Introduction to Machine Learning

Agenda

- What is Machine Learning?
- Applications of ML
- Common ML Libraries
- PyTorch vs TensorFlow

What is Machine Learning?

Machine Learning is a subset of Artificial Intelligence that enables systems to learn from data without explicit programming. It improves over time as it gains more data.

Types of ML:

- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**

Applications of Machine Learning

- Make robots recognize faces or learn to walk
- Discover patterns in user logs, financial data, etc.
- Segment customers for personalized marketing
- Recommend products based on user behavior
- Detect fraudulent transactions
- Forecast sales and demand
- Automate and optimize real-world decisions

Common Machine Learning Libraries

- **Scikit-Learn** – Ideal for regression, classification, clustering (beginner-friendly)
- **TensorFlow** – Google's library for large-scale ML
- **Keras** – High-level deep learning API (comes with TensorFlow)
- **PyTorch** – Facebook's deep learning framework
- **Pandas / NumPy** – Data cleaning and handling tools

For deep learning beginners: start with Keras

PyTorch vs TensorFlow

- Both are popular frameworks for deep learning
- TensorFlow offers more production deployment tools
- PyTorch is often preferred for its ease of use and debugging capabilities

How Machines Learn

Not so long ago, if you had picked up your phone, opened ChatGPT and asked it what to do in a certain particular situation, it would have ignored you—and people might have looked at you like you were losing your mind. Today, it's normal. Technology is growing like never before, and Machine Learning (ML) is at the center of this transformation.

Back in the 1990s, one of the earliest real-world applications of ML was the spam filter. Imagine checking your email back then—your inbox would often be flooded with unwanted messages about dubious offers, miracle diets, or suspicious job proposals. Engineers started building systems that could **learn** to recognize these spam patterns based on examples. The idea wasn't to hard-code the rules (like "if email contains the word 'lottery', mark as spam") but instead to let the system **learn from actual spam and non-spam (ham) emails**.

Over time, as users kept marking messages as spam, the system **learned** to identify new spam even before the user had to touch it. That was early machine learning in action.

What Machine Learning Is Not

If you download hundreds of books onto your computer, it doesn't automatically become smart. Your computer doesn't start writing like Shakespeare or summarizing those books for you. It just stores data. That's **not** machine learning. That's just **data storage** and **retrieval**.

Machine Learning kicks in when a computer can **improve at a task** by learning from **data and experience**—without being manually programmed for every scenario.

What Machine Learning Is

Machine Learning is the science (and art) of programming computers so they can **learn from data**.

Here's a slightly more general definition:

> *"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."* — **Arthur Samuel, 1959**

And a more engineering-focused one:

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."* — **Tom Mitchell, 1997**

Let's break that down with the **spam filter** example:

- **Task (T):** Identify whether an email is spam or not
- **Experience (E):** Examples of labeled emails (spam or ham)
- **Performance measure (P):** Accuracy — how many emails are correctly classified

If the system keeps getting better at flagging spam based on past emails, it is said to be **learning**.

The examples used to train the system are called the **training set**, and each individual example is a **training instance** (or sample). The part of the ML system that does the actual learning and prediction is called the **model**. Think of the model as the brain that’s being trained.

Some popular models include:

- **Neural Networks**
- **Random Forests**
- **Support Vector Machines**

Traditional Programming vs Machine Learning

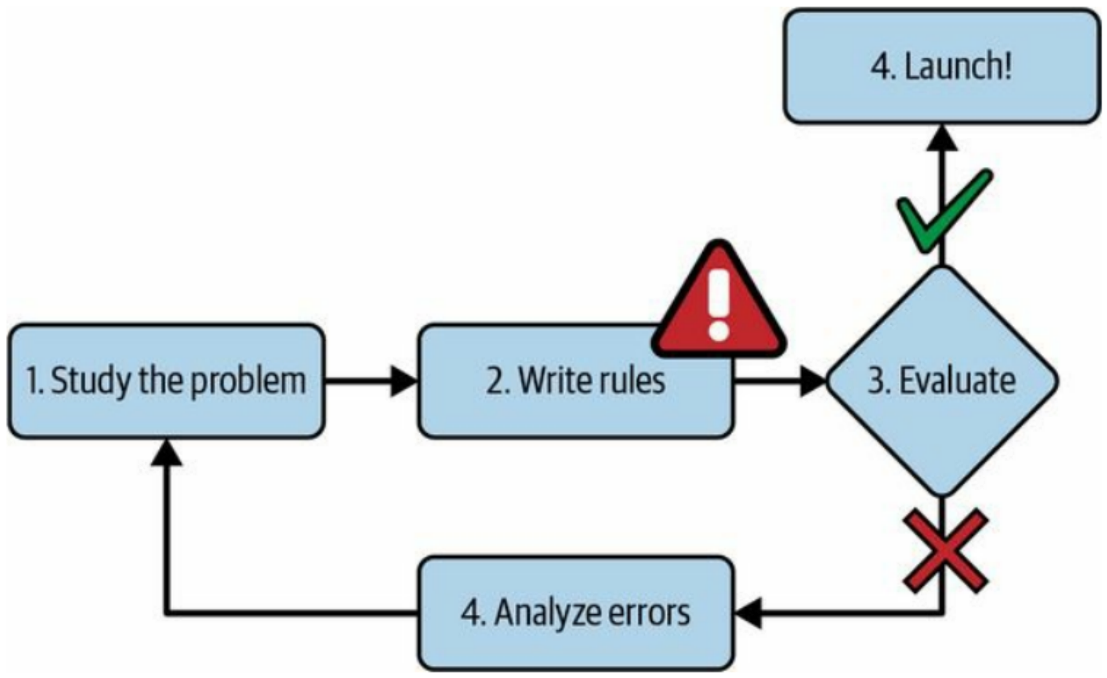
To truly understand what makes ML different, let’s compare it with the traditional approach.

Traditional Approach

In traditional programming:

- A human writes **explicit rules** for the computer to follow.
- Input + Rules = Output

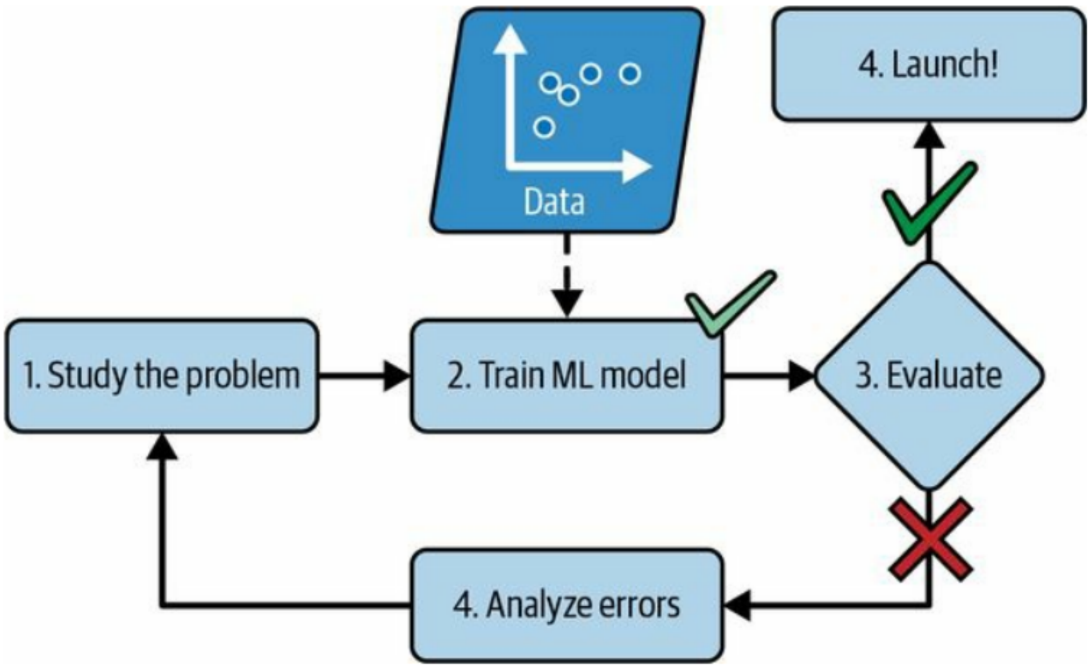
Example: If an email **contains the word** “free money,” mark it as spam. But what if spammers start writing “frëe m0ney”? The rule breaks.



Machine Learning Approach

In the ML approach:

- The computer is given **examples** (data) and learns the rules by itself.
- Input + Output (examples) → Learning → Model (Rules)
- Then: New Input + Model = Prediction



So, instead of hardcoding logic for every possible spam variation, the system **learns patterns** in spammy emails over time and generalizes them to new messages—even ones it hasn't seen before.

Summary

- Just storing data is **not** machine learning.
- ML is about **improving performance with experience**.
- The system **learns patterns** from data rather than following hardcoded rules.
- The **model** is what makes predictions.
- One of the earliest ML use-cases was spam detection.
- Unlike traditional programming, ML systems **adapt** as they see more data.

Installing Scikit-learn

Open your terminal (or use Jupyter Notebook / Google Colab) and run:

```
pip install scikit-learn
```

Or in Google Colab:

```
!pip install scikit-learn
```

Installing Scikit-learn and Making Your First Machine Learning Model

Step 1: Installing Scikit-learn

Open your terminal (or use Jupyter Notebook / Google Colab) and run:

```
pip install scikit-learn
```

Or in Google Colab:

```
!pip install scikit-learn
```

Step 2: Make Your First Machine Learning Model (in 5 minutes!)

Here’s a **“Can we predict if a fruit is an apple or an orange?”** example:

The idea:

We have some fruits.

- Apples are **heavy** and **red**
- Oranges are **lighter** and **orange-colored**

At the first glance, it seems easy to tell them apart, right? You can always compare Orange pixels and red pixels, or check the weight. But as you keep getting more fruits, it becomes harder to write rules for every single fruit and with the messy real world, rules you write starts to break. Let’s train a machine to learn this!

Code Example:

```
from sklearn import tree

# Step 1: Collect Data
# Features: [weight (grams), color (0 = red, 1 = orange)]
features = [
    [150, 0],  # Apple
    [170, 0],  # Apple
    [130, 1],  # Orange
    [120, 1],  # Orange
]
labels = ["apple", "apple", "orange", "orange"]

# Step 2: Train a model
clf = tree.DecisionTreeClassifier()
clf = clf.fit(features, labels)

# Step 3: Predict!
print(clf.predict([[160, 0]]))  # Heavy and red-ish? Probably an apple!
print(clf.predict([[115, 1]]))  # Light and orange? Probably an orange!
```

What we Learned:

- **We just taught a computer to recognize fruits!**
- It looked at the weight and color to decide.
- We didn’t write “if-else” — the computer *learned* by itself!
- We’ve just done **machine learning**!

Step 3: Test Your Model

Now, let’s see how well our model works!

- “Try `[140, 1]` – what does it think?”
- “Try changing colors and weights randomly and test the model!”

Visualize It

You can show how the decision tree looks like:

```python
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plot_tree(clf, feature_names=["weight", "color"], class_names=["apple", "orange"], filled=True)
plt.show()
```