like *Positive* (they have diabetes) or *Negative* (they don't).

But here's the catch — accuracy alone can be misleading. A model can show **95% accuracy** and still fail miserably if the data is imbalanced.

So, how do we really know if our model is good?

That's where **Sensitivity, Specificity, False Positive and False Negative rates, PPV, and NPV** come in.

2. Concept Explanation

Step 1: The Confusion Matrix

Here's the foundation — the **confusion matrix**. It looks like this:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Think of it like a report card of your model — showing what it got right and wrong.

Let's use an easy example.

Out of 100 people tested for diabetes:

10 actually have diabetes.
The model correctly finds 9 of them → **True Positives (TP = 9)**.
It misses 1 person → **False Negative (FN = 1)**.
Among 90 healthy people, it wrongly flags 5 as diabetic → **False Positive (FP = 5)**.
It correctly marks 85 as healthy → **True Negative (TN = 85).**

That's all we need to calculate the rest.

Step 2: Sensitivity / Recall

Sensitivity tells us — *out of all real positive cases, how many did we catch?*

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

In our example: 9 / (9 + 1) = 0.9 → **90% sensitivity.**

👉 High sensitivity means we rarely miss real positives.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

In healthcare, that's crucial — you don't want to miss someone who's actually sick.

Think of a smoke detector. A highly sensitive one will catch even a tiny bit of smoke — it might beep often, but at least it won't miss a fire.

Step 3: Specificity

Specificity asks — *out of all real negatives, how many did we correctly say are negative?*

$$\text{Specificity} = \frac{TN}{TN + FP}$$

In our example: 85 / (85 + 5) = 0.944 → **94.4% specificity.**
That means the model correctly identifies 94% of healthy people as healthy.

Analogy:
Specificity is like a security guard — it decides who *should not* enter.
A high-specificity guard won't wrongly stop innocent people.

Step 4: False Positive & False Negative Rates

Let's flip those formulas:

- **False Positive Rate (FPR)** = FP / (FP + TN)

It's the percentage of healthy people wrongly told they have diabetes.

- **False Negative Rate (FNR)** = FN / (FN + TP)
  It's the percentage of sick people the model failed to detect.

Both should ideally be **low** — but there's always a trade-off.
If you make your model more sensitive (catch more positives), you often increase false positives too.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Step 5: PPV and NPV

Now, let's check if we can trust the model's predictions.

- **PPV (Positive Predictive Value)** or **Precision**

$$\text{PPV} = \frac{TP}{TP + FP}$$

Out of all predicted positives, how many were actually correct?
→ 9 / (9 + 5) = 0.64 or 64%.

- **NPV (Negative Predictive Value)**

$$\text{NPV} = \frac{TN}{TN + FN}$$

Out of all predicted negatives, how many were actually correct?
→ 85 / (85 + 1) = 0.988 or 98.8%.

So, our model is better at saying who's healthy than who's sick — a common issue in imbalanced datasets.

3. Mini Real-World Use Case

I. Job Application Screening AI

Scenario:
Imagine a company using an ML model to filter job applicants for an interview.

- **Positive = Suitable candidate**
- **Negative = Not suitable**

Now,

- A **True Positive** is a genuinely skilled candidate shortlisted.
- A **False Positive** is an unqualified candidate selected — wasting the interviewers' time.
- A **False Negative** is a great candidate rejected — a real loss.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Insight:
A hiring model should have **high sensitivity** if the goal is to *not miss any good candidates*, but **high specificity** if you want to *avoid wasting interview slots* on weak applicants.

II. Autonomous Car – Pedestrian Detection

Scenario:
An ML model detects whether there's a pedestrian on the road.

- **Positive = Pedestrian detected**
- **Negative = No pedestrian**

Now:

- **False Negative (missed pedestrian)** can cause an accident.
- **False Positive (detects ghost pedestrian)** might unnecessarily brake and slow traffic.

Insight:
Here, **high sensitivity** is *critical for safety* — we can tolerate some false alarms, but we can't afford to miss a real person.

III. Face Unlock on Smartphones

Scenario:
Your phone uses facial recognition to unlock.

- **Positive = Authorized face detected**
- **Negative = Unauthorized face**
- **False Positive:** Someone else's face unlocks your phone — security breach.
- **False Negative:** Your phone refuses to unlock for you — annoying but safe.

Insight:
Face unlock systems favor **high specificity** (don't unlock for strangers), even if it means occasionally rejecting your face in poor light.

4. Summary

Let's recap:

- Sensitivity = how well the model catches positives.
- Specificity = how well it avoids false alarms.

- FPR/FNR = the types of mistakes it makes.
- PPV/NPV = how trustworthy its predictions are.

Class Imbalance and Its Impact on Interpreting Errors

Intro

By the end, you'll know:

- What class imbalance is
- Why accuracy can be misleading
- How to evaluate models properly

Concept Explanation

Class imbalance simply means — one class has **many more samples** than another.

You'll see this in real-life ML datasets all the time:

- Spam detection — most emails are *not spam*
- Disease prediction — most people are *not sick*
- Credit card fraud — most transactions are *safe*

So when the dataset is **90% one type and 10% another**, the model starts to get lazy.

It just learns to always predict the majority class and still looks good because it's "right" most of the time.

Accuracy loves imbalance because true negatives dominate the score.
That's why we need other metrics like:

- **Precision:** Out of the predicted positives, how many are correct?
- **Recall (Sensitivity):** Out of the actual positives, how many did we catch?

Visual & Mathematical Explanation

Here's how imbalance fools the confusion matrix.

Think of this 2x2 grid:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP (Correct detection)	FN (Missed case)
Actual Negative	FP (False alarm)	TN (Correct rejection)

When your dataset has mostly negatives — say, 990 "No Fraud" and only 10 "Fraud" —
most cells in the matrix will fill up with **TN (True Negatives)**.

That means accuracy will look high even if the model misses *every* fraud case.
So, we shift focus from *overall accuracy* to *error types.*

In imbalanced problems, **False Negatives hurt more.**
Imagine a cancer test marking sick patients as healthy — that's a critical miss.

Mini Real-World Use Cases

Example 1 – Medical Diagnosis:
You're building a model to detect a rare disease.
Only 2% of patients are sick.
If your model says "healthy" for everyone, you get 98% accuracy — but you've failed all the real patients.

In healthcare, **recall** is more important — catching as many actual positives as possible.

Example 2 – Spam Detection:
In spam filters, it's okay if you sometimes mark a real email as spam (false positive).
It's annoying, but not dangerous.

So here, **precision** might matter more — we don't want to block too many legit emails.
The lesson?

👉 Different problems require different interpretations of model errors.

Summary/ Reflection

Alright, let's wrap this up.
What we learned today:

- **Class imbalance** means one class dominates the dataset.
- **Accuracy** can lie — always check **precision**, **recall**, and **F1-score**.
- **Context matters** — in healthcare, recall may matter more; in spam detection, precision.
- You can handle imbalance with techniques like **resampling**, **class weights**, or **threshold tuning** — which we'll explore later.

Probability Basics for ML (Conclusion)

Concept 1: Events, Sample Space, and Probability Rules

We began with the basics — **events and sample space.**
Think of sample space as *all possible outcomes.*

If you toss a coin, the sample space is {Heads, Tails}.

If your ML model predicts whether an email is spam or not, that's also a sample space — {Spam, Not Spam}.

Events are just specific outcomes inside that space. For example, "Email = Spam."

We also learned the basic **probability rule:**

Probability of any event = number of favorable outcomes ÷ total outcomes.

This idea is the seed that grows into every ML algorithm — because every prediction a model makes is, at its heart, a probability statement.

Concept 2: Conditional Probability & Independence

Next came **conditional probability**, which answers —

"What's the chance of Event A happening given that Event B already happened?"

For example, what's the probability someone will **buy an umbrella given that it's raining**?
That's **P(A|B)** — "A given B."

This is huge in ML.
Say you're building a model that predicts *heart disease.*
You don't just ask, "Does this person have heart disease?"
You ask, "What's the probability of heart disease **given** high blood pressure?"

We also met the idea of **independence** — when knowing one thing doesn't affect the other.

In ML, total independence rarely happens, but assuming partial independence can make computations simpler — that's the logic behind **Naïve Bayes** classifiers.

Concept 3: Bayes' Theorem & Base Rate Effect

Then we unlocked the powerful **Bayes' Theorem.**
Let's say you get a positive result on a rare disease test. Is that result trustworthy?
Bayes' theorem helps us update our belief:

$$P(A|B) = [P(B|A) \times P(A)] / P(B)$$

"How much should we believe in A after seeing evidence B?"

This is exactly what your ML model does every time it makes a prediction — it updates its belief as it sees new data.

We also looked at the **base rate effect** — why ignoring how common or rare something is can mislead you.

A 99% accurate test on a 1-in-1000 condition can still give mostly false alarms.
That's why understanding priors is key in data science.

Concept 4: Sensitivity, Specificity, and Confusion Rates

Next, we explored how probability connects to **model evaluation.**

Every classifier produces four outcomes:

- **True Positive (TP)** – correctly predicts positive.
- **False Positive (FP)** – wrongly predicts positive.
- **True Negative (TN)** – correctly predicts negative.
- **False Negative (FN)** – misses the positive case.

From these, we derived:
- **Sensitivity (Recall)** = TP / (TP + FN)
- **Specificity** = TN / (TN + FP)

In plain English:
Sensitivity checks *how many actual positives the model caught.*
Specificity checks *how many negatives it avoided flagging wrongly.*

These metrics matter a lot in real life.
For example, in medical diagnosis, missing a disease (low sensitivity) is more dangerous than a few false alarms.

In spam detection, false alarms (low specificity) can annoy users.

Each problem needs a balance.

Concept 5: Class Imbalance and Error Interpretation

Finally, we tackled **class imbalance.**

Imagine a dataset where 95% of people don't have a disease, and only 5% do.

If your model just predicts "No Disease" for everyone, it gets 95% accuracy — sounds amazing but actually useless.
That's why we say *accuracy lies.*

In imbalanced data, we rely more on **Precision, Recall, F1-Score**, and **Balanced Accuracy** — because they tell us how fairly the model treats both classes.
Real-world datasets are rarely balanced — fraud, disease, and rare events are always underrepresented. That's why understanding probability helps you design fairer and smarter ML systems.

Summary and Next Steps

Alright, let's wrap up what we've learned in Module 2:

1. Probability is the foundation of all machine learning decisions.
2. Conditional probability and Bayes' theorem help models update beliefs with new data.
3. Sensitivity, specificity, and predictive values help evaluate models fairly.
4. Class imbalance can fool accuracy — so always check deeper metrics.